UNIVERSIDAD POLITECNICA DE VALENCIA

PhD Thesis

An agile model-driven method for involving end-users in DSL development

Maria José Villanueva del Pozo

PhD Advisors: Óscar Pastor López

Francisco Valverde Giromé

December 2015, Valencia

An agile model-driven method for involving end-users in DSL development

Un método ágil dirigido por modelos para involucrar a los usuarios finales en el desarrollo de DSLs

Un mètode àgil dirigit por models per a involucrar als usuaris finals en el desenvolupament de DSLs

PhD advisors:

Dr. Oscar Pastor Lopez, *Universitat Politècnica de València*

Dr. Francisco Valverde Giromé, *Universitat Politècnica de València*

PhD tribunal:

Dr. Vicente Pelechano, *Universitat Politècnica de València*

Dr. Juan Carlos Trujillo, *Universidad de Alicante*

Dr. Jan Mendling, *Vienna University of Economics and Business*

This PhD thesis was defended by Mª José Villanueva on 8th January 2016 to obtain a Doctor of Philosophy in Computer Science from Universitat Politècnica de València.

Agradecimientos

En este pequeño trocito de tesis me gustaría dar gracias a todas aquellas personas que han contribuido de algún modo en la realización de esta tesis.

En primer lugar, quiero dar la gracias a mis directores de tesis. A Oscar, por creer en mi desde el primer día (aunque al principio solo fuera por mi casa de la nieve en Rubielos), por darme libertad para llevar a cabo esta tesis y sobre todo por apoyarme hasta el final. A Paco, por guiarme durante todo el proceso, por su apoyo y ánimo constante, por sus acertadas críticas, por leer esta tesis varias veces y sobre todo por su ayuda incondicional durante estos 5 años.

También quiero dar las gracias a mi familia y amigos por cuidarme y apoyarme, ya sea viniendo a comer conmigo a la Universidad, insistiéndome cada cinco minutos en que acabara la tesis, preguntándome como estaba, haciéndome croquetas, arreglándome los desastres de mi casa, despidiendo positivismo en todo momento o estando ahí aunque solo nos hayamos podido ver una vez al mes o al año.

Por otro lado, quiero dar las gracias a todos los compañeros del Pros que también me han ayudado. A Marce, por ser mi compañera de viaje, por salir conmigo a todo tipo de ferias, y por escuchar todos mis rollos durante nuestras salidas cenicientiles acompañadas de vino de la casa. A mis compañeros del genoma, por su forma de ver la vida, que me ha ayudado a reflexionar y a salir de mi zona de confort, y en especial a Ana Levin, por su ayuda durante los primeros años. A Sergio y a Ignacio, por sus sabios consejos que me han ayudado a mejorar muchos aspectos de esta tesis. A Cristian y a Carlos, por aportar su granito de arena en la implementación de esta tesis. A Ana Cidad por ayudarme con la burrocracia a la que me he tenido que enfrentar. Y en general, a todas aquellas personas que han aguantado mis cambios de humor y aun así han tenido siempre buenas palabras hacia mí.

Tampoco puedo dejar de agradecer a todas aquellas personas de Imegen, GEM Biosoft, e INCLIVA por su colaboración en toda la parte genómica de esta tesis. En especial, me gustaría agradecer a Pablo por sus consejos y por su ayuda durante los meses en los que estuvimos colaborando sin los que la validación de esta tesis no hubiera sido posible.

Y por último, a ti, al más especial, porque el ánimo que me has dado cada día con tu forma de ser y con tu gran éxito "la tesis no se escribe sola" no tiene precio. Gracias, porque sin ti, ni mi vida ni esta tesis serían lo mismo.

Overview

Domain-specific languages (DSLs) are considered to be a powerful tool for enhancing the efficiency of software developers and bring software development closer to end-users from complex domains. However, the successful development of a DSL for a complex domain is a challenge from the technical point of view and because end-user acceptance is key.

Despite this fact, the relevant role of end-users during DSL development has traditionally been neglected. Normally, end-users participate at the beginning to communicate their preferences but they do not participate again until the DSL is completely implemented. As a consequence, if the language to develop reaches a complex domain, the chances that errors appear in the DSL are higher and solving them could involve large modifications that could have been avoided.

As a solution, in this PhD thesis, we propose an agile, model-driven method to involve end-users in DSL development. This thesis researches if the combination of best practices from the model-driven development (MDD) discipline and best practices from agile methods is a suitable approach to involve end-users in the DSL development process.

In order to validate the proposal, we have selected a highly complex domain such as the genetic analysis domain and we have collaborated with geneticists from three organizations. The proposed method has been used to involve these geneticists in the development of a DSL for the creation of genetic analysis pipelines. Simultaneously, we have carried out an empirical experiment to validate whether end-users and developers were satisfied with the proposal.

Resum

Els llenguatges específics de domini (DSLs) son una ferramenta molt potent per a millorar l'eficiència dels desenvolupadors de programari, així com per a apropar el desenvolupament de programari a usuaris sense coneixements informàtics. El problema es que desenvolupar un DSL es complex, no sols des del punt de vista tècnic, sinó especialment perquè l'acceptació de dit llenguatge per part dels usuaris finals es clau.

Malgrat aquest fet, els mètodes tradicionals de desenvolupament de DSLs no emfatitzen l'important rol dels usuaris finals durant el desenvolupament. Normalment, els usuaris participen a l'inici per a comunicar les seues preferències, però no tornen a participar fins que el DSL està completament desenvolupat. Si el llenguatge a desenvolupar aborda un domini complex, la possibilitat de que hi hagen errors en el DSL es major i solucionar-los podria implicar modificacions de gran calibre que podrien haver-se evitat.

Com a solució, en aquesta tesis proposem un mètode de desenvolupament de DSLs, àgil i dirigit per models que involucra als usuaris finals. Aquesta tesis investiga si la combinació de bones pràctiques del desenvolupament dirigit per models (MDD) i de bones pràctiques de mètodes àgils es adequada per a involucrar els usuaris finals en el desenvolupament de DSLs.

Per a validar la idoneïtat de la proposta, s'ha seleccionat un domini complex com el dels anàlisis genètics i s'ha col·laborat amb un conjunt de genetistes procedents de tres organitzacions. El mètode s'ha utilitzat per a involucrar a dits genetistes en el desenvolupament d'un DSL per a la creació de pipelines per al anàlisis genètic. Al mateix temps, s'ha dut a terme un experiment empíric per a validar si tant els usuaris finals com els desenvolupadors estan satisfets amb la proposta de la present tesis.

En resum, les contribucions principals d'aquesta tesis doctoral son el disseny i implementació d'un mètode innovador, àgil i dirigit per models per a involucrar als usuaris finals en el desenvolupament de DSLs, així com la validació de la proposta en un entorn industrial amb un desenvolupament real d'un DSL.

Resumen

Los lenguajes específicos de dominio (DSLs) son una herramienta muy potente para mejorar la eficiencia de los desarrolladores de software, así como para acercar el desarrollo software a usuarios sin conocimientos informáticos. Sin embargo, su principal problema es que desarrollar un DSL es complejo; no sólo desde el punto de vista técnico, sino especialmente porque la aceptación de dicho lenguaje por parte de los usuarios finales es clave.

A pesar de este hecho, los métodos tradicionales de desarrollo de DSLs no enfatizan el importante rol de los usuarios finales durante el desarrollo. Normalmente, los usuarios participan al inicio para comunicar sus preferencias, pero no vuelven a participar hasta que el DSL está completamente desarrollado. Si el lenguaje a desarrollar aborda un dominio complejo, la posibilidad de que existan errores en el DSL es mayor, y su solución podría conllevar a modificaciones de gran calibre que podrían haberse evitado.

Como solución, en esta tesis proponemos un método de desarrollo de DSLs, ágil, y dirigido por modelos que involucra a los usuarios finales. Esta tesis investiga si la combinación de buenas prácticas del desarrollo dirigido por modelos (MDD) y de buenas prácticas de métodos ágiles es adecuada para involucrar a los usuarios finales en el desarrollo de DSLs.

Para validar la idoneidad de la propuesta, se ha seleccionado un dominio complejo como el de los análisis genéticos y se ha colaborado con un conjunto de genetistas procedentes de tres organizaciones. El método propuesto se ha utilizado para involucrar a dichos genetistas en el desarrollo de un DSL para la creación de pipelines para el análisis genético. Conjuntamente, se ha llevado a cabo un experimento empírico para validar si los usuarios finales y los desarrolladores están satisfechos con la propuesta de la presente tesis.

En resumen, las contribuciones principales de esta tesis doctoral son el diseño e implementación de un método innovador, ágil y dirigido por modelos para involucrar a los usuarios finales en el desarrollo de DSLs, así como la validación de dicha propuesta en un entorno industrial en un desarrollo real de un DSL.

Contents

List of Figures

List of Tables

1. Introduction

In the last two decades, computer science has become a transversal field that serves multiples domains and multiples purposes. Besides the software products that have been created for domains such as banking, stock management, or gaming, a broad amount of software products are being developed nowadays for other domains such as genetics, aviation, seismology, or archaeology, among others.

Software tools were initially designed to support miscellaneous tasks, mostly related with work matters. When these tools improved, traditional manual procedures left the room for computer-aid procedures. Software tools became part of our daily life and, since they are better every day, eventually, people have started to use software tools for their routine and entertainment.

As the usage of software tools by different kind of end-users increased, their needs and preferences became an important factor to take into account by developers [1]. In fact, developers believed that the participation of end-users in the creation of software products would increase their suitability and success. This belief was the origin of the "end-user software engineering" [2]. From that moment, end-user needs were important during software development, so the possibility of involving them in the development process seemed an approach worth to explore.

The challenge was involving non-technical end-users within the software development process. Understanding different development activities required having a high technical knowledge related to software engineering. As a solution, the software engineering community proposes using domain-specific languages (DSLs) [3, 4] to overcome this knowledge gap and to bring software development closer to end-users who do not have such development knowledge.

Domain-specific languages are languages that abstract programming concepts and technological implementations by using concepts of a specific domain. Because of the encapsulation of implementation concerns, DSLs enhance the efficiency of software developers. Because of their abstraction and conceptualization, DSLs also facilitate the comprehension of software specifications for end-users.

Initially, DSLs were proposed to improve the efficiency of developers. Developers used DSLs to encapsulate well-established and repetitive procedures so they could be easily reused afterwards (by them or by other developers). Therefore, the most well-known DSLs (such as SQL or HTML) only addressed technical domains related with computer science. These languages achieved a higher abstraction level, but at the end of the day, their target users were still users with advanced technical knowledge.

However, since DSLs are a powerful tool to hide technological details such as programming concepts or implementation technologies, the possibility of using DSLs to involve end-users in the software development process arose interest in complex application domains such as genetics, aviation, or seismology [5]. However, in practice, developing a DSL for this kind of domains was not that easy.

In technical domains, DSL developers are experts of the domain or have enough technical knowledge related with it (besides being the target users of the DSL), which makes the development of the DSL difficult but attainable. In contrast, in complex application domains, DSL developers are not experts (nor future users of the DSL), which means that they must deal with very specialized knowledge that is usually beyond their technical knowledge and reasonable understanding. In addition, the technical gap between developers and end-users hinders the ability of developers to identify the right needs and preferences of these end-users.

An example of this technical gap can be found in the genetic analysis domain. This domain is the knowledge area that gathers the set of tools and procedures that are used by geneticists to analyze genetic samples (such as DNA or RNA) of living beings to obtain information about their external features (such as skin color or a disease condition).

After working with geneticists for several years [6, 7, 8], we have observed a huge gap between software geneticists and developers. For geneticists, software development concepts are beyond basic computer skills. For software developers, genetic concepts are beyond basic biological understanding. This gap, together with the dynamism of genetic concepts and analysis procedures, has made unfeasible for developers to understand all the specific details, to keep track of their constant changes, and to develop accurate software products accordingly. The consequence is that the software tools that have been developed over the last decade to perform genetic analysis have been unsuccessful to fulfill geneticists' needs [9].

In this scenario, geneticists saw no other option than participating in the software development and becoming the developers of their own tools. A common approach followed by geneticists was acquiring basic knowledge of programming and development technologies to create databases, scripts, and web applications. However, their lack of knowledge of software quality standards and best practices for software development (such as usability, interoperability, portability, reusability, maintainability, documentation, etc.) led to a huge proliferation of non-interoperable tools and heterogeneous databases.

Eventually, these software tools are very difficult to use for other geneticists, who also need high programming skills to configure them or to achieve interoperation among them. As a solution, a DSL could improve this situation and help geneticists to reuse and customize available software tools without worrying about technological and low-level details.

However, as we have already mentioned, developing a DSL for a complex application domain such as the genetic analysis domain is a challenge for software developers. From the technical point of view, developing a DSL is already complex and time consuming; in this scenario, the complexity of the genetic analysis domain adds a further barrier to the task. Developing this DSL requires understanding genetic concepts and implementing it requires knowing existing genetic analysis tools.

In conclusion, the set of problems that we have observed in the genetic analysis domain illustrate the important role that geneticists must play during the development of a DSL. Nonetheless, we have realized that these problems are not specific of this domain and are also applicable to other complex application domains in which developers have not enough expertise (for instance aviation or seismology). This means that, in order to ensure that a DSL for a complex application domain represents all the domain concepts precisely and fulfills the end-users' needs, the participation of end-users during the development process itself is essential.

1.1 Motivation

Because of the aforementioned reasons (technical complexity and domain specificity), developing a DSL for a complex application domain requires a development approach that: 1) provides guidance in the different development activities (Requirement 1); 2) ensures that the development time is feasible (Requirement 2); and 3) facilitates the gathering of domain experts' knowledge and preferences (Requirement 3).

With the aim to guide developers (Requirement 1), the state of the art is full of different methodological guidelines. We highlight the work of Mernik et al. [10, 11] (originally published in 2005 and extended in 2011), which presents the different stages of DSL development and illustrates a set of patterns that facilitate different development decisions. Another example is the systematic development approach of Strembeck and Zdun [12], which defines the set of activities to accomplish for DSL development and decision charts that illustrate different development decisions. Also, the best practices and lessons learned to develop DSLs proposed by Van Deursen et al. [3], Spinellis [13], and Czarnecki et al. [14].

Complementing these works, with the aim to improve the development efficiency (Requirement 2), some works propose applying Model-Driven Development principles (MDD) [15, 16], which encourage investing all the development's effort on representing the system abstractly by means of conceptual models and generating the system code (automatically or systematically) through model-to-code transformations. Several authors claim that MDD improves the efficiency when developing a software system [17, 18]. According to [19, 20, 21]

the conceptual models that describe the system are eventually easier to specify, understand, and maintain that the underlying programming code; the complexity is addressed and solved in the problem domain without technological issues; and solutions are not tied to specific implementation technologies. In the context of DSL development, language requirements are formally described using models and DSL artifacts are generated from them [22].

The issue of these aforementioned approaches is that the essential role of the end-user (Requirement 3) is neglected. End-users only participate in the initial requirements gathering step and the DSL is implemented without their further participation. As a consequence, the probability to miss domain misunderstandings and discover them after a first version of the DSL is delivered is higher. The development time of the DSL will substantially increase if the language editor or the execution environment must be re-implemented to support changes.

For these reasons, this PhD aims to provide a solution that fulfills the three requirements; a solution that guides developers through the different development activities (Requirement 1), ensures the efficiency of the DSL development process (Requirement 2), and involves end-users as much as needed in the DSL development process (Requirement 3).

In order to provide the suitable **guidance**, we adopt different guidelines from the state of the art (such as the Mernik et al. stages or the decision charts from Strembeck et al.) to propose a method that details the different stages of DSL development, the different steps that must be accomplish in each stage, and the artefacts to be created in each step. In order to ensure the **efficiency** of the development process, we adopt Model-Driven Development (MDD) to benefit from their advantages in the context of DSL development. We combine the definition of DSLs with the MDD paradigm [22]: using conceptual models to formally describe concepts of the language domain and applying model-based transformations to generate the DSL artefacts from them. Finally, in order **to involve end-users** in the development process, we propose a set of mechanisms that facilitate end-user participation in the definition of those models. We name "mechanism" to the set of activities and artefacts that are proposed to gather end-users' feedback about a certain aspect of a DSL. Thanks to these mechanisms and the MDD approach, the knowledge that is gathered from end-users is represented and propagated throughout the complete set of DSL artefacts.

As we already mentioned, involving end-users in a DSL development process is not trivial; but involving them in the creation of conceptual models when following a model-driven approach is even a further problem because end-users don't usually have the expertise necessary to participate in modeling tasks [2]. MDD approaches provide formalisms to design the conceptual models and to generate the software products from them, but they lack clear guidelines to teach end-users how to contribute to model these formalisms.

In contrast, agile methods [23] advocate the close collaboration of end-users and developers, focusing on requirements, testing, and project management. However, they lack guidelines to carry out different conceptual modeling activities such as domain modeling, business modeling, or behavior modeling [24]. For this reason, we believe that MDD approaches and agile methodologies can complement each other in favor of end-user involvement in the context of DSL development. Therefore, the mechanisms proposed to involve end-users are based on best practices from agile methods that focus on increasing end-user participation.

In summary, **this PhD thesis proposes an agile model-driven method to involve end-users in DSL development**. We explain the complete method stages and steps, the different conceptual artefacts to be created in each step, and the set of involving mechanisms that facilitate the participation of end-users in the creation of those artefacts.

In order to validate the proposal, we focus on evaluating the most innovative contribution of this method, which are the set of mechanisms for involving end-users. The validation of the method artefacts and the benefits of MDD (such as efficiency) in the context of DSL development is outside the scope of this work.

In order to validate the mechanisms for involving end-users, we carried out a controlled experiment with geneticists from the Research Institute INCLIVA[1] to develop a DSL to support genetic analyses. This experiment was an expert opinion research [25] in which geneticists used the mechanisms of the method and provided their opinion about them.

[1] INCLIVA. Instituto de Investigación Médica del Clínico de Valencia

1.2 Research questions and objectives

The main goal of this PhD thesis is to provide a methodological approach to involve end-users in DSL development. In order to accomplish this goal, we must answer to the following research questions:

RQ1. Is it essential to involve end-users in the development of a DSL for a complex application domain?

RQ2. Which are the available approaches to involve end-users in DSL development?

RQ3. How can we provide a methodological approach to involve end-users in DSL development?

RQ4. How can we validate that the solution proposed is a suitable solution to involve end-users in DSL development?

In the search of the RQ1, we choose the genetics domain as the research context. In order to answer this main research question, we must answer the following research questions:

RQ1a. Does the genetic analysis domain require the development of a DSL?

RQ1b. Is it essential to involve geneticists in the development of a DSL for supporting genetic analysis?

In order to answer these questions, the main objectives of this PhD thesis are:

Objective 1 (RQ1): In order to answer RQ1, we need to illustrate whether involving end-users in the development of a DSL for a complex application domain is necessary. We will choose the genetic analysis domain to illustrate this need. First, in order to answer RQ1a, we will analyze the current state of the domain and we will justify the need to provide a DSL. Then, in order to answer RQ1b, we will discuss why the development of a DSL for supporting genetic analysis requires the participation of geneticists during the development process.

Objective 2 (RQ2): In order to answer RQ2, we will search for current approaches for DSL development. We will analyze the approaches that provide guidance for developers and improve the efficiency of the DSL development process. We will specially focus on the approaches that involve end-users. In order to analyze them, we will propose several analysis criteria and we will apply them to characterize each approach. Finally, we will discuss the contributions of these

works and their open problems in relation with the problem addressed by this PhD.

Objective 3 (RQ3): In order to answer RQ3, we will propose an agile model-driven method to involve end-users in DSL development. First, we will study methodological guidelines for DSL development from the state of the art (specifically, model-driven oriented) in order to configure the different steps and the model-driven artefacts to be created in each stage of the DSL development. Then, in order to involve end-users in the creation of some of these artefacts, we will propose a set of mechanisms based on agile practices.

Objective 4 (RQ4). In order to answer RQ4, we will conduct a controlled experiment with geneticists from an industrial environment to validate the suitability of the mechanisms for capturing domain knowledge and end-users' needs. Another goal of this collaboration is to obtain a preliminary version of a DSL for supporting genetic analysis.

1.3 Methodology

1.3.1 Methodological framework

Design science [26, 27, 28] is a methodology that fosters the creation of artefacts that solve problems of the environment (whose motivation is driven by a business problem) and contribute to the current knowledge base. Specifically, the research methodology selected to guide this thesis is the proposal of Wieringa [25] of Design Science methodology as set of nested regulative cycles.

Wieringa proposes to solve engineering and research problems that are found by researchers by decomposing them into engineering and research sub-problems: a main problem that is seen as a set of nested problems. An engineering problem is the "difference between the way the world is experienced by stakeholders and the way they would like it to be" and a research problem is the "difference between the current knowledge of stakeholders about the world and what they would like to know".

The approach to solve both types of problems is to follow a regulative cycle made of five tasks: problem investigation, design, validation, implementation and evaluation. However, as it is shown in Figure 1.1, depending on the type of problem, each problem is addressed with a slightly different regulative cycle. An

engineering cycle (EC) has the tasks: problem investigation, solution specification, specification validation, specification implementation, and implementation evaluation. While a research cycle (RC) has the tasks: research problem investigation, research design, design validation, research execution, and analysis of results.

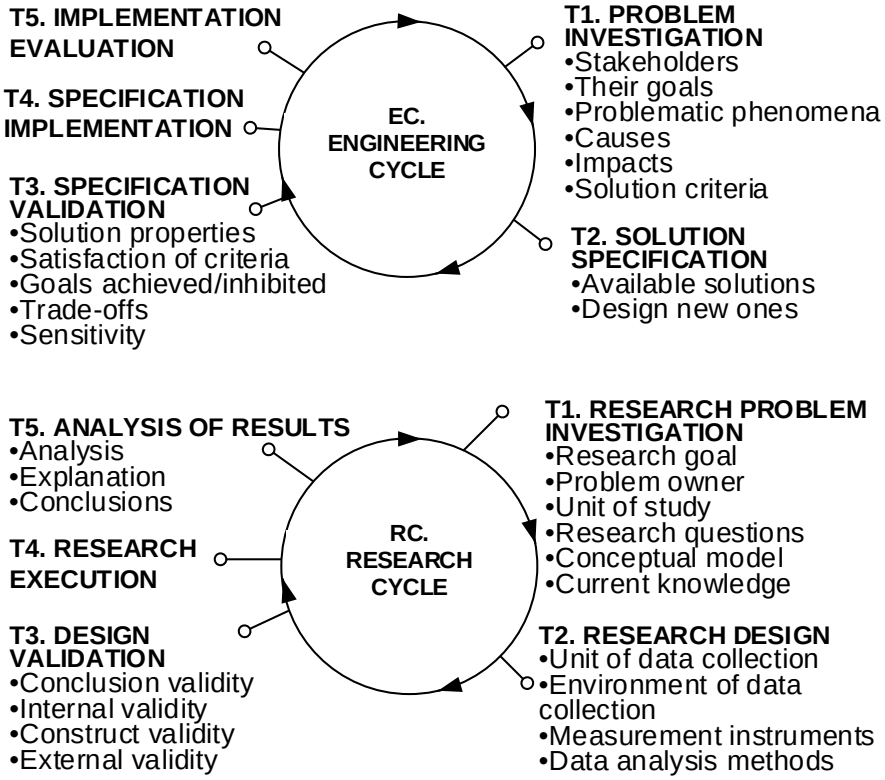


Figure 1.1 Design science as a regulative cycle

When solving a problem, the methodology starts characterizing the general problem as a research or as an engineering problem so that the corresponding cycle can be applied. On the one hand, if the problem found is an engineering problem, the engineering cycle is applied (EC). If this is the case, the first step is to analyze the business problem in detail (*Problem Investigation*): 1) the stakeholders that have the problem; 2) the specific goals they want to accomplish; 3) the problem they are facing and their causes; 4) the impact of the problem in their context; and finally, 5) the criteria for stakeholders to consider the problem as solved.

After the problem has been characterized, it is necessary to research the domain in order to justify that none existing solution solves the problem. When no satisfactory solution has been found, there is room to propose a new one. As a consequence, a new solution is designed with the aim to solve the problem (*Solution Specification*).

Once this design is completed, it is necessary to validate the solution (*Specification Validation*) before their realization. For that matter, the solution properties are assessed according to the criteria defined in the problem investigation, characterizing the context of application and the coverage of the solution. If the solution has the desired effect for stakeholders in their context, the solution can be finally implemented (*Specification Implementation*). In the next iteration of the cycle, the implementation of the solution is evaluated.

On the other hand, if the problem found is a research problem, the research cycle is applied (RC). If this is the case, the first step is to analyze the knowledge problem in detail (*Problem Investigation*): 1) the specific research goal that is being pursued; 2) the problem owner; 3) the unit of study; 4) the set of research questions that want to be answered; 5) the conceptual model; and 6) the current knowledge.

After the problem has been characterized, it is necessary to design how the research is going to be conducted by establishing the data collection unit and environment, the instruments, and the data analysis methods. Next, it is necessary to assess the threats to validity. The design must ensure conclusion, internal, construct, and external validity. After ensuring the validity of the research design, the research can be conducted. Finally, the results obtained from the research are analyzed and conclusions are extracted.

During the execution of a regulative cycle (both EC and RC), it is likely that new (sub) problems arise. In order to solve them, the methodology proposes to open new regulative cycles and to address all the tasks before continuing with the previous cycle. Eventually, all the cycles will be completed and the original main problem will be solved.

1.3.2 Methodology applied to this thesis

The motivation of this PhD is to solve the problem: "*Provide a methodological approach to involve end-users in domain-specific languages development (for complex application domains)*". According to the methodology, we characterize this

problem as an engineering problem and we address the tasks of the regulative engineering cycle (EC).

Following this methodology (Figure 1.2), we start investigating this problem (T1.1) by defining the motivation to involve end-users in DSL development. To do this, we analyze the genetic analysis domain and the suitability of designing a DSL for this domain (T1.1.1). Then, we discuss why it is necessary to involve geneticists in the development of that DSL (T1.1.2). After justifying the need to involve end-users in DSL development, we analyze if any proposal of the state of the art fulfills this need (T2.1). The goal of this analysis is to understand the current open problems and to know whether it is necessary to propose a new solution. Since this is the case, we propose a DSL development method to involve end-users (T2.2). Then, this method is validated in a real environment with end-users (T3.1). To do this, we apply the method together with geneticists to develop a DSL for genetic analysis (T3.1.1) and we validate with an empirical experiment whether the proposed mechanisms of the method are a good solution for involving end-users in DSL development (T3.1.2). The search of this knowledge leads us to open a new research regulative cycle (RC1).

The goal of the RC1 is to "*Validate the mechanisms of the DSL development method*". In order to conduct this validation, we first define the research goals and the research questions of the experiment (T4.1). In order to address them, we design a controlled experiment to validate the method in a real environment with geneticists. Specifically, we design an experiment of the type expert opinion research (T5.1) and we analyze the validity of this design (T6.1) to be aware of the threats to validity. After this analysis, we execute the controlled experiment with geneticists (T7.1) and extract conclusions from the gathered data and lessons learned from this experience (T7.2).

The final step (T8.1) is to transfer the method to industry, which is outside the scope of this PhD thesis.

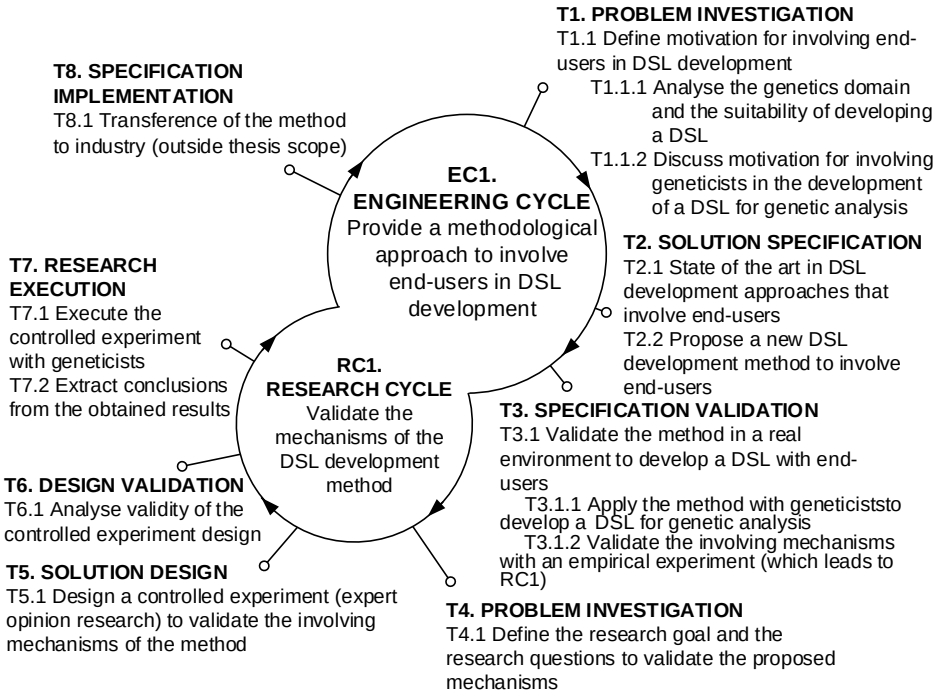


Figure 1.2 Regulative cycles of this PhD

1.4 Thesis outline

In order to address the objectives of this thesis, we follow the task order presented in the regulative cycles of the methodology (from task T1 to task T7). Accordingly, the structure of the thesis is as follows:

- In chapter 2, *Problem Investigation*, we address the task T1.1 to illustrate the motivation of involving end-users in DSL development by using the genetic analysis domain. We start the chapter describing the business goals of the geneticists from two industrial environments and we characterize the problems that remain unsolved that justify the need of a DSL. Then, we discuss why it is necessary to involve geneticists in the development of this DSL for supporting genetic analysis. In this chapter, we address the objective 1.
- In chapter 3, *State of the Art*, we address the task T2.1 to understand the current state of DSL development for complex application domains. We start analyzing works that provide guidelines for DSL development and

address how to improve the development efficiency and then, we focus on DSL development proposals for involving end-users. In order to analyze the suitability of these works, we propose several criteria and we analyze each work accordingly. Finally, we compare all the proposals together in order to justify that the main problem described by this PhD thesis has not been completely solved yet. In this chapter, we address the objective 2.

- In chapter 4, *Overview of the Method and Illustrative Example*; chapter 5, *Knowing the domain of the DSL*; chapter 6, *Realizing the DSL;* and chapter 7, *Releasing the DSL to end-users*; we address the task T2.2 to propose a solution to develop DSLs for complex application domains. We start explaining the methodological foundations and the existing approaches of the state of the art in which the method proposed is based and inspired. We provide an overview of the method and then, we go on explaining in detail the stages, the steps of each stage, and the mechanisms for involving end-users. Additionally, at the end of each chapter, we illustrate the DSL that we created by applying the method together with geneticists (task T3.1.1). In these chapters, we address the objectives 3 and 4.
- In chapter 8, *Validation*, we describe the complete experiment that was carried out to validate the proposal (task T3.1). We start establishing the research goals and questions (T4.1), we design the experiment (T5.1), we validate the experiment design (T6.1), we describe the experiment execution (T7.1), and we extract conclusions from the data gathered in the experiment (T7.2). In this chapter, we address the objective 4.
- Finally, in chapter 9, *Conclusions*, we overview the results of the thesis and the industrial collaborations, we list the research publications performed, and we discuss about the fulfilment of the objectives, the lessons learned and the future work.

2. Problem Investigation

Developing DSLs for complex domains is a challenge for software developers, since it requires dealing with domain complexity and overcoming the gap between developers and end-users. During DSL development, developers need to represent domain concepts and end-users' preferences into the language. Otherwise, the end-users will not see the value of the DSL and will not be willing to use it.

The challenge lies in the necessity to acquire all this domain knowledge so it can be precisely represented into the future DSL. An example of this challenge can be found in the genetic analysis domain; a complex domain that is characterized by its specialization and constant evolution. In this domain, developing a DSL will require developers and end-users to understand each other to achieve that the DSL precisely represents domain concepts such as DNA, Gene, or Genome as well as the geneticists' needs.

In order to illustrate the reality of this challenge, we collaborated with two organizations: Imegen[2], a Small and Medium Enterprise (SME) whose expertise is the diagnosis of genetic diseases, and INCLIVA, a Research Institute whose expertise is the research of genetic diseases.

[2] Imegen. Instituto de Medicina Genómica. www.imegen.es

The collaboration with geneticists from both environments revealed some problems in the domain that could be solved with the development of a domain-specific language (DSL). However, this analysis has also shown the difficulties of developing this DSL and the need of involving geneticists in the development process as a way to overcome them.

In this chapter, we start with a brief overview about the genetic analysis domain by explaining the basic concepts that are needed for next sections and chapters. Next, we analyze the situation of this domain by exploring Imegen and INCLIVA business processes and their problems with software tools for genetic analysis. After this analysis, we justify the suitability of developing a DSL for the genetic analysis domain as a solution that could tackle with some of those problems. Then, we provide the lessons learned about this DSL. Finally, we discuss about the complexity of developing DSLs for complex application domains, including the genetic analysis domain, and the need of involving end-users in the development process.

2.1 Introduction to the genetic analysis domain

The analysis of DNA has improved the existing knowledge of human traits and has leveraged the eager of human beings to understand the reasons for our differences and similarities. DNA is a molecule that encodes the information to create a living being. By means of four chemical bases (adenine A, guanine G, cytosine C, and thiamine T), DNA joins these bases in a sequence and uses these bases as instructions (similarly to the binary code) to create the proteins that are responsible of different life processes. For this reason, as if it was a software program, if any of these bases change in an individual, a "bug" could appear and cause an anomalous function.

When the Human Genome Project was funded in 1990 [29], a genomic revolution was yet to come. The goal of this project was to obtain the genome of an individual (the complete DNA sequence of their cells), which will serve as a model for future genomic analyses.

Since then, both sequencing techniques and technologies have been evolving constantly, being especially revolutionary in 2005 with the appearance of Next Generation Sequencing (NGS) technologies. In 2000, the cost per raw genome was almost 10 millions of dollars but nowadays, the cost has decreased to less than

1.000 dollars (Figure 2.1) and predictions of improving these figures are still optimistic.

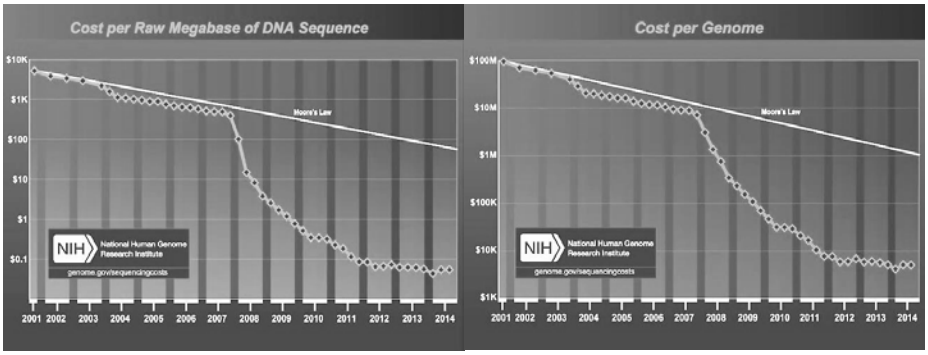


Figure 2.1 Sequencing costs from the National Human Genome Research Institute[3]

This scenario opened the door to the personalized medicine, where individuals could be analyzed genetically and treated according to their genetic features. Additionally, analyses of this kind were the starting point of a new methodology to predict and prevent certain genetic diseases.

Nowadays, geneticists are able to sequence a DNA sample (such as blood) and obtain the complete genome of an individual. From this genome, the goal is to establish the corresponding relationships of DNA bases (or sets of those bases) with human traits, but mostly, with damaging traits associated with diseases. Their approach (Figure 2.2) is comparing the sample DNA sequence with a DNA reference sequence to locate the differences, also known as variations (step 1). Once the individual's variations have been obtained, geneticists analyze the biological features of these variations in order to try to understand their damage. Then, they search for additional information over different databases, where other geneticists had previously shared their knowledge about the same variations (step 2). Finally, geneticists generate a report with all the information gathered and their conclusions (step 3).

[3] Figure obtained from the National Human Genome Research Institute. http://www.genome.gov/

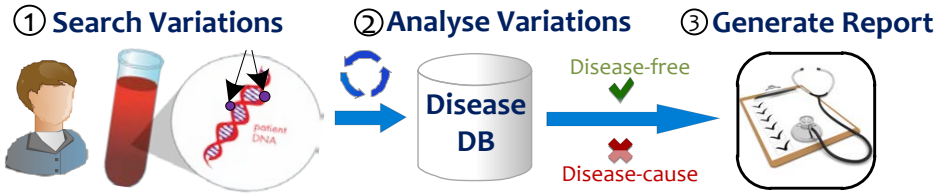


Figure 2.2 Overview of the genetic diagnosis process

Initially, the high price and long time for sequencing the complete genome hindered the work of geneticists, who had to limit their work to analyze one or several genes at a time. A gene is a subset of the sequence of the genome that is responsible to codify for a specific function of the human body. A human gene is usually made by 10 or 15 Kbases in comparison with the 3 or 4 Gbases of the human genome.

When geneticists could only sequence one gene at a time (due to the high sequencing costs and time), they were able to perform their work applying some manual procedures. However, with the appearance of NGS technologies, a higher set of genes could be sequenced at a time and several diseases could be simultaneously analyzed. As a consequence, manual procedures have become obsolete, forcing geneticists to embrace new procedures that are only feasible if they are supported by software tools.

2.2 Illustrative scenario: A DSL for the genetic analysis domain

In this section, we describe the Imegen and INCLIVA business processes using the Business Process Management Notation (BPMN[4]) and we analyze the existing problems with the software tools they use. As a solution to these problems, we discuss the potential benefits of developing a DSL for this domain.

2.2.1 The Imegen scenario

Imegen is a SME (Small and Medium Enterprise) whose expertise is supporting the genetic diagnosis of patients using sequencing technologies. In short, Imegen geneticists apply their genetic knowledge to provide evidences that confirm or discard the genetic nature of a disease in a patient.

[4] Busines Process Management Notation. Object Management Group. http://www.bpmn.org/

The diagnosis process of Imegen (Figure 2.3) starts when a patient (after detecting a set of symptoms) goes to the physician's practice (T1) and explains their symptoms (T2). From the set of symptoms of the patient, the physician performs a preliminary diagnosis (T3) and decides the set of tests that will be necessary to confirm or discard the disease (T4). Currently, physicians always run non-genetic tests (T5) but if the disease may have a genetic nature, a genetic analysis is required. When this is the case, the physician extracts a DNA sample from the patient (T6) and requests diagnosis support to the genetic laboratory (Imegen) (T7). When the genetic laboratory receives the request together with the DNA sample, they carry out a genetic diagnosis (T8) and send a report of the genetic diagnosis back to the physician. From this report and the results of the other non-genetic tests, the physician is able to confirm or discard the diagnosis (T9). Then, the physician gives the patient the definitive genetic diagnosis report.

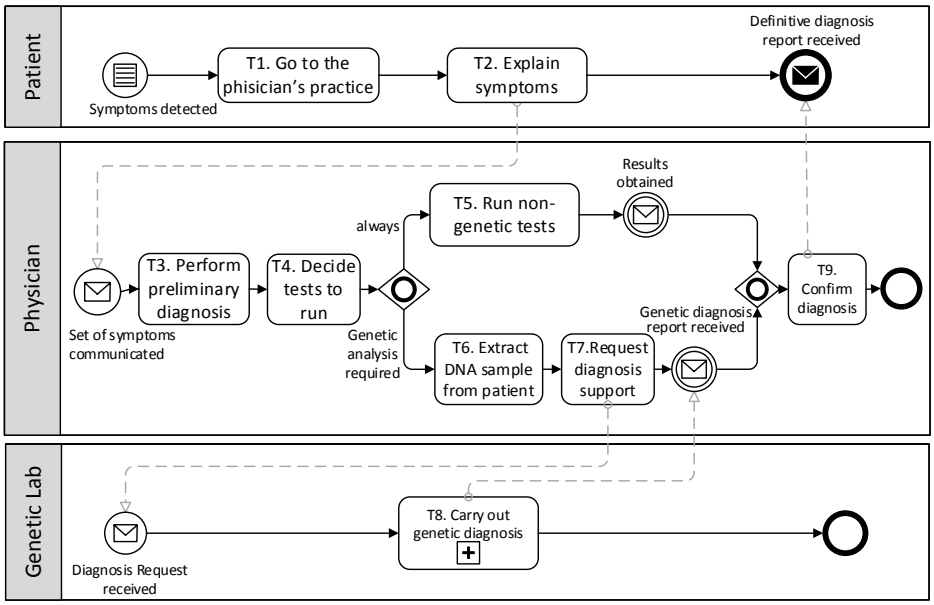


Figure 2.3 The Imegen business process

Figure 2.4 describes the business process of the genetic laboratory Imegen when a request to carry out a diagnosis and the DNA sample to be analyzed are received. First, the geneticists read the diagnosis report (T1) in order to understand the goal of the analysis and the way to proceed. Depending on the disease they want to diagnose, the geneticists must decide between sequencing a

subset of genes[5] or sequencing the complete set of genes of the genome that are relevant for diagnosis (a.k.a. the clinical genome).

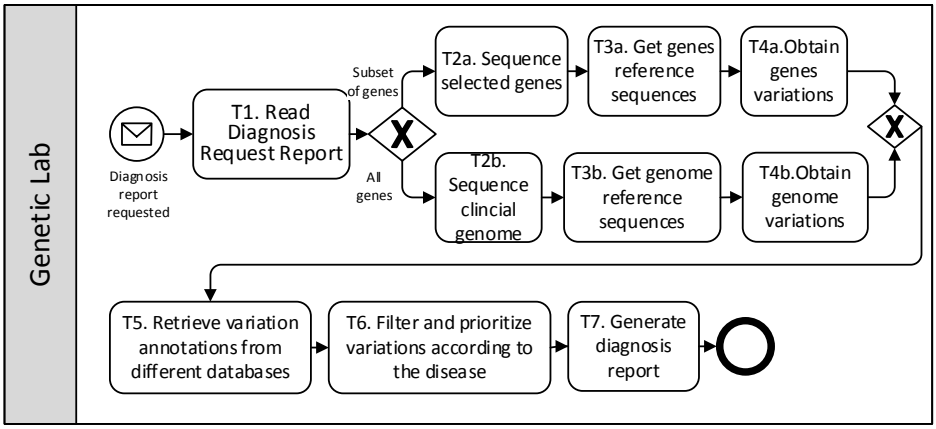


Figure 2.4 The Imegen genetic diagnosis process in detail

If the geneticists decide to focus only on a subset of genes (path a), they sequence only those genes (T2a), get the genetic reference sequences (T3a), and compare those sequences to obtain all the variations (T4a). On the contrary, if the geneticists decide to look into the whole genome (path b), they sequence the clinical genome (T2b), get the genomic reference sequences (T3b), and compare those sequences to obtain all the genome variations (T4b).

Once all the patient's variations of the DNA locations of interest are obtained, the geneticists annotate each variation with additional genetic data, which is retrieved from different genetic databases (T5). In the genetic jargon, annotate means to retrieve and attach metadata about an entity, in this case, a variation. Examples of annotations are the gene of the variation, its position in the sequence, or their damage effect. Once all the variations are annotated, the geneticists filter and prioritize them according to different criteria (T6). This way, they are able to see first the variations they are interested in and observe the annotated

[5] In practice, in order to reduce the sequencing price, geneticists do not sequence complete genes but only the parts of the genome that codify for proteins (a.k.a. exons). For simplicity, we have not included this detail in the explanation or the diagram.

information regarding the disease. With all this information, they generate the final diagnosis report (T7).

2.2.2 The INCLIVA scenario

Besides the collaboration with Imegen, we also started a collaboration with the researcher geneticists from the Genetic Diagnosis Unit (UGDG) of the INCLIVA Research Institute. The mission of these geneticists is researching different genetic and genomic diseases. During our collaboration, they were conducting a research project focused on the research of the Diabetes Mellitus Type 2 disease.

Specifically, the process for researching Diabetes Mellitus Type 2 starts (Figure 2.5) when the geneticists from the Analysis and Interpretation Unit choose the case (T1a) and control (T1b) individuals to be used in the research project. After these individuals' DNA samples are collected, they are treated as two different subsets: all cases together and all controls together. For each subset, this unit requests their sequencing (T2) to the Sequencing Unit. When this request is received, this unit sequences the genome (T3) and sends the sequencing files back to the Analysis and Interpretation Unit. When the sequencing files are received, this unit analyzes them (T4). After the analyses of both samples subset have finished, the results are compared to extract conclusions (T5).

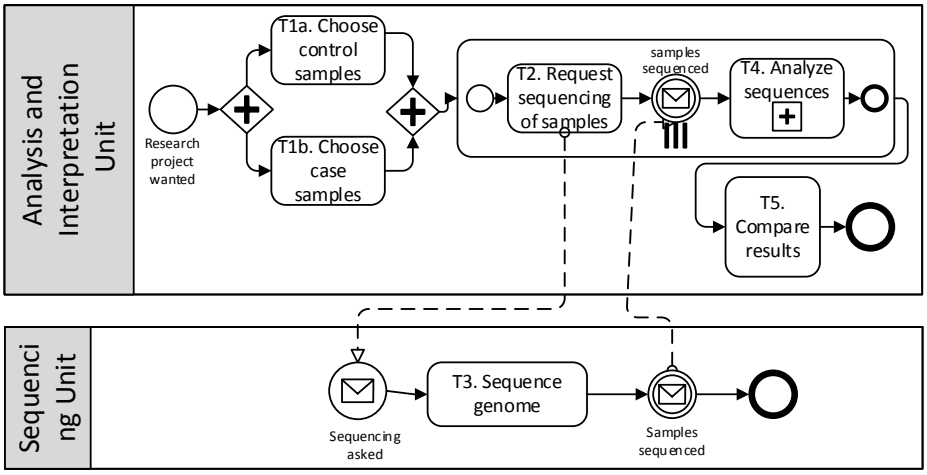


Figure 2.5 The INCLIVA business process

In order to carry out the analysis of each subset of samples, the researcher geneticists (Figure 2.6) get the genome reference sequences (T1). After

performing the comparison of the samples against those references, the list of all genomic variations from all the samples (T2) is obtained. Then, these variations are annotated with additional and relevant metadata (T3). Once all the variations from all samples are annotated, the researchers filter the variations according to the criteria they want to research (T5). In order to know these criteria, it may occur that a previous research is needed to find the suitable filtering criteria (T4). After filtering, the researchers prioritize the resulting variations to obtain a list of the variations ordered by relevance (T7). As it happened with the filtering criteria, it may occur that a previous research is needed to identify the suitable criteria (T6). Once both operations are applied to the variations, the researchers generate the final report (T8).

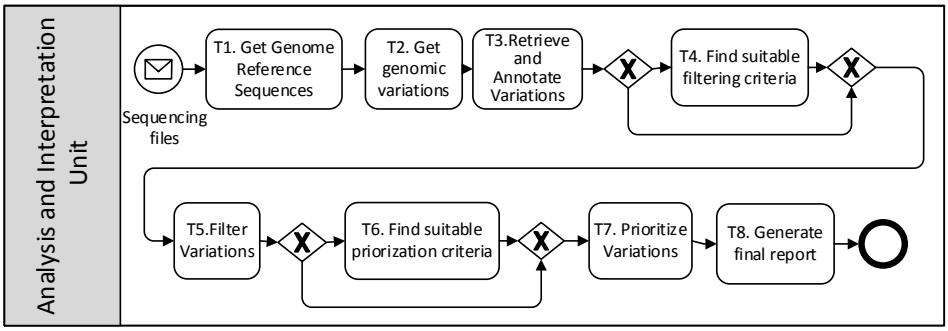


Figure 2.6 The INCLIVA sequence analysis process in detail

In general, there are few differences among the genomic process from Imegen and INCLIVA. Both processes have the same goal: "finding and characterizing variations", although the ultimate goal of each organization is slightly different. Imegen aims for diagnosis support and their procedure is based on a previous knowledge and is fixed. INCLIVA aims for genetic and genomic disease research and their procedure has some dynamic and not predefined activities that need to be configured. All in all, both procedures are part of the same kind (or family) of analyses.

2.2.3 Current issues and challenges

Imegen and INCLIVA are two examples of industrial environments whose original manual procedures to support their business processes (Figure 2.3 and Figure 2.5) were no longer feasible to analyze large amounts of data. Those

manual procedures had to leave the room for new ones supported by software tools.

The geneticists from those organizations now use several software tools to accomplish their genetic analyses: 1) gene-related tools, such as Sequencher [30], SeqScape [31], Codon Code Aligner [32], Mutation Surveyor [33], Polyphred [34], or inSNP [35]; and 2) genome-related tools, such as VCF Tools [36], BIOMART [37], Annovar [38], SNPEff [39], VEP [40], GATK [41], or SAM Tools [42].

The complete analysis of these tools is available in [43]. In this technical report, we gathered information about each tool such as authors, current version, or installation information, etc., and we analyzed the functionality provided by each of them in regards to the business process of Imegen and INCLIVA. As a conclusion of this analysis, we found several unsolved issues despite the full amount of software tools available. This analysis complements the contributions from other authors like [44, 9, 41, 45, 46].

The most common problems found in the aforementioned tools are the following:

1. **Several tools to support the complete analysis**: Geneticists do not easily find a tool that gathers all (or the majority of) the functionality that is required to execute a complete genetic analysis. There are many useful software tools for their analyses, but each of them focuses on accomplishing one or few tasks. For instance, addressing the alignment of sequences, annotating variations, or visualizing genomes.
2. **Difficulties for customization**: Geneticists need to customize the existing software tools to fulfill their specific needs. However, customization usually requires a deep knowledge of the technological implementation of the tool. For instance, when a new annotation must be included in the analysis or when geneticists need to change the default parameters of an alignment algorithm to fit the specific features of their samples, they need to modify some components of the software tools. Depending on the dimension of the customization, geneticists must acquire technical knowledge that can go from understanding the insights of the tool to change the source code written with a general programing language such as Python or Java.

3. **Difficult integration among tools**: Since a tool-that-fits-all is not a feasible approach, geneticists need to integrate different tools to accomplish their specific analyses. With this aim, geneticists need to create data processing pipelines composed by several tools. However, geneticists hit a wall when each tool uses different technologies and formats to represent data. As a solution, new standards for file formats have appeared so that tools can communicate to each other. Different genetic tools have adopted them but sometimes, geneticists use their extension mechanisms, creating files with formats outside the standard. Eventually, other tools cannot interpret such extended formats, having the same initial integration problems.
4. **Usability issues.** Current software tools have several usability issues such as: 1) some functionality does still not work properly; 2) there are not source code comments; or 3) there is not friendly documentation for end-users. In order to use them, geneticists must read technical documentation, which increases the learning curve of the tools. Another issue is that the majority of tools run under the Linux operative system. This operative system has been traditionally used at IT environments, and although their popularity is increasing among geneticists, its optimal usage requires skills on the Unix command line and scripting.

Besides the detected issues, another conclusion of this analysis is that these problems are a consequence of the huge existing gap between developers and geneticists. On the one hand, developers could not create the suitable tools for geneticists because they did not achieve a full comprehension of their needs. On the other hand, geneticists got tired of software tools that did not fulfill their needs and saw no other option than becoming the developers of their own software to accomplish their analyses; they acquired basic programming skills and programmed these tools.

Geneticists with programming skills, also named bioinformaticians, have been developing databases, analytics software, and repositories for sharing genomic data. Their common development approach has been acquiring some technical knowledge, for instance a programming language, and implementing a set of structured scripts to run their analyses over flat text files. However, their lack of knowledge of software quality standards and best practices for software development (such as usability, interoperability, portability, reusability,

maintainability, documentation, etc.) led to a huge proliferation of tools and heterogeneous databases; too frequently lacking the required technical quality.

2.2.3.1 A preliminary solution: Pipeline development environments

As a consequence of the aforementioned problems, geneticists who want to reuse existing solutions to perform their genetic analysis must face a technical challenge. Whether they like it or not, they must learn technological details and programming skills to use the existing tools, to achieve interoperation among them, or to customize them to fit their needs.

As a solution of this problem, pipeline development environments have been proposed with the aim to facilitate the reuse and customization of genetic software tools [47]. These environments support the creation of bioinformatics pipelines by integrating under a common interface existing components, such as libraries or software tools, and by providing friendly means to aid in the composition of the pipeline, such as predefined programming structures or a graphical notation.

The problem is that although geneticists know these environments, their use is still modest. Examples of these environments are BioPython [48], BioPerl [49], BioJava [50], Taverna [51], Galaxy [52], and eBioFlow [53].

The complete analysis of these environments is available in [43]. In this technical report, we gathered information about each environment such as authors, current version, or installation information, etc. We also analyzed the advantages and drawbacks while creating the bioinformatics pipelines that supported the processes of Imegen and INCLIVA (Figure 2.3 and Figure 2.5). As a conclusion of this analysis, we found that the usage of these environments is friendlier than programming with a general-purpose language, but they still have unresolved issues. This analysis complements the analyses from other authors like [46, 54, 55, 56].

As advantages, these environments achieve to solve two main issues:

1. **Several tools to support the complete analysis**: These environments praise for the creation of pipelines that integrate existing software tools under the same context. After composing the pipeline, geneticists can use it as a unique piece of software that receives an input, executes the desired analysis, and provides the corresponding results.

2. **Usability:** These environments integrate different analytics tools under a friendlier environment than a command line terminal. Thanks to this functionality, geneticists can easily choose the software tools they want to use and they do not have to worry about their installation. This functionality avoids geneticists the need to learn about Unix management and the usage of the command line.

As drawbacks, geneticists must still deal with the following unsolved issues:

1. **Difficulties for customization:** These solutions have eased the usage of the integrated tools by providing a higher abstraction level, for example, by using friendly interfaces. Thanks to this abstraction, geneticists can configure the parameters of the tools and execute it without the need of knowing the underlying command. In some cases, the provision of input data is made with a form with droplists, radiobuttons, checkbuttons, etc., which improves the usability for geneticists. However, in other cases, the abstraction level is not enough and the interfaces provide generic input fields whose format is unknown and maps directly to the parameters of an underlying command line tool. Therefore, when geneticists need to customize the tools to fit their needs, they still must know the technical details of the tool. In some cases, in order to achieve this customization, geneticists also need further technical knowledge about the pipeline development environment.
2. **Difficult integration among tools:** As an aid to create pipelines, these environments provide parsers for unifying the most common data formats. Since not all parsers for the formats that are used by geneticists are supported, these environments also provide text utilities to filter and rearrange data. As a consequence, geneticists must learn how to use the provided text utilities, which may be tedious when they require working at a lower level using columns, rows, and fields, and complex when they require the description of regular expressions and programming structures.
3. **High technological coupling:** These environments provide a tool set of bioinformatics software tools from which geneticists can chose to compose their analysis. As a consequence, they must identify the mapping between the genetic task they want to accomplish and the technological artefact most suitable for the task. This means that the pipeline created is coupled with the set of selected software tools.

A further conclusion from this analysis is that besides these drawbacks, the level of abstraction and expressivity provided by these environments is not fully satisfactory for geneticists. For geneticists with experience in genetic analysis technologies, these environments are not expressive enough. They need to describe all the details of the pipeline with the specific tools and parameters they want to apply, but these environments usually hide all the configuration possibilities in favor of usability.

Similarly, geneticists with less technological experience do not want such expressivity, but they need higher abstractions. They need a friendly environment to analyze what they want and to stop worrying about tools, parameters, and technological errors. However, current environments still provide an abstraction level where geneticists need to deal with all these problems: tool selection, configuration, and integration.

2.2.4 A DSL as a solution

As some authors mentioned [57, 58], bioinformatic pipeline development environments need to focus on their real users and to provide the suitable conceptualizations of their domain that describe "what" geneticists want to do instead of "how" they are going to accomplish their analysis. A friendly environment for geneticists must allow them to describe pipelines through genetic analysis concepts, instead of choosing specific software tools, configuring each of them with their specific parameters, and managing the interoperation among them.

In this direction, some advances have been made to improve the abstraction level of existing software products. One example is the Biocatalogue [59] repository, which gathers analytic services for the bioinformatics domain that have been annotated with ontology terms and keywords. This work goes a step further towards the development of a conceptualization environment, but it is still missing a friendly environment that takes advantage of these domain annotations and provides a conceptual environment in which geneticists perform their analyses using those conceptual abstractions.

Another example is the workflow development environment eBioflow [53], which proposes the identification of roles and actors and their association with tasks and technological artefacts, respectively. The roles and actors are described at design time and the workflow is enacted at execution time. Although the

environment achieves a better abstraction of the genetic analysis by providing the role and actor abstractions, it still lacks a robust proposal to define roles related with the genetic domain. Moreover, its current implementation integrates existing datatypes from several third-party repositories without taking into account their curation.

The idea of an environment that provides common conceptualizations to create genetic analysis pipelines seems fitting into place when we observe the effort of the community to create and share pipelines. Since geneticists reuse existing pipelines, it is undeniable that geneticists with different goals share tasks to some extent, which means that there is an underlying commonality among their goals. Furthermore, although geneticists use pipelines created by others, they also need to customize them to fit their needs, which means that behind those commonalities, the domain also manifests some variabilities that need to be managed.

Due to all these reasons, we strongly believe that a **domain-specific language can provide a friendly and effective environment for geneticists with the right abstractions to create pipelines using genetic analysis concepts**. Using this DSL, geneticists will be able to choose the domain tasks they want to accomplish, to configure them according to the features of the genetic analysis, and to avoid the underlying technological issues.

This DSL aims to solve the current issues of existing software tools for genetic analysis:

- **Several tools to support the complete analysis:** Thanks to the DSL, geneticists will be able to specify the genetic analysis they want to accomplish and the DSL infrastructure will instantiate the corresponding technological artefacts of the implementation environment. Afterwards, geneticists will be able to use this instantiation as a single software tool that completely fulfills their needs.
- **Difficulties for customization**: Thanks to the DSL, geneticists will not have to worry about acquiring technical knowledge about existing software tools while creating pipelines. The customization will be done while designing a pipeline by using the constructs of the DSL. The underlying customization of existing software tools will be solved during the development of the DSL.

- **Difficult integration among tools**: Thanks to the DSL, geneticists will not have to worry about the incompatibility of tool formats. They will not be responsible for interconnecting existing tools, only for choosing the set of DSL constructs they want to use. The integration complexity will be solved during the development of the DSL.
- **Usability:** The DSL will be specially designed for geneticists. This means that in order to specify a genetic analysis, geneticists will only have to use language constructs that use domain concepts and fulfill their preferences. The ultimate goal is that geneticists find the DSL easy to use and useful.
- **High technological coupling**: Thanks to the DSL, geneticists will not have to worry about selecting the software tools to use because the DSL will automatically selected them. Tool selection will be performed according to the DSL constructs used by geneticists while specifying their genetic analyses. In order to ensure the correct instantiation, the domain experts will supervise the mappings between constructs and tools during the design of the DSL.

The proposal of domain-specific languages in the bioinformatics domain is not novel as we can see in the works: MOLGENIS [60], a DSL for the rapid prototyping of user interfaces for genetic repositories, Greg [61], a DSL to describe genetic regulatory mechanisms, and the work from [62], a DSL to design organisms' expression vectors. However, these DSLs were designed to improve the usability and reduce mistakes when modelling genetic structures. Geneticists create models using these DSLs and use these models as a base to formulate hypothesis and to conduct experiments outside the bounds of the DSL. In short, these DSLs are used to describe genetic concepts but not to describe how to conduct bioinformatic analytic tasks over genetic data.

Nevertheless, there is a DSL, named BIOBIKE [63], for the combination of tools, data, and knowledge to conduct biological analysis. This DSL provides a higher conceptualization level than current existing development environments. However, it is based on the Lisp language [64] and still contains some programming elements that are not related with the biological domain such as the definition of variables and functions and the use of arithmetic or string operations. Nowadays, this DSL is outdated because it was created in 2009 and it only supports the biological operations that were used before the evolution of the genetic analysis domain and bioinformatics tools of the last years.

This means that there is still a need to provide a DSL with a higher conceptualization level and updated to support the knowledge related to the appearance of NGS technologies, the additional analytics options offered by the new bioinformatics tools, and the access to genetic and genomic data repositories that have been created in the last five years.

2.3 Lessons learned

Thanks to the collaboration with the geneticists, we have understood the situation of the genetic analysis domain and analyzed the potential benefits of a DSL for this domain. In addition, we have realized that developing a DSL for this domain requires taking into account several important concerns.

First, the DSL should represent concepts of the genetic analysis domain and hide concepts that are outside their domain (like programming concepts). Since geneticists are the experts of this domain and developers are experts in identifying the right abstractions to create a DSL, both developers and geneticists need to collaborate to identify which of these concepts are relevant and should be represented in the language.

Second, the DSL language structure that is offered to geneticists must be specially designed for them. Geneticists are the target users of the DSL, so the language structure must be designed according to their preferences. Since developers are the experts in designing the language structure, they should offer geneticists with ideas for that structure. With these ideas and the developers' aid, geneticists should draft this design; otherwise, it could contain elements only suitable from the developers' perspective.

Finally, the DSL provides a friendly interface for geneticists to specify a pipeline using genetic concepts, whose specification will be instantiated as a pipeline that integrates different genetic analysis software tools. Currently, the genetic analyses to be provided are highly coupled to existing genetic analysis software tools. For this reason, the mapping between the concepts of the domain and the existing software tools must be made in collaboration with geneticists since they are the ones who know the tools that must be used and how they are configured to provide the suitable functionality.

2.4 Conclusion

The genetic analysis domain has invested many efforts to both create and share existing software tools and pipelines for genetic analysis. Geneticists have acquired software development knowledge and invested their efforts developing their own software tools. However, regardless all these efforts, geneticists complain that the genetic analysis domain still lacks a fully satisfactory solution.

In order to understand the domain situation, in this chapter we have reviewed the context of Imegen and INCLIVA geneticists and the genetic software tools (and bioinformatics pipeline development environments) that they used or know to support their genetic analyses. From this collaboration, we have detected unresolved issues related with customization, integration, and usability.

As a solution to their problems, we have proposed the creation of a DSL for supporting genetic analyses. Instead of learning programming languages, database and web technologies, and operative systems management, they will be provided with a domain-specific language that will avoid them the need to acquire this technical knowledge, and will provide them with the possibility to focus only on domain-related issues.

In the analysis presented in this chapter, we have realized that developing a DSL for the genetic analysis domain is a difficult task that requires overcoming the huge existing gap between developers and geneticists (a gap that is not that huge in other technical domains such as the domain of web applications). In order to overcome this gap, we also realized that we must enhance the participation of geneticists during the development of the DSL. Specifically, we need their participation to ensure that genetic concepts, such as DNA or gene, are well represented, that the DSL structure is suitable according to their preferences, and that the DSL underlies the right software tools for genetic analysis.

As a conclusion, we have detected that in complex domains with high specificity, the feedback from end-users during the DSL development process is essential to ensure that the DSL created represents the domain correctly and fits end-users' needs. For this reason, the motivation of this PhD thesis is to provide an approach that supports the creation of domain-specific languages for complex application domains (not only for the genetic analysis domain) by enhancing the participation of end-users within the DSL development process.

Hence, the goal of this PhD thesis is to propose a method that we can use afterwards to involve geneticists in the development of a DSL for genetic analysis, but may also be used by further developers to create a DSL for another complex application domain.

3. State of the Art

As we have seen in Chapter 2, developing a DSL for a complex domain is a difficult task that requires the collaboration of end-users during the DSL development process. However, traditional and well-established approaches for DSL development are mostly focused on improving the efficiency of developers; neglecting the importance of the end-users during the DSL development process itself.

The aim of this chapter is to find whether there is in the state of the art a DSL development approach that guides and simplifies the development process for developers, ensures that the DSL is developed within the suitable time, and facilitates the gathering of end-users' feedback. Since our main goal is to find an approach that helps overcoming the complexity of the domain and the gap with end-users, we pay special attention on the approaches that involve end-users in the development process.

In the first section, we briefly overview the state of the art of DSL development. In the second section, we propose several analysis criteria to assess whether a DSL development approach is suitable for developing a DSL for a complex application domain. In the third section, we overview the identified related works and assess each of them in regards to the analysis criteria proposed. Finally, in the last section, we discuss the result of the assessment and the issues that remain unsolved.

3.1 State of the art of DSL development

Thanks to the popularity of DSLs like HTML, SQL, or VHDL (and the general believe that DSL development is complex and requires a big effort), the interest of researchers in DSL development has gained a lot of attention, especially in the last decade. As Nascimiento et al. [5] shows in their systematic mapping, the applied analysis criteria found 7 primary studies in 1996 in contrast to the 225 that were found in 2010 (See Figure 3.1 obtained from Nascimiento et al.).

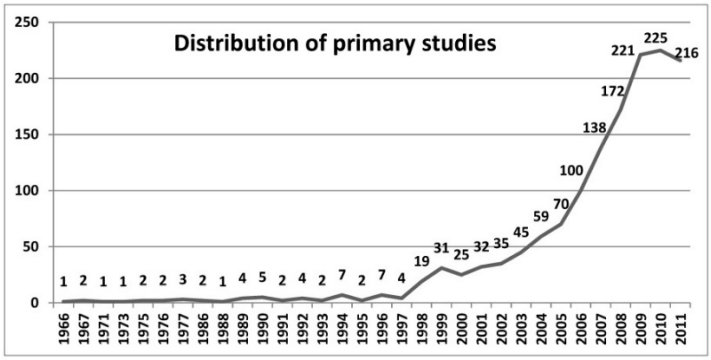


Figure 3.1 Works of the literature about DSLs (from Nascimiento et al.)

Research interests of these works range from defining DSL concepts, providing guidelines for development, describing examples of developments, proposing new techniques, improving certain aspects of the development process, to discussing lessons learned from industrial practice.

Among them, we highlight a set of knowledge base works. In 2000, VanDeursen [3] provided a snapshot of the current state of DSL development, and a brief description of the most relevant works at that time. In addition, Czarnecki [14] presented in his PhD thesis the state of the integration between domain engineering and application engineering methods, providing details about different analysis and design techniques and implementation technologies. Then, Spinellis [13] introduced one of the first methodological works in DSL development, which describes and analyzes a set of design patterns so that other DSL developers could benefit from their use. Finally, Mernik et al. [10] organized the DSL development process in different stages and provided methodological guidelines to guide developers in each of them.

In the last decade (mostly from 2008-2013), we highlight another set of reference works. Kelly and Tolvanen [65] provided a guidebook both to beginners and to advanced developers for the development of DSLs, including discussions about domain abstractions, industrial experience, and code generation from models. Strembeck and Zdun [12] proposed a systematic approach to guide developers in the different decisions to take into account when developing a DSL, by identifying each of these decisions and using decision charts that illustrate each of them. Fowler [4] provided further details and examples about DSL concerns, such as the implementation of external and internal DSLs, options for code generation, and available workbenches. Finally, Voelter [22] overviewed the state of DSL development, starting from explaining the conceptual foundations of DSL design, the design approaches options, the useful IDEs for implementation, and ending explaining the role that DSLs play in software engineering.

During more than a decade, the aforementioned works (among others) have contributed to DSL development by improving conceptual and methodological foundations but also by addressing different aspects of development such as performance and tool support. However, a common absence can be found in all of them: the small attention to the end-user role during DSL development. Traditionally, this role has been neglected because the developers used to play both roles: developer and end-user. Initially, these works were applied to develop DSLs for technical domains with the aim to enhance the productivity of software developers. In this case, the developers usually had enough technical knowledge to develop the DSL. However, nowadays, DSLs are also developed for complex application domains with the aim to involve end-users of non-technical domains. Therefore, in this case, it is unlikely that the developers have such specialized knowledge. Since these DSLs must be especially designed to be used by the end-users of those domains, their consideration during the development process cannot be avoided.

3.2 Analysis criterion

The goal of reviewing the related works of the literature is to find a complete guidance to successfully develop a DSL in close collaboration with end-users. For this reason, in order to identify the contributions of each related work to this goal, we propose to analyze them according to the following analysis criteria: a) process

completeness; b) application of current research about end-user development; and c) end-user involvement.

3.2.1 Process completeness

This criterion analyzes if the approach covers all the stages of DSL development. As a base to define the process completeness, we use the stages proposed by Mernik et al. [10] and the continuation of this work in [11]. We have selected Mernik's work for being one of the most relevant works in the literature regarding methodological guidelines for DSL development and one of the most cited works of the literature by authors that put in practice the development of a DSL. The stages proposed by these works are:

1. Decision Stage: In this stage, the decision whether or not to develop the DSL is made. If the benefits are worth the efforts and the DSL contributions to end-users are clear, the DSL is developed.
2. Analysis Stage: In this stage, a domain analysis is conducted. The domain knowledge is gathered from sources of explicit and implicit knowledge such as technical documentation, GPL code, and from interviews or discussions with end-users. As an explicit representation of all the knowledge gathered, it is created a domain model with the following elements: a) a domain definition defining the scope of the domain; b) domain terminology; c) description of domain concepts; and d) commonalities and variabilities of domain concepts and their interdependencies.
3. Design Stage: In this stage, the language structure (syntax) and the language meaning (semantics) are described. Syntax comprises the abstract and the concrete syntax. Semantics comprise semantic restrictions and behavioral semantics. According to this decomposition, this stage is organized into four activities:
    a. *Abstract Syntax Specification*: Descriptions of the concepts and relationships of the DSL's constructs.
    b. *Concrete Syntax Specification*: Descriptions of the specific notation of the DSL constructs.
    c. *Semantic Restrictions Specification*: Descriptions of additional constraints and relationships that affect the concepts of the DSL constructs.

   d. *Behavioral Semantics Specification*: Descriptions of the meaning of each DSL construct in the domain that is targeted by the DSL.
4. Implementation Stage: In this stage, the complete DSL infrastructure is implemented. As a result, it is created an environment that understands specifications written using the DSL syntax, ensures that specifications written with a different syntax are reported as erroneous, and executes the corresponding set of actions that represent the semantics of that DSL specification.
5. Testing Stage: In this stage, it is checked both that the DSL language can be used by end-users to specify what they wanted and that the corresponding artefacts (models, executable applications, etc.) that are created from DSL specifications fulfil end-users' needs correctly and accurately. Also, the DSL editor is stressed with negative DSL specifications to check that the corresponding errors are informed to end-users with the suitable messages for them to understand the ongoing mistake(s).
6. Deployment Stage: In this stage, developers release a complete DSL to be used by the end-users by themselves.
7. Maintenance Stage: In this stage, new requirements or detected misbehaviors are described in order to be included in the next version of the DSL.

In order to assess process completeness, we check the support of each stage and the support of each of the activities that compose the stage. We characterize each stage or activity as "*Supported (S)*" if the stage or activity is completely addressed; "*Partially-Supported (PS)*" if the stage or the activity is supported but some issues remain unsolved; and "*Not supported (NS)*" if the stage or activity is not addressed or none details have been provided about it.

3.2.2 Application of existing End-User Development (EUD) practices

This criterion assesses if the proposal applies any kind of knowledge from current and previous research regarding end-user development [2]. For instance, best practices for end-user development, behavior heuristics, or end-user oriented artefacts such as sketches. Concretely, for each related work and for each DSL development activity, we provide a brief description of the type of end-user development practice that has been applied.

3.2.3 End-user involvement

This criterion assesses if end-users participate in the different DSL development activities. For those activities in which end-users are involved, we analyze the mechanisms that are proposed to achieve their participation.

Concretely, for each related work and for each DSL development activity, if end-users participate, we provide a brief description of the approach used to involve end-users.

3.2.4 Analysis table

In order to analyze each related work in regards with the three described criteria, we propose the analysis table that is shown in Table 3.1. For each of the stages Analysis, Design, Testing, Deployment, and Maintenance (and the activities that compose each stage), we assess their support by categorizing them with the abbreviation "S, PS, or NS", a brief description of the end-user development practice applied, and a brief description of the approach for involving end-users.

As we can see in the table, we have left the Decision stage outside the comparison because this stage is barely mentioned in the works analyzed. Although its participation is likely, we have not been able to assess whether and how end-users are involved. Likewise, we assessed neither Implementation nor Deployment because both stages usually require a highly technical knowledge for end-users to participate.

Table 3.1 Analysis table

Stage	**Activity**	**Support (S,PS,NS)**	**EUD practices**	**End-user Involvement**
Analysis	Domain Analysis			
	Domain Model Specification			
Design	Abstract Syntax Specification			
	Concrete Syntax Specification			
	Semantic Restrictions Specification			
	Behavioral Semantics Specification			
Testing	DSL infrastructure testing			
Maintenance	New requirements addition			

3.3 Analysis execution

An approach to ensure that a DSL has been developed according to the end-users needs is to include them as much as possible in the process. However, since end-users availability can be limited and the technical level of DSL development activities can be difficult for end-users, it is necessary to establish a balance between quality and feasibility of end-user participation. In the search of this balance, we found three different types of approaches to develop DSLs taking into account end-users needs.

First, we found works that although do not involve end-users in the DSL development process, they take into account end-users needs by applying ideas proposed in previous research works about end-user development, such as best practices, patterns, heuristics, etc. Examples of this approach are Perez et al. [66], which applies best practices of end-user development, and Nishino [67, 68], which uses cognitive dimensions and language heuristics.

Second, we found works that propose to involve end-users more than traditional approaches by adopting an agile development process: dividing the DSL development into iterations and involving end-users at the beginning and the end of each of them. Examples of this approach are Sadilek [69], which introduces the notion of agile language engineering, and Barisic et al. [70], which involves end-users at different iterations to ask about the DSL usability.

Third, we found works that propose to involve end-users in the creation of different artefacts of a DSL. In order to engage end-users in this task, these works propose mechanisms that simplify the technical level so they can understand easily the procedure to be accomplished. Examples of this approach are Wuest et al. [71], Cho et al. [72], Kuhrman et al. [73], and Sanchez-Cuadrado et al. [74], which propose a friendly environment for describing domain examples, and Canovas et al. [75], which proposes a collaborative infrastructure for end-users and developers to collaborate in the design of the DSL syntax.

Next, we describe each of these works with detail, discuss their contributions towards end-user involvement in DSL development, and fulfil the analysis table to compare these works. Among them, we did not analyze in detail the work of Sadilek et al. “Towards an Agile Language Engineering”. This work provides an introduction to agile language engineering by analyzing: 1) how to organize the development process in iterations; 2) the different roles involved in the process;

3) the different available formalisms to describe a language; 4) the possible implementation approaches, etc.; and 5) an illustrative example about the seismology domain. Despite being a relevant related work that describes important background to take into account for DSL development, we have discarded it because it only describes the general course of action instead of describing a development method in detail.

3.3.1 Towards the involvement of end-users within model-driven development

The motivation of Perez et al. [66] is to involve end-users in a model-driven development (MDD) process. Since end-users do not usually know about domain-specific modelling languages and modelling tools as professionals do, their goal is to develop a modelling language with a good usability so that end-users are engaged to participate in a future model-driven development process.

As an approach to develop a more suitable DSL for end-users, this work applies the following best practices from end-user development. For example:

- End-users should be provided with a Domain-Specific Visual Language (DSVL)
- End-users should focus on user-dependent properties, whereas software engineers should focus on quality or maintenance properties.
- End-users should use a library of components as a starting point in order to customize their system.
- End-users should be supported by specific tools that are made especially for them.

With these practices in mind, they propose a method of six steps that takes, as input, an existing DSL and a MDD approach and obtains, as output, a visual DSL supported by a tool with a graphical interface. The steps of the method are:

1. Identify the properties of the system that require end-user participation (analysis). As a result, the variabilities and commonalities of the domain are described using a feature model.
2. Select a visual DSL: According to the best practice “end-users should be provided with a DSVL”, the visual syntax of this DSL will become the concrete syntax to be used by end-users.

3. Design the base system of the new DSL from the original DSL (abstract syntax design).
4. Design the components that help end-users to complete the base system: According to the best practice "end-users have to use a library of components as a starting point in order to customize their system", the variabilities in the DSL will be configured by means of components (behavioral semantics).
5. Define mappings between the input DSL and the visual DSL that has been selected in step 2 (behavioral semantics).
6. Create the tool support of the visual DSL (implementation).

3.3.1.1 Discussion

This approach takes into account end-users needs during DSL development by applying best practices of end-user development during the development of different DSL artefacts. However, the specific requirements and needs of the target end-users are not directly asked to them during the DSL development process.

As we can see in Table 3.2, the Analysis stage is partially supported (PS). Domain Analysis is supported (S) but Domain Model Specification is partially supported (PS) because although the commonalities and variabilities of the domain are made explicit, the other elements of the domain model are not described.

The Design stage is also partially supported (PS). Abstract Syntax Specification, Concrete Syntax Specification, and Behavioral Semantics Specification are supported (S). However, none reference is made of how semantic restrictions could be described (NS). Regarding end-users needs, this approach applies the use of visual syntaxes to design the concrete syntax of the DSL and the use of existing components as a starting point to define the behavioral semantics. End-users are not involved in any design activities.

Since stages Testing and Maintenance are not mentioned, we characterize them as not-supported (NS).

In this work, end-users are not directly involved during the development process. For instance, during the design of the concrete syntax, end-users are taken into account because a visual DSL is selected; however, the specific end-users have no say about which visual syntax they prefer, neither about the

possibility to customize the visual syntax selected. In fact, the proposal always creates visual DSLs to improve usability, which may not be the best approach for some end-users.

Likewise, during the design of behavioral semantics, end-users are provided with a library of components that have been preselected by developers, without intervention of end-users. Although it is true that not every end-user will be aware of the existing components of their domain, it is also true that in the end, they are the experts of the domain. In fact, they are more likely to know current software that implements different domain-related behaviors, even if they do not know or understand their underlying implementation details.

Table 3.2 Assessment of Perez et al.'s work

Stage	**Activity**	**Support (S,PS,NS)**	**EUD practices**	**End-user Involvement**
Analysis	Domain Analysis	S	-	-
	Domain Model Specification	PS	-	-
Design	Abstract Syntax Specification	S	-	-
	Concrete Syntax Specification	S	Visual DSLs	-
	Semantic Restrictions Specification	NS	-	-
	Behavioral semantics Specification	S	Components Library	-
Testing	DSL infrastructure testing	NS	-	-
Maintenance	New requirements addition	NS	-	-

3.3.2 Misfits in abstractions: Towards user-centred design in DSLs for end-user programming

The motivation of Nishino [67] is to solve the usability problems of DSLs caused by a bad design. According to authors, if developers do not have the suitable domain knowledge during DSL development, inappropriate abstractions occur as a consequence of conceptual misfits.

For this reason, this work proposes to identify usability problems by analyzing a set of cognitive dimensions proposed by Blackwell et al. [76] and a set of feature heuristics proposed by Sadowski et al. [77]. Concretely, they assess the language according to a set of cognitive dimensions, such as quick to learn, quick to apply,

applicable at any stage of design, etc.; and a set of feature heuristics such as abstraction gradient, consistency, error proneness, hidden dependencies or error recovery, among others.

After identifying the list of usability problems, developers propose a redesign that tackles with all these problems but also to assess this new design before it is implemented. As an illustrative example, they apply this approach to design a DSL in the music domain.

3.3.2.1 Discussion

In the context of DSL development, this work can be placed as an approach that addresses the testing of a DSL: the approach analyses the usability problems of an existing DSL and proposes to redesign some parts of the DSL to solve them. However, no additional details are provided about how to evolve the different elements of the DSL, that is, it is not explained how to proceed to apply changes in a potential existing analysis model, how to evolve the design models, or the implementation infrastructure.

As we can see in Table 3.3, all stages are not supported (NS) but testing, which is partially supported (PS). This approach is partially supported because it addresses the testing of usability problems, but it does not test for example, the fulfilment of functional requirements. In this stage, previous research about end-user development is applied to detect usability problems but end-users are not involved in it.

Table 3.3 Assessment of Nishino's work

Stage	**Activity**	**Support (S,PS,NS)**	**EUD practices**	**End-user Involvement**
Analysis	Domain Analysis	NS	-	-
	Domain Model Specification	NS	-	-
Design	Abstract Syntax Specification	NS	-	-
	Concrete Syntax Specification	NS	-	-
	Semantic Restrictions Specification	NS	-	-
	Behavioral Semantics Specification	NS	-	-
Testing	DSL infrastructure testing	PS	Cognitive dimensions and feature heuristics	-
Maintenance	New requirements addition	NS	-	-

3.3.3 How to reach a usable DSL? Moving toward a systematic evaluation

The two works of Barisic et al. [78, 70] state the importance of addressing usability concerns early in the DSL development process. Specifically, their idea is assessing the quality in use of the DSL before it is completely implemented.

With this aim, this work proposes a DSL development and evaluation process that divides the DSL development into iterations. At the end of each iteration, end-users are asked about the quality in use of the DSL. To assess this quality, authors studied the usability dimensions of the quality standard ISO IEC CD 25010.3 and built a quality model [78] with the following quality dimensions: understandability, readability, efficiency, effectiveness, learnability, flexibility, expressiveness, freedom of risk, and satisfaction. To assess the DSL, they use a goal-question-metric approach that is based on that model. This approach describes a set of goal-question-metric trio for each quality dimension [70]:

- **Goal**: the usability dimension to be evaluated. For example: "*Effectiveness*"
- **Question**: The end-users' perception about the usability dimension. For example: "*Is the user able to specify all parts of the example?*"
- **Metric**: Set of attributes that assess the degree of achievement of the usability dimension. For example: *"Number of errors while describing the example"*.

In order to illustrate the proposal, it has been applied in the usability assessment of a DSL for specifying humanitarian campaign processes (named FlowSL). Two cycles of the DSL development process have been conducted, in which four evaluations with two end-users were performed.

3.3.3.1 Discussion

This work tackles with a very important aspect of the DSL success that is usability and highlights the importance of involving end-users early in the DSL development to evaluate such usability.

In their first proposal, they mentioned different ideas for quality assessment to be included in the stages Analysis, Design, Implementation, and Testing. However, in their most recent work, authors presented their quality assessment approach only for the Testing stage, without further details about how to address the different stages Analysis, Design, and Implementation and how these stages

contributed to the quality assessment. Accordingly, in Table 3.4, we characterize the different activities of the stages Analysis and Design as not supported (NS).

The proposal describes how end-users are involved in the Testing stage at the end of each iteration of DSL development. It describes the roles of different end-users and the usability questions that were used for one specific use case. However, it does not provide much detail about the approach such as the specific tasks that must be followed to perform the DSL evaluation, how the metrics obtained should be interpreted to assess each usability dimension, or how to deal with the feedback provided by end-users. Because of these missed details and the lack of support for testing functional requirements, in Table 3.4, we characterize the Testing stage as partially supported (PS). In this stage, none EUD practices is applied but end-users are involved in the usability assessment by means of the goal-question-metric approach.

Finally, the approach does not mention any maintenance activity, so we characterized it as not supported (NS).

Table 3.4 Assessment of Barisic et al.'s work

Stage	**Activity**	**Support (S,PS,NS)**	**EUD practices**	**End-user Involvement**
Analysis	Domain Analysis	NS	-	-
	Domain Model Specification	NS	-	-
Design	Abstract Syntax Specification	NS	-	-
	Concrete Syntax Specification	NS	-	-
	Semantic Restrictions Specification	NS	-	-
	Behavioral Semantics Specification	NS	-	-
Testing	DSL infrastructure testing	PS	-	Usability (quality in use): Goal-Question-Metric Approach
Maintenance	New requirements addition	NS	-	-

3.3.4 Semi-automatic generation of metamodels from model sketches

The motivation of Wuest et al. [71] is to facilitate the metamodeling activity to non-experts and to provide modelers with additional freedom when using metamodeling tools. For example, by supporting the free sketching during a brainstorming session. In the context of DSL development, their goal is to create

domain models together with end-users by means of sketches and to seamlessly obtain the corresponding abstract syntax metamodel of the DSL.

With this aim, the authors have created FlexiSketch, an environment for modelers and end-users to design together the examples of the domain using sketches. For the generation of the metamodel, the environment identifies the different parts of the sketches as nodes and edges (see Figure 3.2).

From nodes and edges, and with additional metamodeling information (such as types and cardinalities), an inference algorithm (proposed by the authors) is able to infer the entities, the attributes, the relationships, and the cardinalities of the metamodel. The environment uses wizards to guide developers in the addition of this extra information that cannot be found in sketches. Besides the inference of the metamodel, the sketches themselves can be proposed afterwards as candidate elements for the concrete syntax.

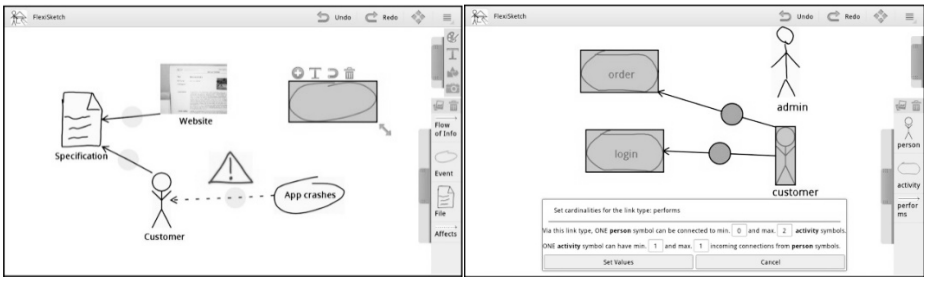


Figure 3.2 The FlexiSketch environment (extracted from Wuest et al.)

3.3.4.1 Discussion

This approach provides a friendly solution for end-users and developers to analyze the domain by means of domain examples. These examples are used for the creation of the DSL syntax, both the abstract syntax and the concrete syntax. However, these examples do not completely represent the complete domain model explicitly (scope, terminology, concepts, and commonalities and variabilities).

Accordingly, in Table 3.5, Domain Analysis, Abstract Syntax Specification, and Concrete Syntax Specification are characterized as supported (S). However, Domain Model Specification is characterized as not supported (NS). In these supported activities, end-users are involved by using of sketches (through the environment FlexiSketch), a well-known technique in the end-user development field.

During the metamodel inference, developers may add cardinality constraints between entities but they are not able to define further constraints. For this reason, we characterize Semantic Restrictions Specification as partially-supported (PS). In this activity, neither research about end-user development nor participation by end-users is included.

This work is not a method for DSL development, so the other stages and activities of DSL development are not supported. For example, neither behavioral semantics are mentioned as a part of the proposal nor how to implement, test, deploy or maintain the DSL. For this reason, the Testing and Maintenance stages are characterized as not supported (NS).

Table 3.5 Assessment of Wuest et al.'s work

Stage	**Activity**	**Support (S,PS,NS)**	**EUD practices**	**End-user Involvement**
Analysis	Domain Analysis	S	Sketches	Sketches (FlexiSketch environment)
	Domain Model Specification	NS	-	-
Design	Abstract Syntax Specification	S	Sketches	Sketches (FlexiSketch environment)
	Concrete Syntax Specification	S	Sketches	Sketches (FlexiSketch environment)
	Semantic Restrictions Specification	PS	-	-
	Behavioral semantics Specification	NS	-	-
Testing	DSL infrastructure testing	NS	-	-
Maintenance	New requirements addition	NS		

3.3.5 Creating visual DSMLs from end-user demonstration

Like the previous work (Wuest et al.), the motivation of Cho et al. [72] is to involve end-users in the metamodeling task of DSL development by means of sketches. Similarly, this work infers from the sketches the DSL syntax and the semantic restrictions.

The proposal starts when end-users and developers draw together a set of domain examples using sketches. From them, unique graphical shapes are identified and end-users choose the ones that are suitable to become elements of the concrete syntax.

Then, the abstract syntax and the semantic restrictions are inferred using graph theory. First, the domain examples sketches are transformed into graph representations and then, an inference engine (implemented by the authors) obtains the metamodel from those graphs. This engine requires training data, both positive and negative examples, but also, as a part of this training, the engine uses a set of design patterns that have been proposed by the authors in a previous work [79]. According to the authors, these patterns represent common features of DSLs and can be used to refine, and thus improve, the metamodel obtained by the engine.

Regarding semantic restrictions, end-users review the domain sketches in order to establish association links between their elements. During the metamodel inference, the engine asks for feedback about those constraints by providing a set of options that they must choose.

3.3.5.1 Discussion

This approach provides a friendly solution for end-users to describe domain examples, the creation of the DSL syntax, and the semantic restrictions. Nevertheless, the approach still does not make explicit the domain model details, nor addresses other activities of DSL development such as behavioral semantics design, testing, deployment, or maintenance.

Accordingly, the analysis table (Table 3.6) shows that Domain Analysis, Abstract Syntax Specification, Concrete Syntax Specification, and Semantic Restrictions Specification are supported (S). The rest of activities are not supported (NS). End-users create domain examples for the domain analysis, select the most suitable shapes of those sketches for the concrete syntax specification; create links between elements of the sketches for the semantic restrictions specification, and answer inference questions about constraints in the sketches for the semantic restrictions specification as well.

Table 3.6 Assessment of Cho et al.'s work

Stage	Activity	Support (S,PS,NS)	EUD practices	End-user Involvement
Analysis	Domain Analysis	S	Sketches	Design example sketches
	Domain Model Specification	NS	-	-
Design	Abstract Syntax Specification	S	-	-
	Concrete Syntax Specification	S	Sketches	Shape selection and sketches metadata
	Semantic restrictions Specification	S	Sketches	Sketches links and engine inference questions
	Behavioral semantics Specification	NS	-	-
Testing	DSL infrastructure testing	NS	-	-
Maintenance	New requirements addition	NS	-	-

3.3.6 Rapid prototyping for DSLs: From stakeholder analyses to modelling tools

The motivation of Kuhrman et al. [73] is to bring together DSL developers and domain experts when developing DSLs in complex application domains. The author's proposal aims to assist developers to capture the suitable knowledge from end-users and represent it into the DSL.

This work proposes a DSL named "*PDE language*", whose aim is to facilitate the creation of DSLs, both to developers and end-users. First, the PDE language editor provides a digital panel in which end-users draw examples of their domain. To do that, they can choose graphical shapes from images that are saved in the file system or predefined shapes from the palette and create links among images and shapes. Eventually, these shapes, images, and links will become into elements of the concrete syntax. From these domain examples, the PDE language editor creates an instance of the PDE language. This instance, besides being a PDE language instantiation, it is also the abstract syntax metamodel of the DSL to be created.

Once a preliminary version of the abstract and the concrete syntax have been obtained, the editor generates a visualization model that shows different aspects of the DSL (Figure 3.3). This way, developers and end-users can easily refine the different DSL details. This editor has a view to validate the abstract syntax metamodel, another view to customize the graphical concrete syntax, and a view

to program validation functions that ensure the fulfilment of semantic constraints. Once the DSL design is finished, the tool implements the corresponding DSL.

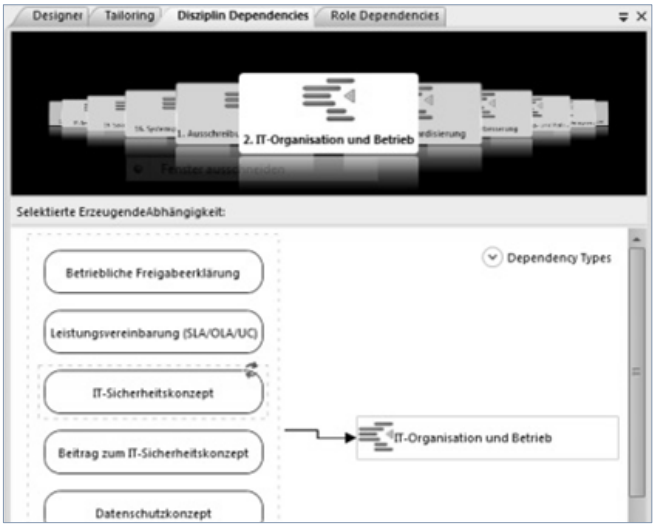


Figure 3.3 The PDE language visualization model (extracted from Kuhrman et al.)

3.3.6.1 Discussion

This work provides a DSL as a friendly environment to facilitate the creation of DSLs to both end-users and developers. However, authors acknowledge that usability is still an issue to improve and only end-users with some DSL development experience are able to accurately understand the whole environment.

First, the approach supports the Analysis and Design stages and end-users are involved in both of them. Regarding the Analysis stage, Domain Analysis is supported (S) because the PDE language editor provides an environment for the creation of domain examples. In this activity, end-users are involved by means of sketches, more specifically, through the graphical interface of the PDE language editor. However, like the previous works, Domain Model Specification is not supported (NS), since no scope, terminology, concepts, and commonalities and variabilities are explicitly described.

Regarding the Design stage, Abstract Syntax Specification, Concrete Syntax Specification, and Semantic Restrictions Specification are supported by the PDE editor (S). Although the PDE editor provides a visualization model with different views to facilitate the participation of end-users in those design tasks, actually end-users can only participate in the concrete syntax design by selecting their

graphical elements. As authors admit, some views are still difficult for end-users, such as the view for refining the abstract syntax metamodel or the view for programming validation functions.

Finally, no further stages are presented after the implementation of the DSL editor. For this reason, we characterize Testing and Maintenance as not supported (NS).

Table 3.7 Assessment of Kuhrman et al.'s work

Stage	**Activity**	**Support (S,PS,NS)**	**EUD practices**	**End-user Involvement**
Analysis	Domain Analysis	S	Sketches	Graphical interface of the "PDE Language" editor
	Domain Model Specification	NS	-	-
Design	Abstract Syntax Specification	S	Views	-
	Concrete Syntax Specification	S	Views	Graphical elements within the "PDE Language" editor
	Semantic restrictions Specification	S	Views	-
	Behavioral semantics Specification	NS	-	-
Testing	DSL infrastructure testing	NS	-	-
Maintenance	New requirements addition	NS	-	-

3.3.7 Bottom-up meta-modelling: An interactive approach

Similarly to the works [71], [72], and [73], the motivation of Sanchez-Cuadrado et al. [74] is to support the use of informal drawing tools as a friendly interface that facilitates the metamodeling task. The goal of this work is enhancing end-user participation within the DSL development process by using these tools to sketch a set of domain examples. The difference of this work in respect to the others is the approach used to generate the metamodel from the domain examples sketches.

In this approach, during Domain Analysis, end-users are encouraged to draw a set of sketches that represent examples of their domain and designers are responsible to annotate these examples with additional domain information (Figure 3.4). An example of domain annotation is the *intention* of a graphical element of a domain example.

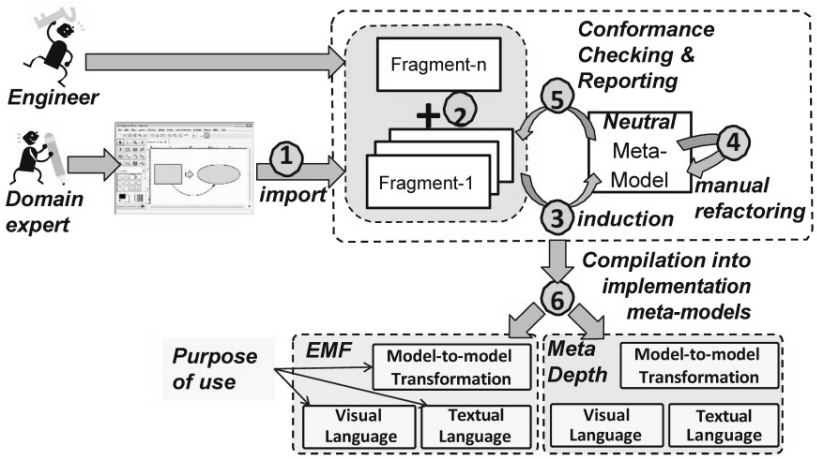


Figure 3.4 Overview of the bottom-up meta-modelling approach (extracted from Sanchez-Cuadrado et al.)

From these examples and annotations, the corresponding metamodel that complies with these domain examples is induced. This metamodel is obtained iteratively, one example at a time, in which developers are able to assess the evolution of the metamodel and the specific effects of each domain example over the metamodel.

Finally, designers supervise and refine the metamodel according to their metamodeling design expertise. If some changes have been applied, a procedure checks for possible mismatches between the final metamodel and the domain examples.

Another contribution of this work is that the metamodel generated is platform independent. For this reason, in the last step of the proposal, a specific platform is selected to implement the metamodel.

3.3.7.1 Discussion

This work proposes an approach to facilitate the participation of end-users in DSL development: creating the domain examples and inducing the abstract syntax metamodel from them. The proposal only focuses on the domain analysis and the metamodeling task but other DSL development activities are not addressed.

Accordingly, in Table 3.8, only the activities Domain Analysis and Abstract Syntax Specification are supported (S) and end-users participate in the definition

of domain examples by means of sketches using an informal panel. Like the previous approaches, domain model specification is not supported (NS) because, although the domain is described by a set of examples, the domain model elements are not explicitly specified.

The rest of the stages and activities are not detailed, so we characterize all of them as not supported (NS).

Table 3.8 Assessment of Sanchez-Cuadrado et al.'s work

Stage	Activity	Support (S,PS,NS)	EUD practices	End-user Involvement
Analysis	Domain Analysis	S	Sketches	Informal environment to design example sketches
	Domain Model Specification	NS	-	-
Design	Abstract Syntax Specification	S	-	-
	Concrete Syntax Specification	NS	-	-
	Semantic restrictions Specification	NS	-	-
	Behavioral semantics Specification	NS	-	-
Testing	DSL infrastructure testing	NS	-	-
Maintenance	New requirements addition	NS	-	-

3.3.8 Collaboro: Enabling the collaborative definition of DSMLs

The motivation of Canovas et al. [75] is to highlight the importance of the end-users role in the definition of DSLs and provide means to enable the collaboration between end-users and developers in the context of DSL development. With this aim, they propose a community-driven DSL development process to encourage end-user participation in the definition of DSLs.

In order to involve end-users, they take the traditional DSL development process as a basis and modify each stage to be iterative, i.e. the process only proceeds with the next stage when end-users completely agree with the outcome of the current stage. In each stage, end-users and developers collaborate to create the different DSL artefacts.

The collaboration among end-users and developers is supported by a DSL named Collaboro. Collaboro describes the elements of the collaborative activity (comment, vote, solution, etc.) and the elements of the abstract and concrete syntax of a DSL (entity, attribute, relationship, textual notation, etc.). These elements are used to track the evolution of both the abstract and concrete syntax.

The collaborative development starts after requirements gathering, when developers design a preliminary abstract and concrete syntax (Figure 3.5, Step 1). As a way to ensure that end-users understand the syntax that has been designed by developers, end-users are provided with domain examples that have been rendered using the concrete syntax. Then, end-users are able to comment, propose solutions, and vote other participants opinions and proposals (Figure 3.5, Step 2). This interaction continues until the syntax, after all the changes proposed and applied (Figure 3.5, Step 3), satisfies end-users completely.

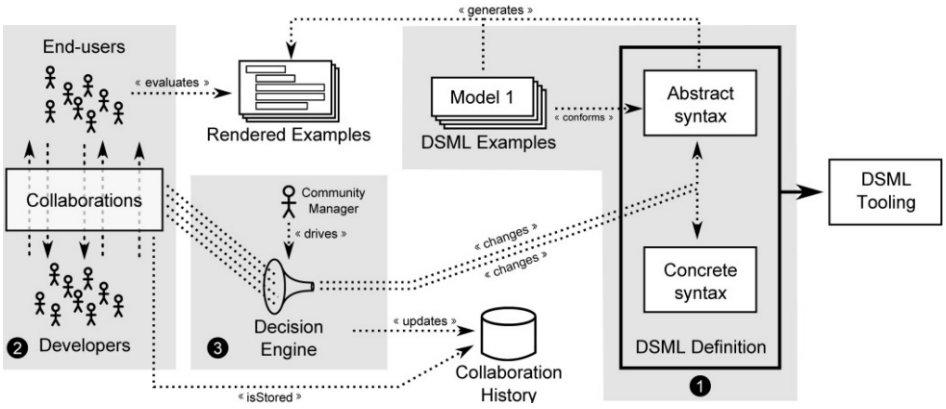


Figure 3.5 Overview of the Collaboro approach (extracted from Canovas et al.)

3.3.8.1 Discussion

This work provides a collaborative infrastructure to design the abstract and the concrete syntax of a DSL. However, at the moment, this work is still under development, for that reason, not all the stages of the process have been addressed yet. On the one hand, this work assumes a previous analysis of the domain has been conducted, so Domain Analysis and Domain Model Specification are characterized as not supported (NS).

This work focuses only on syntax design. Accordingly, in Table 3.9, we characterize Abstract Syntax Specification and Concrete Syntax Specification as supported (S) but Semantic Restrictions Specification, Behavioral Semantics

Specification, Testing, and Maintenance as not supported (NS). This work, unlike the previous ones, is the first work that provides a friendly mechanism for involving end-users both in the definition of the abstract syntax and the concrete syntax. Instead of participating in the syntax specification, this approach renders a set of examples to illustrate the syntax proposed. This way, end-users check the different syntax elements in their domain context, and they are able to provide more accurate feedback. This feedback is provided by means of the Collaboro environment, an Eclipse-based tool. End-users comment about the syntax but also about the changes that are proposed by developers.

As an additional contribution, this work provides a collaborative infrastructure that tracks all the design decisions that are made during design in order to have a better traceability of the collaboration process.

Table 3.9 Assessment of Canovas et al.'s work

Stage	Activity	Support (S,PS,NS)	EUD practices	End-user Involvement
Analysis	Domain Analysis	NS	-	-
	Domain Model Specification	NS	-	-
Design	Abstract Syntax Specification	S	-	Collaborative infrastructure (Collaboro)
	Concrete Syntax Specification	S	-	Collaborative infrastructure (Collaboro)
	Stemantic restrictions Specification	NS	-	-
	Behavioral semantics Specification	NS	-	-
Testing	DSL infrastructure testing	NS	-	-
Maintenance	New requirements addition	NS	-	-

3.3.9 Engaging end-users in the collaborative development of DSMLs

The motivation of the two previous works [74, 75] was to improve DSL development by supporting the involvement of end-users. Since each of their works addressed different activities of the process, their authors integrated their approaches under the same approach context to increase the completeness of the approach [80].

This new work proposes a DSL development process (Figure 3.6) with 5 steps. First, end-users (aided with developers) use informal drawing tools to create domain examples. Second, from those domain examples, the abstract syntax

metamodel is induced according to the approach of Sanchez-Cuadrado et al. [74]. Third, after obtaining the metamodel, the collaborative infrastructure Collaboro, proposed by Canovas et al. [75], is used by end-users and developers to propose potential changes to the abstract syntax metamodel. Additionally, a recommender system also identifies potential changes according to metamodel-quality patterns. Fourth, from all the proposed changes, some of them are accepted and incorporated to the abstract syntax. The proposal iterates over the steps three and four until no more changes are pending. Finally, in the fifth step, the final version of the abstract syntax metamodel is implemented.

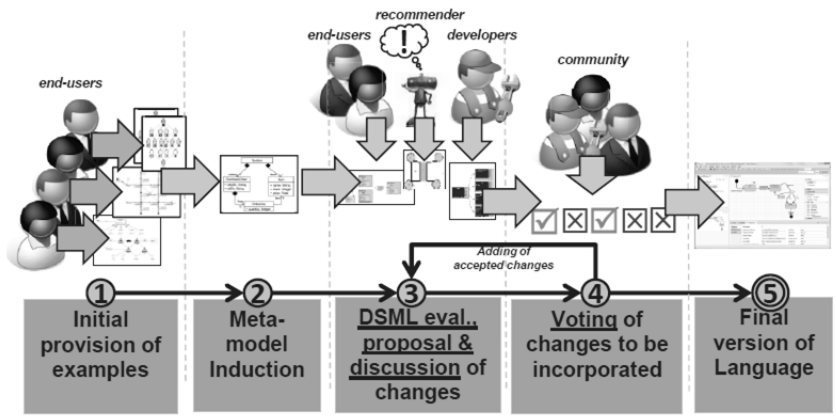


Figure 3.6 Overview of the combined approach (extracted from Canovas et al.)

3.3.9.1 Discussion

The combination of two involving approaches in the context of DSL development brings together the contributions of both works, complementing activities that were not addressed individually. As a result, the approach starts from a metamodel obtained from end-users participation, instead of starting the proposal from a metamodel proposed by developers, and then, a collaborative environment allows discussion and tracking of syntax changes.

In summary, the main contribution of this work as a whole, in contrast of previous works, is the use of friendly mechanisms, such as the use of an informal panel and the use of examples, as a way to reason about the domain and the abstract and concrete syntax.

Accordingly, in Table 3.10, Domain Analysis, Abstract Syntax Specification and Concrete Syntax Specification are supported (S). End-users are involved in

all of them by means of domain sketches and by means of examples; first through an informal environment and then, through the collaboro infrastructure.

As we already mentioned in the analysis of the work of Sanchez-Cuadrado, the approach analyzes the domain using examples, but no domain model is specified explicitly, so the activity is characterized as not supported (NS). Likewise, Semantics Specification (restrictions and behavior), Testing, and maintenance are not supported (NS).

Table 3.10 Assessment of Canovas et al.'s work

Stage	**Activity**	**Support (S,PS,NS)**	**EUD practices**	**End-user Involvement**
Analysis	Domain Analysis	S	Sketches	Informal environment to design example sketches
	Domain Model Specification	NS	-	-
Design	Abstract Syntax Specification	S	-	Collaborative infrastructure (Collaboro)
	Concrete Syntax Specification	S	-	Collaborative infrastructure (Collaboro)
	Semantic restrictions Specification	NS	-	-
	Behavioral semantics Specification	NS	-	-
Testing	DSL infrastructure testing	NS	-	-
Maintenance	New requirements addition	NS	-	-

3.4 Discussion

As a result of the previous analysis, Table 3.11 compares the described related works by observing the end-user involvement through the complete DSL development cycle. For each activity of the DSL development, we analyze the support of the activity and the end-users involvement. We indicate “S” for supported, “PS” for partially supported (when some issues of the activity remain unsolved), and "x" for not supported.

Table 3.11 Comparison of state of the art works

Stage	Activity	Criteria	Perez	Nishino	Barisic	Wuest	Cho	Kurhman	SCuadrado	Canovas
Analysis	Domain Analysis	Support	S	x	x	S	S	S	S	x
		EU Inv	x	x	x	S	S	S	S	x
	Domain Model Specification	Support	PS	x	x	x	x	x	x	x
		EU Inv	x	x	x	x	x	x	x	x
Design	Abstract Syntax Specification	Support	S	x	x	S	S	S	S	S
		EU Inv	x	x	x	S	S	x	x	S
	Concrete Syntax Specification	Support	S	x	x	S	S	S	x	S
		EU Inv	x	x	x	S	S	S	x	S
	Semantic Restrictions Specification	Support	x	x	x	PS	S	S	x	x
		EU Inv	x	x	x	x	S	x	x	x
	Behavioral Semantics Specification	Support	S	x	x	x	x	x	x	x
		EU Inv	x	x	x	x	x	x	x	x
Testing	DSL infrastructure testing	Support	x	PS	PS	x	x	x	x	x
		EU Inv	x	x	PS	x	x	x	x	x
Maintenance	New requirements addition	Support	x	x	x	x	x	x	x	x
		EU Inv	x	x	x	x	x	x	x	x

As we can see in the table, there isn't any work that involves end-users in DSL development and also supports the complete DSL development process. All of them focus on involving end-users in the development of some aspect of the DSL, but none of them integrates their proposal within a complete DSL development process.

The majority of the works focus on the Analysis and Design stages and two of them in the Testing stage, but none has provided a proposal for the Maintenance stage. We can also observe that Domain Analysis and Concrete Syntax Specification are the activities in which end-users are most involved.

Regarding the Analysis stage, the majority of the proposals have successfully involved end-users in the domain analysis, although only in one of them, the domain knowledge gathered in this analysis is made explicit. The other approaches gather domain examples, but although these examples embed domain

knowledge, they do not explicitly describe the domain scope, all the domain concepts and their inter relationships, the details of the domain terminology, or the domain commonalities and variabilities.

The formalization of the domain knowledge into a domain model (according to the definition of Mernik et al.), although it is not essential to develop a DSL as an input required by other stages, it is very important to understand the domain, especially for complex application domains, in which the gap between developers and end-users is wide. In this kind of domains, the formalization of the domain model becomes essential to ensure that the domain is well understood by developers without ambiguities, to understand the rationale of development decisions, to understand any possible evolution occurred in the domain, and the possibility to use this model as an artefact for model-driven development.

All in all, in the analyzed works, although the domain knowledge is acquired and represented informally and the domain model is not created, eventually, this knowledge is formalized usually in the form of a metamodel that represents the abstract syntax of the DSL.

Regarding the Decision stage, the majority of works propose involving end-users in some aspects of the syntax specification. However, only one of them successfully involves end-users in Abstract Syntax Specification, only three of them in Concrete Syntax Specification, but any of them in Semantics Restrictions Specification or Behavioral Semantics.

In conclusion, we have observed two main lacks in the state of the art. First, as far as we are concerned there is not any approach that addresses how to involve end-users in the specification of the behavioral semantics, the testing of the DSL infrastructure, and the maintenance of the DSL, which are three aspects of the DSL in which the collaboration of end-users is also important. Second, we have observed that these proposals focus their work only on a subset of stages of the complete DSL development process. Although all of them have importantly contributed to improve the involvement of end-users in DSL development, none of them continues their proposal through the complete DSL development life-cycle. These works explain the analysis and design artefacts of their proposals. However, they omit how to use those artefacts to implement the complete DSL development infrastructure such as the different steps to follow, the artefacts involved in each step, the common patterns to take into account during development, or the potential problems that could arise during development.

3.5 Conclusion

In this chapter, we have analyzed the state of the art of DSL development, focusing our attention on proposals that involve end-users in the different stages of DSL development.

As a result of this analysis, we have found that traditional DSL development approaches provide useful guidelines for developers, but do not properly consider the role of end-users. In contrast, the approaches that consider the role of end-users are still ongoing works and all of them only focus on one or two stages of DSL development. As a consequence, although all these approaches involve end-users in the specification of some aspect of the DSL, there is still not a method to support the whole DSL development life-cycle.

In conclusion, this analysis provides evidences that the problem that addresses this PhD has not been completely solved: "*Provide a methodological approach to involve end-users in domain-specific languages development (for complex application domains)*". As far as we known, there is a need of a complete DSL development method that: 1) provides guidance throughout the complete DSL development life-cycle, so it can be adopted in real practice (Requirement 1); 2) ensures the feasibility of the DSL development time (Requirement 2); and 3) facilitates the gathering of domain experts' knowledge in the stages in which they can collaborate (such as Analysis, Design, Testing, and Maintenance) (Requirement 3).

4. Method Overview and Illustrative Example

Developing DSLs for a complex domain is a challenge for developers, since it implies understanding the domain complexity and overcoming the existing gap between developers and end-users. In order to reduce this complexity, developers need to be provided with a DSL development approach that involves end-users throughout the process. However, as we have shown in the previous chapters, traditional DSL development approaches usually neglect the importance of the end-user role and the ones which involve end-users have still some unresolved issues.

Involving end-users in DSL development requires the process to be clear, efficient, and engaging for end-users. The goal is facilitating end-users the comprehension of the different artefacts of the DSL, so they can contribute in their creation, and achieve that those artefacts represent their knowledge, needs, and preferences. With this aim, we believe that the combination of model-driven development practices and agile practices will provide efficiency, end-user engagement, and the formalisms necessary to create the different DSL artefacts.

As a solution, we propose an agile model-driven method to involve end-users in DSL development. The method is organized in different stages according to

the DSL life-cycle. In each stage, we have selected a set of artefacts to describe the different DSL aspects and proposed a set of mechanisms, based on agile practices, to involve end-users in the creation of those artefacts.

In this chapter, we start explaining the approach followed to build the method. Then, we explain the rationale of the combination of model-driven and agile practices. Next, we describe how those practices were applied to build the method and we provide a brief overview of this method. Finally, in order to facilitate the comprehension of the method for the reader, we introduce the example that is used in the next chapters to illustrate the method.

4.1 Building a method for DSL development

In order to propose a method to involve non-technical end-users in the DSL development process, we needed the collaboration of end-users as well. We wanted to propose a set of activities in the method to involve end-users. But in order to be sure about their suitability, we needed end-users without knowledge of software development to provide feedback about them.

With this aim, we established a collaboration with the geneticists from Imegen and INCLIVA (as we explained in Chapter 2) and the geneticists from the SME GEM Biosoft[6]. We collaborated with geneticists because they fit the profile of users without software development knowledge. Actually, the aim of this collaboration was two-fold: besides participating in the design of the method, just right after the design, they participated in applying the method to develop their own DSL, so we had again the opportunity to gather further feedback about the method.

The approach to build our proposal (Figure 4.1) started by searching state-of-the-art proposals to develop DSLs (explained in Chapter 3) and agile methods (explained next in Section 4.3). From them, we adopted guidelines, best practices, and lessons learned. Then, in collaboration with geneticists, we proposed the first draft of the method (Figure 4.1, step 2), we applied it to create the first draft of the DSL prototype (Figure 4.1, step 3), and we gathered feedback about this draft

[6] Genome Knowledge Software, GEM Biosoft. http://www.gembiosoft.com/

of method (Figure 4.1, step 4). We iterated over the steps 2, 3, and 4 three times. Below, we describe some details of these iterations:

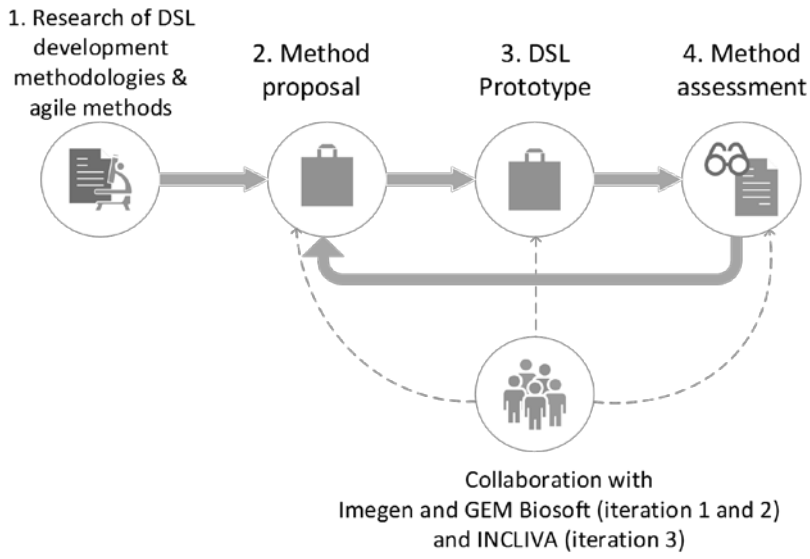


Figure 4.1 Approach to build the method and the DSL

- Iteration 1: We designed a draft of the stages Decision, Analysis, and Design. We applied these stages with the geneticists from Imegen and GEM Biosoft. As a result of this iteration, we detected a set of problems in the comprehension of some artefacts, the lack of guidelines for developers, and new ideas to improve the interaction with end-users. These results were presented in [81] and [82].
- Iteration 2: We applied several changes to solve the issues detected in the previous iteration and we designed a draft of the stages Implementation and Testing. Again, we applied these stages with the geneticists from Imegen and GEM Biosoft. As a result of this iteration, we detected that some artefacts were incomplete and some aspects of the proposal to involve end-users were still complex and contained ambiguous activities.
- Iteration 3: We applied several changes to solve the detected issues and we designed a draft of the stages Deployment and Maintenance. This time, we validated the method with geneticists from INCLIVA with an empirical experiment (explained in Chapter 8). As a result, we detected some issues and we proposed potential solutions to improve them. The experiment and the obtained results have been submitted for publication.

As a result of the third iteration, we obtained the first version of the method and, from this point, we applied the method to build the first version of the DSL for genetic analysis. The method and examples of this DSL are explained in Chapters 5, 6, and 7. The complete set of artefacts of the DSL for the three iterations are gathered in a technical report [83] and the artefacts of the last version of the DSL are shown in Annex B.

4.2 Combining model-driven and agile practices for DSL development

Involving end-users in any development process is a difficult task. The few interest and time of end-users to participate requires the involvement activities to be efficient, as less intrusive as possible, and engaging for end-users.

A novel trend for improving the efficiency of DSL development processes is applying model-driven development principles [16]. These principles propose to use models to formally describe concepts of the language domain and generating the corresponding DSL artefacts from them [22]. But applying a model-driven development approach and involving end-users simultaneously is a challenging task because end-users do not usually have the expertise necessary to participate in modelling tasks [2]. MDD approaches provide formalisms to design the conceptual models and to generate the software products from them, but they lack clear guidelines to teach end-users how to model these formalisms.

In contrast, agile methods [23] advocate the close collaboration of end-users and developers, focusing on requirements gathering, continuous testing, and project management. Although these methods lack guidelines to carry out different modelling activities such as domain modelling, business modelling, or behavior modelling, we believe that model-driven development approaches and agile methodologies can complement each other in the context of DSL development.

Our goal is to create a DSL development method that combines the ideas of the conceptual-model programming manifesto [84], which states that "the model is the code" and that conceptual-modelling languages must be executable, with the ideas of Agile Modeling [24], which advocate finding the balance between the completeness of the traditional modelling task and agile principles.

In the literature, we have found development approaches that combine MDD and agile principles, but for a different context or goal. For example, Rivero et al. [85] proposes an agile model-driven approach to develop web applications. This work configures the development process as iterative and uses mock-ups to involve end-users in the web interface design and also in the assessment of the web application derived from them. However, this approach is proposed for the development of a web application, which is different of developing a DSL.

Another example is the work of Visser [86], which proposes an approach to design DSLs by introducing agile practices such as carrying the DSL incrementally. This work adopts agile practices but it still does not address the issue of how to involve end-users in the process. This method was designed to develop a technical DSL to facilitate the creation of web applications (WebDSL), so the participation of end-users was not needed for the development of this DSL. Despite these differences, our work is inspired by these two proposals and some of their contributions were adopted.

Hence, with the aim to configure an efficient and user-friendly DSL development process, we propose **an agile model-driven method to involve end-users in DSL development**. In the context of this work, we follow the "method" definition presented in [87]:

> *"A method is an approach to perform a software/systems development project, based on a specific way of thinking, consisting, inter alia, of guidelines, rules and heuristics, structured systematically in terms of development activities, with corresponding development work products and developer roles (played by humans or automated tools)"*

This method describes the different conceptual models that must be created in each stage and describes the set of mechanisms that facilitate the participation of end-users in order to develop such models. We use the term "**mechanism**" to refer to the set of activities that are proposed to gather end-users' feedback about a certain aspect of a DSL. These mechanisms act as interface for end-users to participate in the creation and assessment of several DSL artefacts. From the feedback provided by means of a mechanism, the DSL developers specify the underlying conceptual models. Therefore, we establish a concise connection between agile practices and model-driven development.

In order to create the complete infrastructure of the DSL, the model-driven approach consists on creating different models that represent the different aspects of the DSL. In order to go through the different steps of the DSL life-cycle, we propose a set of "**model-based guidelines**" that explain to the developers how to transform the models of one step to the models of the next step. Thanks to this approach, the feedback that is gathered from the end-users and represented in a model is propagated through the different stages. This way, we ensure that the feedback provided by the end-users is represented in all the DSL artefacts, even in the ones that are too technical for end-users to participate (such as the implementation artefacts).

4.3 Overview of the method

As we have discussed in Section 1.1, a suitable DSL development method to address a complex domain must fulfil three requirements: 1) supporting the full DSL development lifecycle, so it can be applied in real practice (Requirement 1); 2) improving the DSL development efficiency (Requirement 2); and 3) enhancing end-user involvement in the stages in which they can collaborate (such as Analysis, Design, Testing and Maintenance (Requirement 3)).

In this chapter, we propose a method that combines model-driven development (MDD) and agile practices. This method: 1) clarifies and differentiates the stages of DSL development as well as the steps of each stage (to address Requirement 1); 2) concretizes the artefacts to be created in each step (to address Requirement 1 and 2); 3) provides guidelines (for developers) with the model-based transformations required to follow a MDD approach (to address Requirement 2); and 4) proposes a set of mechanisms to facilitate the participation of end-users in the creation of some of the DSL artefacts (to address Requirement 3). Additionally, we have included a set of guidelines, both to end-users and developers that explain how to apply each of the mechanisms in practice. These guidelines are explained in Annex A.

In order to specify each method stage, the specific steps of each stage, and the artefacts of the method that are required to follow a MDD approach (Requirements 1 and 2), we adopted the method stages proposed by Mernik et al. (explained in Chapter 3) (Figure 4.2): Decision, Analysis, Design, Implementation, Testing, Deployment, and Maintenance. Then, in order to

define our MDD approach, we followed the guidelines and suggestions of the works by Mernik et al., Strembeck et al., and Voelter et al., to propose how to support Analysis and Design stages using conceptual models and how to transform these models to a DSL implementation.

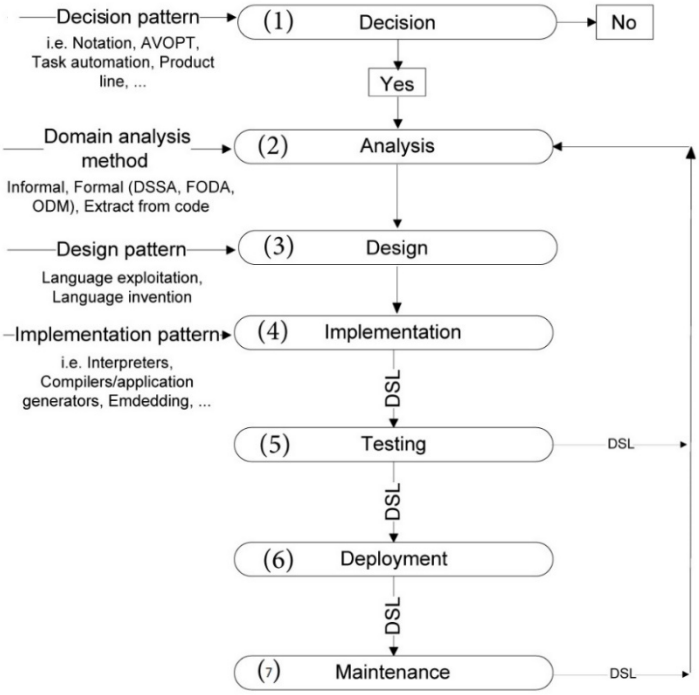


Figure 4.2 Overview of DSL development process and patterns by Mernik et al. (adapted from Ceh et al.)

Similarly, in order to design the suitable mechanisms to involve end-users (Requirement 3), we analyzed different agile methods (XP [88], Scrum [89], and Agile Modeling [24] to find the most suitable practices that fit into the context of DSL development and that also facilitate the comprehension for end-users. The reason for searching agile practices in several methods is justified by the need to find the most suitable practices specially designed to involve end-users.

Next, we analyze the agile practices that we found suitable for involving end-users, and we discuss how each of them could fit into the DSL development context:

- **Iteration planning or sprint (XP, Scrum, and Agile Modeling)**: The majority of agile methods manage the development in a set of iterations in which only a small set of features is implemented. This practice encourages the delivery of faster releases that can be evaluated by stakeholders, or end-users in our context. DSL development can benefit from this practice by checking if the concepts required by the end-users are included in incremental subsets of the DSL. Also, errors can be detected before developing the complete DSL infrastructure.
- **Incremental design (XP, Scrum, and Agile Modeling)**: Together with a development based on iterations, agile practices promote the incremental design of the different DSL artefacts. Many DSL development approaches design the entire language at once, and this design is only evaluated when it is complete. We believe that if the DSL developers and the end-users focus on a small set of language constructs, the DSL will be easy to assess and it will be more accurately reviewed.
- **User Stories (XP and Scrum)**: In order to manage requirements in agile methods, they are divided into user stories, which are brief descriptions related by the end-users about a demand that contributes to add value. The set of user stories provides a simplified view of the functional features to be developed. In the context of DSL development, the goal of the user stories is to describe language requirements instead of specific requirements of a software product. The structure of user stories does not differ from its traditional usage; however, they will be used to discover the language constructs and concepts to be introduced in the DSL.
- **Acceptance tests (XP) or usage scenarios (Scrum):** In order to check the fulfilment of requirements in agile methods, end-users briefly describe scenarios that must be accomplished by the software to be developed. Like user stories, in the context of DSL development, the difference lies in the fact that acceptance tests will check the fulfillment of language requirements instead of the requirements of a specific software product. The structure of acceptance tests does not change, although the input of an acceptance test will be a set of constructs instead of a software state.
- **Product backlog (Scrum):** In order to manage the list of requirements that should be addressed in each of the iterations, agile methods propose to make this list explicit. The application of this practice in the DSL context does not differ from its typical use. Since our goal is that end-

users participate thorough the entire DSL development process, the product backlog will be a useful tool to record all the new requirements that arise in any stage of process during the iteration. Also it is a guide to check what it is expected to be released in each of the iterations.

- **Architectural envisioning (Agile Modeling)**: This practice encourages the early identification of a viable technical strategy. Many times, DSL development is guided by domain concepts –especially when applying a model-driven approach—and its executable environment is decided in late stages. Because of that, as some authors have stated [86], delaying this decision complicates the possibilities to translate concepts to a working implementation. Hence, we believe that combining the model-driven approach with the envisioning of an architectural strategy can benefit the efficiency of the implementation of the DSL.
- **Test-driven development (XP):** This practice promotes the creation of tests as the main artefact that guides the coding process. In the context of DSL development, this agile practice is applied normally, although the tests drive the creation of the artefacts necessary to implement the complete DSL infrastructure, instead of its typical usage of guiding the development of a specific software product. The DSL infrastructure is the environment that allows the end-users to use the language to create and execute DSL specifications.
- **Definition of done (Scrum):** This practice promotes the definition of a classification criteria that classifies the features of the software design as "done" or "not done", establishing a common framework to understand the development state. In the context of DSL development, this classification criterion must take into account the completeness of the implementation of the DSL infrastructure.
- **Customer review or demonstration (Scrum)**: The main goal of this practice is to show the end-users the software release and get them to agree with it. In the context of DSL development, the demonstration should focus on showing the implementation of the language. Although it is important that the end-users are comfortable with the complete environment to use the language, the truly goal of DSL testing must be to achieve the assessment of the language constructs.

As a result of this analysis, we structured the process as an iterative short cycle, following two common practices adopted by agile methods: sprints (or iteration

planning) and incremental development. Specifically, we organized the DSL development process into iterations, addressing only one subset of all of the DSL requirements at a time, and demonstrating the result to the end-users at the end of each iteration. This practice allows the end-users to test a working subset of the DSL as soon as it is available, and the developers to detect errors and misunderstandings sooner, when they are easier to fix.

Besides these two agile practices, we adopted different agile practices (user stories, acceptance tests, usage scenarios, product backlog, architectural envisioning, test-driven development, definition of done, and customer demo) to design a set of mechanisms to facilitate end-user involvement in the stages Analysis, Design, Testing, and Maintenance. In total, we proposed five mechanisms to involve end-users:

- **Mechanism M1**: Two requirements templates based on user stories, acceptance tests, and usage scenarios for the Analysis stage.
- **Mechanism M2**: A syntax questionnaire based on usage scenarios for the Design stage.
- **Mechanism M3**: A semantic template based on user stories for the Design stage.
- **Mechanism M4**: A demonstration for the Testing stage.
- **Mechanism M5**: A testing questionnaire for the Testing stage.

Figure 4.3 and Figure 4.4 overview the method proposed: the iterative cycle, the stages, the steps, the artefacts involved in each stage, and the five mechanisms (M1-M5) for involving end-users. The iterative cycle only includes the following stages: Analysis, Design, Implementation, and Testing. The Decision stage is not included in this cycle because the decision to develop the DSL is only addressed once (at the beginning of the development process) and this decision is usually not revisited. The Deployment and Maintenance stages conform an alternative path after the Testing stage because it is only effective to deploy the DSL release to be used by end-users when there is a final stable version of the DSL.

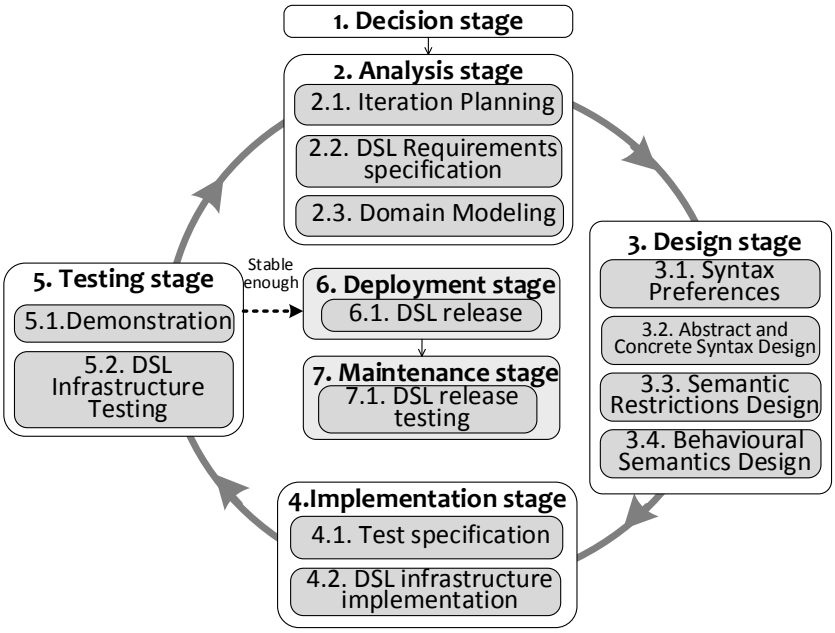


Figure 4.3 Stages and steps of the proposed method

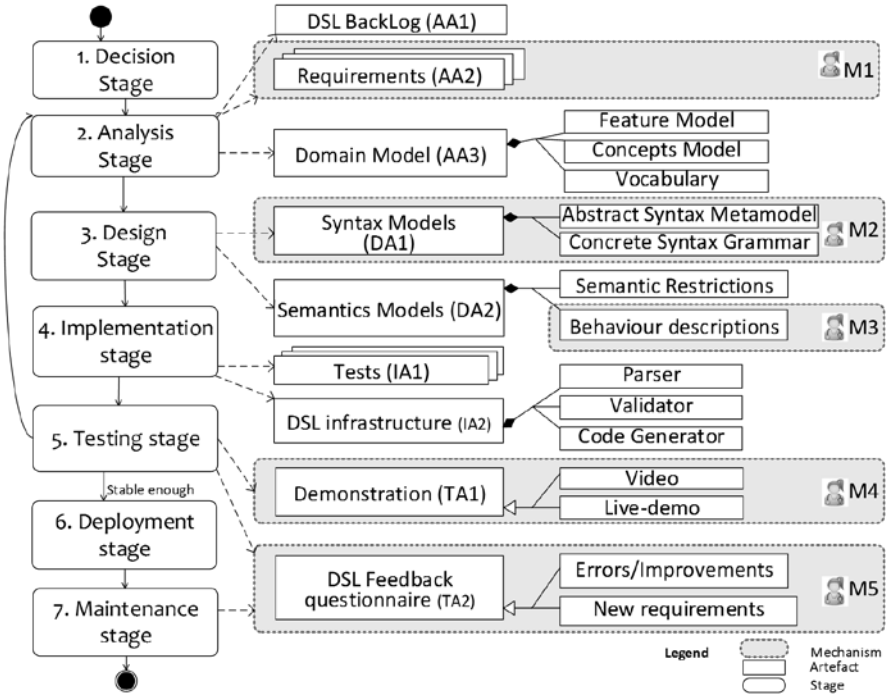


Figure 4.4 Artefacts of the proposed method and mechanisms for involving end-users

The development of the DSL starts with the **Decision stage** (see Chapter 5)**.** In this stage, the end-users and the developers discuss with each other whether or not to develop a DSL. In order to aid in this decision making, we adopt the decision patterns proposed by Mernik et al. (Notation, AVOPT, Task automation, Product Line, etc.).

Once the decision has been made, in the **Analysis stage** (see Chapter 5), the developers must understand the domain and make the end-users' knowledge explicit. In this stage, the developers plan the iteration (step 2.1), gather the DSL requirements from the end-users (step 2.2), and create a domain model that precisely describes the DSL context (step 2.3).

The three output artefacts of this analysis stage are: 1) a DSL backlog (artefact AA1), which organizes the requirements of the DSL into different iterations (adopted from Scrum); 2) the formalized end-users' requirements (artefact AA2); and 3) the domain model (artefact AA3), which is made up of a vocabulary (glossary of terms) for describing the domain terminology, a concepts model for describing the domain concepts, and a feature model for describing the commonalities and variabilities of the domain concepts (based on the Mernik et al. domain model definition and the Voelter et al. guidelines). To create this domain model (artefact AA3), the developers apply model-based transformations from the formalized requirements (artefact AA2).

In this Analysis stage, in order to formalize the end-users' requirements and use a notation that is understandable to them, we propose mechanism M1, a template based on user stories, acceptance tests, and scenarios from XP and Scrum.

Once the requirements and the domain have been elicited and formalized, in the **Design stage** (see Chapter 6), the developers create the artefacts that represent the DSL syntax and semantics. In this stage, the developers elicit the end-users' syntax preferences (step 3.1), design the abstract and concrete syntax according to those preferences (step 3.2), specify the semantic restrictions (step 3.3), and specify the behavioral semantics (step 3.4).

The two output artefacts of this Design stage are: 1) the syntax models (artefact DA1), and 2) the semantics models (artefact DA2). The syntax models are made up of an abstract syntax metamodel (which describes the internal structure of the DSL) and the concrete syntax grammar (which describes the

notation that will be exposed to end-users to use the DSL). The semantic models are made up of semantic restrictions (which describe the restrictions that must be ensured when creating domain instances with the DSL), and behavioral semantics (which describes the functional behavior that underlies each entity of the abstract syntax metamodel). To create the syntax and semantic models (artefacts DA1 and DA2), the developers apply model-based transformations from the domain model (artefact AA3).

In this Design stage, in order to involve end-users in the design tasks, we propose two mechanisms: *mechanism M2*, which is a questionnaire based on usage scenarios from Scrum that gathers end-users' preferences about the syntax; and *mechanism M3*, which is a template based on user stories from XP that facilitates their participation in the specification of the behavioral semantics.

Once the design artefacts have been created and refined by the end-users, in the **Implementation stage** (see Chapter 6), the developers must create the complete DSL infrastructure. This infrastructure aids end-users in the creation of DSL specifications and generates the corresponding artefacts (such as executable code) in the target implementation platform. In this stage, the developers design a set of tests that check for the correctness of the syntax and semantics implementation (step 4.1), and then the complete DSL infrastructure is implemented (step 4.2) by applying both model-driven development (MDD) practices and test-driven development (TDD) (from Scrum) using the design models (DA1 and DA2) and the tests from the previous step.

The two output artefacts of this Implementation stage are: 1) a set of tests (artefact IA1), which check for the correctness of syntax and semantics implementation (based on Scrum for applying TDD); and 2) a complete DSL infrastructure (artefact IA2), which is made up of a parser that understands the syntax, a validator that checks the correctness of the syntax and semantic restrictions, and a code generator that transforms DSL specifications into software with the associated behavior.

In this Implementation stage, we do not propose any mechanism that involves end-users because they require advanced skills in software engineering and MDD.

Once the DSL has been implemented, in the **Testing stage** (see Chapter 7)**,** the end-users assess the preliminary DSL infrastructure to check whether or not it fulfills their needs and preferences. In this stage, the developers conform a

functional DSL infrastructure and demonstrate it to the end-users (step 5.1), and then, the end-users try this DSL infrastructure and provide feedback about their experience using it (step 5.2).

The two output artefacts of this Testing stage are: 1) a DSL demonstration (artefact TA1), which can be a video or a live-demo (from Scrum); and 2) an iteration feedback report (artefact TA2), which contains errors that have been found in the DSL, improvements related to the current iteration requirements, or proposals of new requirements to be addressed in next iterations.

In this Testing stage, in order to facilitate the assessment of the DSL for end-users, we propose two mechanisms: mechanism M4, which is a demonstration based on usage scenarios from Scrum that facilitates the comprehension of the DSL infrastructure; and *mechanism M5*, which is a set of activities based on usage scenarios from Scrum and a testing questionnaire that facilitate the testing of the DSL infrastructure.

At this point in the method, if the DSL infrastructure is stable enough and it is considered to be valuable for end-users, in **the Deployment stage** (see Chapter 7), the current DSL infrastructure is released to the end-users so they can use it freely (step 6.1), and after the end-users have tried it for an extended period of time, in **the Maintenance stage** (see Chapter 7), they provide feedback about their experience like they did in the Testing stage, that is, by testing the DSL release (step 7.1).

4.4 Illustrative example

In order to apply the proposed method in a real environment, we selected the genetic analysis domain. Since we had access to geneticists from three industrial companies (Imegen, GEM BioSoft, and INCLIVA), we collaborated with them to develop a DSL for supporting genetic analysis.

The explanation of all the interactions and collaborations is out of the scope of this PhD thesis. However, in order to show how we applied the method in practice and the resulting DSL, we have selected a small example of a genetic analysis and we will use it to illustrate each method stage, step, artefact, and mechanism. This genetic analysis represents a simplified analysis that geneticists

carry out to research the Diabetes Mellitus Type 2 disease. The rest of the details of the DSL are gathered in a technical report [83].

The complexity of the genetic analysis domain, its constant evolution, and the interest of geneticists to try different experiments to find new discoveries, has led to a situation in which software products for genetic analysis cannot evolve on time to fully satisfy all the geneticists' needs.

The consequence of this situation is geneticists dealing with programming and technological issues in their daily work in order to perform their analysis, whether they like it or not. The problem is that instead of focusing their efforts in genetics research, geneticists spend a lot of time acquiring technical knowledge, programming their own pipelines, and trying to solve programmatic issues. Another problem is that without the suitable software engineering knowledge, the quality of their programs is clearly affected because, among other issues, data is manipulated using programming scripts, there are no well-defined data abstractions, and pipelines are hard-coded.

As a solution, geneticists need to be provided with an infrastructure with a higher level of abstraction to customize their pipelines and manage the underlying technological issues. Developing a DSL seems to be an appropriate solution for this domain. For this reason, we proposed to develop a DSL for the creation of pipelines for processing DNA data.

But developing a DSL for this complex domain is a task that requires the involvement of geneticists (who know all the details of these pipelines) because the domain concepts are very difficult to understand for developers. Hence, in order to develop this DSL, the participation of geneticists is essential.

As we explained in Section 4.1, the current version of the proposed method was designed within three iterations. In each of these iterations, in order to validate the method, we applied it together with geneticists to create and evolve the genetic DSL. The three iterations of this DSL were:

- Iteration 1: We applied the first method draft to build a DSL that supported a set of requirements related to the HGVS notation (a standard notation used in the genetic analyses domain). Geneticists from Imegen and GEM BioSoft collaborated in this iteration.
- Iteration 2: We applied the second method draft to evolve the DSL to support a basic usage scenario related with the diagnosis of the Breast

Cancer Disease. Geneticists from Imegen and GEM BioSoft collaborated in this iteration.

- Iteration 3: We applied the first stable version of the method to evolve the DSL to support several usage scenarios related with the research of Diabetes Mellitus Type 2. Geneticists from INCLIVA collaborated in this iteration.

In each iteration, we evolved the existing DSL artefacts of the previous iteration to support the new requirements. As a result, the current version aggregates the artefacts of the three iterations. The final version of this DSL (Iteration 3) addresses the requirements of genetic analysis domain that is shown in Table 4.1.

Table 4.1 Set of requirements supported by the third *version* of the DSL

Related to	**DSL requirement**
Input patient data	Read variations from a VCF file
	Read genotypes from a VCF file
Data Analysis	Annotate Variations with: hgvs, gene, rsId, transcript names, predicted effect (SIFT and POLYPHEN[7]), and sample minor allele frequency
	Search Variations by: HGVS DNA, HGVS Coding and HGVS Protein
	Filter Variations by: gene and predicted effect (SIFT and POLYPHEN)
	Prioritize Variations by: predicted effect score (SIFT and POLYPHEN), and sample minor allele frequency
Reporting	Report variations general properties
	Report variations annotations: gene, rsId, hgvs, transcript, predicted effect, and sample minor allele frequency

In order to facilitate the comprehension of the illustrative example, next subsections explain: 1) the workflow that describes the genetic analysis; 2) the pipeline created by the INCLIVA geneticists to implement that genetic analysis; and 3) the specification of that genetic analysis using the DSL.

4.4.1 The default workflow

The goal of the genetic analysis example that has been selected is to read a list of genetic variations, retrieve information about them, select the ones that are interesting for the study of the disease, and report the selected ones in a friendly way. It is worth to mention that geneticists will refer as "annotate variation" to the action of retrieving additional information about a variation and as "filter

[7] SIFT and POLYPHEN are two well-known algorithms in the genetic analysis domain community that predict the seriousness of a variation in an individual.

variations" to the action of selecting a subset of variations from a list according to a specific criterion.

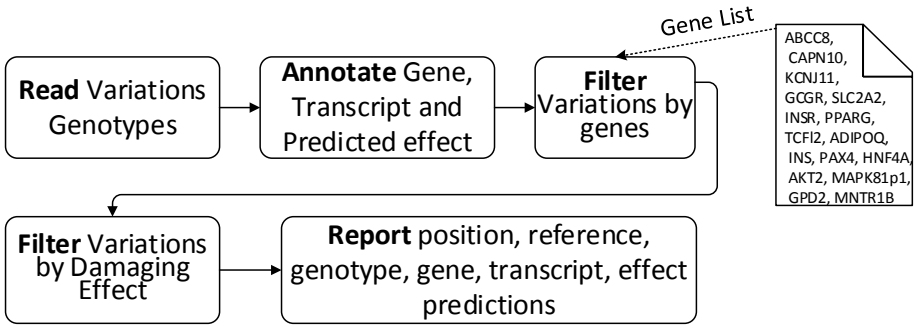


Figure 4.5 Steps to analyze Diabetes Mellitus Type 2

Figure 4.5 shows the workflow that describes the genetic analysis selected. The workflow starts reading the genetic data of the samples, being a sample one of the patients that have been sequenced for the study. The first goal of this step is to read the list of genetic variations, their position and the value ("A", "C", "T", or "G") of the reference sequence in that position. The second goal is to obtain the value ("A", "C", "T", or "G") of each individual sample. This last value is what geneticists named as the genotype of a sample. As an example, Table 4.2 shows three variations, their corresponding position, and the reference sequence values, but it also shows for each variation (variation 1, 2 and 3), the genotypes of each sample (sample 1, 2, and 3).

Table 4.2 Example to illustrate Variation Genotypes

	Position	**Reference**	**Sample 1**	**Sample 2**	**Sample 3**
Variation 1	100	A	A	A	T
Variation 2	1000	A	A	C	C
Variation 3	1020	C	G	G	G

After reading the variation genotypes, variations are annotated with different information. Continuing with the example of Table 4.2, geneticists want to annotate each variation with the gene where the variation is located, the transcript that the variation is affecting, and the effect that is likely to happen due to this variation. Table 4.3 shows the different variation annotations. For example, the variation 1 is located in the gene *BRCA1*, is affecting to the transcript *NM_0001.1*, and its predicted effect is *benign*.

Table 4.3 Example to illustrate annotations

	Position	Reference	Sample 1	Sample 2	Sample 3	Gene	Transcript	P. Effect
Variation 1	100	A	A	A	T	BRCA1	NM_0001.1	Benign
Variation 2	1000	A	A	C	C	ADIPOQ	NM_0002.1	Unknown
Variation 3	1020	C	G	G	G	ADIPOQ	NM_0023.2	Damaging

Once all the variations are annotated with the information required by geneticists, they want to apply a set of filters so they can focus only on the variations that may be related with the disease they are researching. Continuing with the example of Table 4.3, geneticists want to focus only on the genes related with Diabetes Mellitus Type 2 (*ABCC8, CAPN10,KCNJ11, GCGR, SLC2A2, HNF4A, INS, INSR, PPARG, TCFl2, ADIPOQ, AKT2, PAX4, MAPK81p1, GPD2, and MNTR1B*), focus only on the variations whose predicted effect is *damaging*, and discard the rest of the variations. Table 4.4 shows the result of applying the gene and the effect filters described. Only the variation 3, whose gene is *ADIPOQ* and predicted effect is *damaging* passes both filters.

Table 4.4 Example to illustrate filters

	Position	Reference	Sample1	Sample2	Sample3	Gene	Transcript	P. Effect
Variation 3	100	C	G	G	G	ADIPOQ	NM_0023.2	Damaging

In the last step of the workflow, geneticists want to create a report so they can visualize only the information they are interested in. This report gathers the variations filtered together with the information chosen by geneticists. Following with the example of Table 4.4, the final report gathers the variation 3 and shows the fields position, reference, genotypes, gene, transcript, and predicted effect.

Once geneticists know the analysis they want to perform, they must look for the software tools that give support to each step. Since sequencing machines have improved their sequence throughput, performing the pipeline using a manual approach is no longer feasible.

4.4.2 The tool implementation

Geneticists from INCLIVA describe their genetic analyses by means of pipelines that interconnect a set of technological tools transferring datasets (usually flat text files) among them. In order to carry out their analysis, they specify their pipeline (for instance using a scripting language), they provide a

dataset input to the first tool, and they expect an output (or several outputs) that contains the analysis result.

Figure 4.6 shows the pipeline created by geneticists from INCLIVA to implement the illustrative example. This pipeline integrates the tools Annovar [38] to retrieve genetic information, VEP (Variant Effect Predictor) [40] also to retrieve genetic information and to select the corresponding variations, and a custom reporter to create a HTML-based report with the list of selected variations.

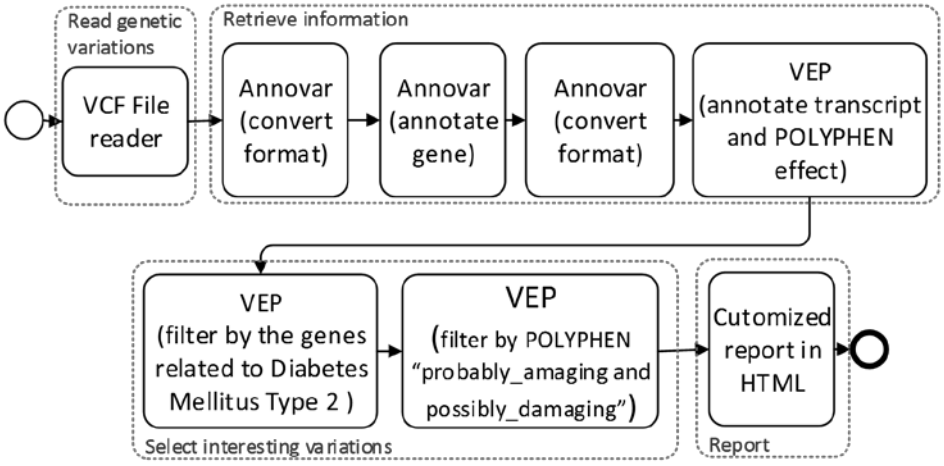


Figure 4.6 Software tools to analyze Diabetes Mellitus Type 2

In order to implement this pipeline the geneticists from INCLIVA program a Unix script (Figure 4.7). This script configures the parameters of the tools Annovar, VEP and the HTML reporter, so they annotate the suitable annotations, apply the suitable filters, and create the corresponding report. For geneticists, the complexity of implementing this pipeline lies in selecting the suitable tools for each goal, configuring the tools with the right parameters, and managing the interoperation among them.

```
/*Annotate gene and transcript with Annovar*/
perl convert2annovar.pl -format vcf4 file.vcf > file.avinput
annotate_variation.pl -buildver hg19 –geneanno file.avinput humandb/
perl convert2vcf.pl -format vcf4 file.avinput> annotatedGeneTranscript.vcf

/*Annotate gene and transcript with Annovar*/
perl variant_effect_predictor —cache –i annotatedGenefile.vcf—polyphen b –o
annotatedAll.vcf

/*Filter by gene*/
$filterFieldGene= “EFF[*].GENE='ABCC8 ' |EFF[*].GENE='CAPN10 '
|EFF[*].GENE='KCNJ11' |EFF[*].GENE='GCGR' |EFF[*].GENE=' SLC2A2'
|EFF[*].GENE='HNF4A' |EFF[*].GENE='INS ' |EFF[*].GENE='INSR'
|EFF[*].GENE='PPARG' |EFF[*].GENE='TCFl2' |EFF[*].GENE='ADIPOQ’
|EFF[*].GENE='AKT2' |EFF[*].GENE='PAX4' |EFF[*].GENE='MAPK81p1'
|EFF[*].GENE='GPD2' |EFF[*].GENE='MNTR1B' |”
perl filter_vep.pl -i $input -o $output --force_overwrite --filter "$filterFieldGene"

/*Filter by effect*/
$filterFieldEffect= “\“Polyphen is possibly_damaging\” or \“Polyphen is
probably_damaging\””
perl filter_vep.pl -i $input -o $output --force_overwrite --filter "$filterFieldGene"

/*Report variations and fields*/
java –jar reporter.jar annotatedAll.vcf --gene –transcritpt --polyphen
```

Figure 4.7 Scripting language to analyze Diabetes Mellitus Type 2

4.4.3 The DSL specification

As a solution to solve the complexity of this pipeline, Figure 4.8 shows how the DSL is used to specify the same example. The first line describes the goal of the analysis, and the next lines describe the input of the pipeline, the information that must be retrieved, the criteria to select the variations, and the configuration of the final variation report.

```
Diagnose Diabetes Mellitus Type 2 (Analysis 1)
Read Variations genotypes from VCF file Patient1.vcf
Annotate Variations with gene, transcripts, polyphen
Filter Variations by genes {ABCC8, CAPN10, KCNJ11, … , GPD2,
MNTR1B}
Filter Variations by predicted effect polyphen damaging
Report Variations with gene, predicted_effect
```

Figure 4.8 The DSL to analyze Diabetes Mellitus Type 2

This specification shows how the DSL provides geneticists with a higher level of abstraction for creating and customizing their pipelines. The main difference between this specification (Figure 4.8) and the script (Figure 4.7) is the use of genetic analysis concepts instead of technological concepts. This example hides

the specific software tools that implement the genetic analysis pipeline, such as Annovar or VEP, avoids the need to learn the specific details about how each of these tools are configured, such as the syntax of VEP to filter a gene ("Eff[*].gene=BRCA1"), and avoids the need to manage the interoperation among tools such as the use of convert2annovar to convert the data from the format VCF to the avinput format and backwards.

4.5 Conclusion

In this chapter, we have explained how we collaborated with end-users (geneticists from three organizations) to build the method proposed in this PhD. In order to avoid reinventing the wheel, we have analyzed existing approaches for DSL development and agile practices focused on involving end-users. As a result, we have proposed an agile model-driven method to involve end-users in DSL development.

In order to apply this method in a real environment, we have collaborated with geneticists to develop a prototype of a DSL for genetic analysis. In this chapter, we have introduced a small example of this domain. This example is used in next chapters to illustrate the different stages, steps, artefacts, and mechanisms of the method.

Chapters 5, 6, and 7 will explain the Decision, Analysis, Design, Implementation, Testing, Deployment, and Maintenance stages in detail, and in each of this chapters, after explaining the corresponding stage, we will use the illustrative example to show how the stage can be applied in practice.

5. Understanding the domain: The Decision and Analysis stages

The first thing to do in order to develop a DSL for a complex domain is approaching developers and domain experts (a.k.a end-users) so they could assess together whether developing a DSL for this domain is worth. Otherwise, the effort invested in the development could be a waste of resources. In our method, this decision is addressed in the Decision stage.

Once it has been decided to go on with the development, the developers must acquire the necessary domain knowledge that allows them to understand the needs and preferences of the end-users. The goal is to ensure that the developers are able to understand all the details that must be included into the DSL. In our method, this knowledge acquisition is addressed in the Analysis stage.

In this chapter, we explain the Decision and Analysis stages and how we applied each of them for developing of a DSL for the genetic analysis domain. We created several versions of the method and the DSL; however, in order to simplify the explanation of the method, we focus only on the method version that corresponds to the last iteration. Similarly, in order to simplify the explanation of

the application of the method to build the DSL, we only provide fragments of the DSL in regards to the illustrative example presented in Chapter 4.

In summary, we start describing the Decision stage and how we applied this stage in the real use case. Then, we proceed equally to describe the Analysis stage.

5.1 The Decision stage

The goal of the Decision stage is to analyze the domain of the DSL, identify the need of a DSL for the domain, and justify that the efforts that will be invested in its creation are worth.

In order to make this decision, the end-users and the developers discuss the end-users' requirements and inspect existing implementations and documentation about the domain. In order to facilitate this decision making, Mernik et al. [10] identifies a set of patterns that can be used to justify this decision. The patterns are the following:

- **Notation**: The context of this pattern is a domain in which there is an existing notation or a software API that is being used by domain experts but it does not completely satisfy them. This pattern is applied when there is a need to transform a visual notation to a textual notation or to improve an existing software API with friendlier abstractions.
- **AVOPT**: The context of this pattern is a domain in which domain programs are written with a general-purpose language and domain experts need to Analyze, **V**erify, **O**ptimize, **P**arallelize, or **T**ransform those programs. This pattern is applied when domain experts found very difficult to perform these operations over the existing domain programs and would rather have friendlier abstractions to do it.
- **Task automation:** The context of this pattern is a domain in which domain experts spend a lot of time programming to accomplish the same tasks. This pattern is applied when domain experts would like to encapsulate these tasks and use them easily any time they need them.
- **Product line**: The context of this pattern is a domain whose applications share a common architecture and are built by selecting or unselecting a common set of basic elements. This pattern represents the need to

support the specification of each member of this family by selecting and configuring those elements easily.

- **Data structure representation**: The context of this pattern is a domain in which domain experts rely on well-known predefined data structures to build domain programs. This pattern represents the need to facilitate the description and maintenance of these data structures.
- **Data structure traversal:** The context of this pattern is a domain in which domain programs underlie complex structures hidden under general programming code. This pattern represents the need to facilitate the description of those programs taking into account this underlying structure.
- **System front-end:** This context of this pattern is a domain in which domain experts need to configure different aspects of their systems. This pattern represents the need to provide a front-end for the end-users so they can customize this configuration and avoid dealing with configuration files directly.
- **Interaction**: The context of this pattern is a domain in which domain experts interact with the software by means of menus or text introduction. This pattern represents the need to make the interaction among the end-users and the system programmable.
- **GUI construction**: This pattern represents the need of having a DSL to ease the construction and configuration of a graphical interface for the end-users.

5.1.1 The decision of developing a DSL for supporting genetic analysis

In this stage, we had to decide whether or not to develop the DSL for supporting genetic analysis. Together with the geneticists we identified which of the aforementioned decision patterns justified the decision to proceed with this DSL:

- **Notation**: In the genetic analysis domain there is not an existing notation that is being used by domain experts to describe genetic analysis. Some geneticists use pipeline development environments but, as we described in Chapter 2, the geneticist's problems are not related with the notation. However, there is an existing "software API" that does not satisfy them completely. When geneticists want to perform a genetic analysis, instead

of choosing which genetic task they want to perform, they must discern which specific software tool to use from a wide array of choices and configure the technological details. For example, instead of choosing "protein alignment" they must execute a command line tool named "blastp" (blast algorithm for proteins). Also, this command tool requires a database object, which is a structure that gathers a set of DNA reference sequences and its identifiers. In this example, geneticists should decide if this command line tool is the most suitable to execute the protein alignment they want to perform and configure all the specific parameters of this tool. A DSL could improve this "software API" and hide these technological details that are outside their domain scope and abstract the different decisions to make by means of genetic analysis domain concepts.

- **AVOPT**: Since geneticists perform analysis over data coming from real patients, they need to verify that all the parameters of the genetic analysis are correct. Also, due to the improvement of genetic sequencing machines, bigger amounts of genetic data are available, which means that geneticists also need to optimize and parallelize their pipelines. Therefore, a DSL could have several constructs for performing those operations over genetic analysis pipelines.
- **Task automation**: While performing genetic analyses, geneticists must deal with some tasks that could be automated. One example is the interoperability among genetic analysis tools. Geneticists transform the data manually or use programming utilities to perform format transformations. A DSL could hide the details of those tasks and allow data exchange among different tools transparently for them.
- **Product line**: Although different genetic analyses are done for different genetic diseases, they have a lot of commonalities. Each specific genetic analysis can be seen as a set of analysis elements that are selected and configured depending on the disease being analyzed. These commonalities could be encapsulated by a set of DSL constructs and geneticists will select and configure them in order to specify each analysis.
- **Data structures**: In the genetic analysis domain, geneticists do not describe their genetic analysis by means of pre-defined data structures. Therefore, this decision pattern does not apply to this domain.

- **Data structure traversals**: In the genetic analysis domain, genetic analysis programs do not underlie complex structures. Therefore, this decision pattern does not apply to this domain.
- **System front-end**: The genetic analysis domain is highly coupled to specific software tools. Some geneticists perceive a genetic analysis as a set of tools whose parameters must be configured. Therefore, the DSL should also provide constructs to configure tool parameters for those geneticists that want to describe their analysis using this approach.
- **Interaction:** In order to perform a genetic analysis, geneticists do not need to introduce complicated input or program the introduction of this data. They perform those analysis by introducing text files that contain genetic data. Therefore, this decision pattern does not apply to this domain.
- **GUI construction**: The genetic analysis field is an evolving field whose domain applications easily become obsolete. For this reason, applications do not focus on the user interface and geneticists are getting used to prototypical interfaces. At the moment, geneticists are more focused on the functionality of their genetic analysis and do not care about the structure of their user interfaces. Therefore, this decision pattern does not apply (yet) to this domain.

The analysis of these patterns confirm that geneticists have a set of problems that could be solved by using a DSL. For this reason, we conclude that there are evidences to think that developing a DSL will be worth the effort.

5.2 The Analysis stage

The goal of the Analysis stage is to analyze the domain of the DSL and the requirements of the end-users. According to Mernik et al., the output of this stage is a domain model that characterizes the domain.

This stage is divided in three steps: 2.1) Iteration planning; 2.2) requirements specification; and 2.3) domain modeling. Table 5.1 shows the step number and description, the artefacts created in each step, the model-based transformations guidelines for developers, and the mechanisms proposed for gathering end-users' input. All these elements were introduced in Section 4.2.

Table 5.1 Overview of the Analysis stage

Step	Step Description	Artefact	Model-based transformations guidelines	Mechanism for gathering end-user input
2.1	Iteration planning	DSL backlog (AA1)	-	Interview to identify priorities
2.2	Requirements specification	Requirements models (AA2)	-	Mechanism M1: Templates based on user stories (US), acceptance tests (AT) and usage scenarios (USC)
2.3	Domain modeling	Feature model, concepts model, and vocabulary (AA3)	Transformation guidelines from requirements model	Interview to define vocabulary terms

5.2.1 Iteration planning

In the step **Iteration planning**, the end-users provide an overview of their requirements, add them to the product backlog (artefact AA1) and discuss their priorities with the developers. The developers' goal during this task is to ensure that the requirements added to the product backlog are well distributed in different iterations, and the highest priority ones are addressed in the current iteration. This planning is done in collaboration between the end-users and the developers, since the end-users are the ones who know their priorities, and the developers are the ones that can evaluate whether the distribution of requirements among the iterations is feasible.

5.2.2 Requirements specification

Next, in the step **Requirements specification,** the end-users and the developers discuss about the requirements of the current iteration. The end-users explain their requirements, the developers formalize them explicitly, and the end-users review them again to check for their correctness.

It is worth to mention that in the context of DSL development, we have identified two types of requirements: *end-user requirements*, which are the requirements that the end-users expect from a specific software application, and *DSL requirements*, which are the requirements that the end-users expect from a language that will allow them to create and customize their own software applications. Since the end-users do not usually have language development expertise, they are only responsible for describing *end-user requirements*, while the developers are responsible for obtaining the *DSL requirements* from the corresponding *end-user requirements*.

In order to gather end-user input and to formalize both types of requirements (artefact AA2), we propose **mechanism M1.** This is composed of two templates: the *user story template*, which is based on user stories and acceptance tests from XP, and the *usage scenario template,* which is based on scenarios from Scrum. Each of these templates are proposed either to describe *end-user requirements* or *DSL requirements*. They contain the same fields for both purposes, but depending on the type of requirement being described, the content of each field can be slightly different.

The *user story template* describes the details of each of the individual requirement expected by the end-users. This template gathers the following fields*:*

- User Story: For end-user requirements, this field describes how a user with a specific *role* executes an *action* to achieve a *goal*. In order to facilitate the understandability of user stories, they are described in natural language using a predefined structure (Figure 5.1). For DSL requirements, the *role* is the user of the language, the *action* is the language construct used, and the *goal*, the behavior associated with that language construct. Additionally, we complement this description with the attribute *mandatory* to indicate whether the language construct is mandatory for any DSL specification.

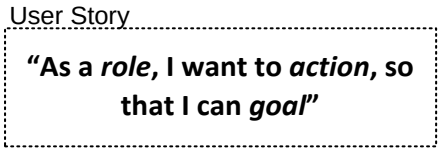


Figure 5.1 Predefined structure of user stories

- A set of acceptance tests: An acceptance test checks whether a user story works with a real example. For end-user requirements, it describes how a user with a *role* gives an *input*, executes an *action*, and expects a specific *response*. A user story should have a set of acceptance tests that is representative of the different situations that can occur in relation to the user story. Like user stories, in order to facilitate the understandability of acceptance tests for end-users, they are described in natural language using a predefined structure (Figure 5.2). For DSL requirements, the *role* is the language user, the *input* is a set of language constructs, the *action* is a language construct, and the *response* is the expected behavior of this language construct.

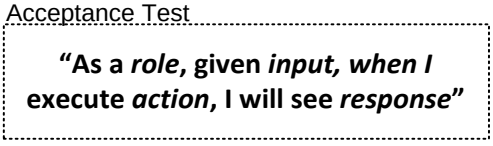


Figure 5.2 Predefined structure of acceptance tests

- A set of dependencies: For end-user requirements, a dependency is the interrelationship that exist between two user stories. This concept is not adopted from any agile methodology. For DSL requirements, a dependency is used to describe preconditions among DSL constructs. If the *precondition* is not satisfied, when the DSL users execute the *action*, an *error message* should be provided.

As an example, Table 5.2 and Table 5.3 show the user story templates to describe the requirement *"extracting records that fulfill a specific criterion"* related to the SQL construct *WHERE condition*. Table 5.2 describes the end-user requirement of a user who wants to see only a subset of records. It contains an acceptance test (AT1) that describes the response that is expected when a specific input data set is provided. Additionally, this template describes the dependency (DP1) of having a set of records already selected.

Table 5.2 The user story template to describe an end-user requirement

User Story "Select fields by name"			
Description	As a user, I want to extract records that fulfill a specific criterion so that only those records are shown in the result		
Role	**Mandatory**	**Action**	**Goal**
User	No	Extract records that fulfill a specific criterion	To show a subset of records
Acceptance Test AT1			
Description	As a user, given the dataset ((Id=0, Name="Maria", Department="CAP"), (Id=1, Name="Joseph", Department="DSIC")), when I extract records whose department name is DSIC, I will see (1, Joseph, DSIC)		
Role	**Input**	**Action**	**Response**
User	(Id=0, Name="Maria", Department="CAP") (Id=1, Name="Joseph", Department="DSIC")	Extract records whose department name is DSIC	(1, Joseph, DSIC)
Dependency DP1			
Description	If no records have been selected from a table, when I extract records that fulfill a specific criterion, I will see the error "Data needs to be selected before extracting records"		
Precondition	Action	Error Message	
Select records from a table	Extract records that fulfill a specific criterion	"Data needs to be selected before extracting records"	

On the contrary, Table 5.3 describes the DSL requirement expected by a user who wants to include in their software application the extraction of records that fulfill a specific criterion. It contains two acceptance tests. One (AT1) describes the response that is expected when a specific set of language constructs are provided and the other (AT2) describes the error that should arise when the DSL construct is not correctly used. Additionally, it contains a dependency that describes the prerequisite of the user story "*select data from a table*", which corresponds to the prerequisite of the SQL language construct *SELECT data FROM table*.

Table 5.3 The user story template to describe a language requirement

User Story Extract records by criterion			
Description	As DSL user, I want to write extract and a specific criterion so that the records that fulfill that criterion are extracted		
Role	**Mandatory**	**Action**	**Goal**
DSL user	No	Write Extract and a specific criterion	Extract records that fulfill that criterion
Acceptance Test AT1			
Description	As a DSL user, given the constructs select from the table Department, when I write extract and department name DSIC, I will see the source code that extracts the records whose department name is DSIC.		
Role	**Input**	**Action**	**Response**
DSL User	Select from table Department	Write Extract and department name DSIC	Source code that extracts records whose department name is DSIC
Acceptance Test AT2			
Description	As a DSL user, given the constructs select from the table Department, when I write extract and no condition, I will see an error saying "You should specify the criteria to extract"		
Role	**Input**	**Action**	**Response**
DSL User	Select from table Department	Write Extract	"You should specify the criteria to extract"
Dependency DP1			
Description	If select records has not been written, when I write extract records that fulfill a specific criterion, I will see the error "You need selecting records before extracting records"		
Precondition	**Action**	**Error Message**	
Write Select records from a table	Write Extract and criterion	"You need selecting records before extracting records"	

The *usage scenario template* describes a real example of the domain that includes several user stories working at a time. As an example, Table 5.4 and Table 5.5 show the usage scenario template to describe the requirement that integrates the user stories "*Select data from a table*" and "*Extract records by criterion*". Table 5.4 describes the end-user requirement of a user who wants to obtain all the departments whose name is DSIC from a specific database and Table 5.5 shows the DSL requirement of a user who wants to create a software application that obtains all the departments whose name is DSIC.

Table 5.4 Usage scenario template to describe an end-user requirement

Usage Scenario	**"Obtain all the departments with name DSIC"**
Description	"From the database UPV, select all the fields from the table Department whose name of the department is DSIC"

Table 5.5 Usage scenario template to describe a DSL requirement

Usage Scenario	**"Tool for DSIC Departments"**
Description	"Write Select, all the fields, and Department. Write extract and Department Name as DSIC"

In summary, as a result of this stage, the following requirement models are obtained: user story templates and usage scenario templates that describe end-user's requirements and DSL requirements.

5.2.3 Domain modeling

Next, in the step **Domain modeling**, developers create the domain model (artefact AA3) that represents the requirements of the current iteration. As we mentioned in Chapter 4, following the guidelines from Mernik et al. [10] and Voelter et al. [22], we propose to specify this domain model by using (Figure 5.3): 1) a feature model to describe the commonalities and variabilities of the domain; 2) a concepts model (which can be a UML Class diagram) to describe the concepts of the domain; and 3) a vocabulary of terms to describe the domain terminology.

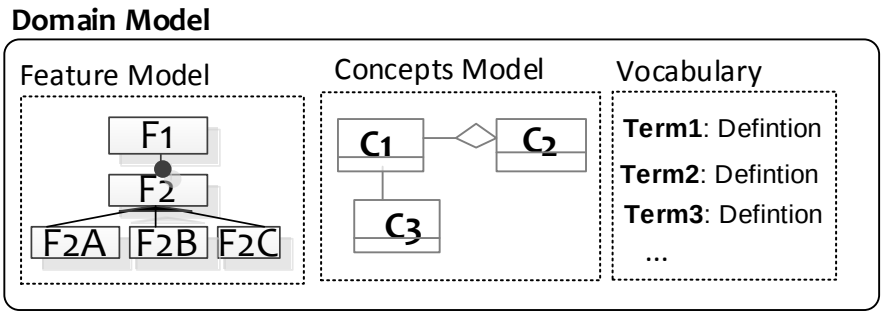


Figure 5.3 Specific artefacts proposed to describe the domain model

The domain model must describe as well the relationships that exist among these three elements. Sometimes, a feature needs a set of concepts in order to describe its behavior. Therefore, in order to represent this need we propose to specify relationships between the feature model and the concepts model. Specifically, we propose to specify relationships among features of the feature

model and entities of the conceptual model (Figure 5.4). On the contrary, the relationships between the concepts model and the vocabulary are not made explicit because the vocabulary uses the same terms that are written in the concept entities.

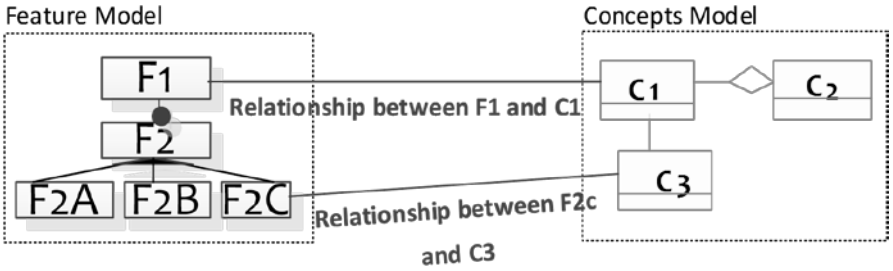


Figure 5.4 Relationships between the feature model and the concepts model

In order to create all these elements of the domain model, the developers extract the knowledge from the user story templates that were described in the previous step (the ones that describe *DSL requirements)*. Since end-users do not usually have modeling experience, they only contribute by answering the questions from the developers and by defining the terms of the vocabulary. Next, we describe the guidelines for developers to build these models.

For the feature model, the developers create the features and establish the hierarchy and dependencies between them (Figure 5.5):

- First, a root feature represents all the possible instances of the domain. Its name must be representative of the domain. In Figure 5.5, the root feature is named Domain.
- As a general rule, one feature is created per each user story. The feature name is a summary of the action of the user story. In Figure 5.5, this relationship is illustrated by the arrows 1 and 4.
- When a feature is added to the feature model, it is initially added as a child of the root feature. If the user story is mandatory or optional, the feature is also mandatory or optional, respectively. In Figure 5.5, this fact is illustrated by the arrows 2 and 3.
- Hierarchies between user stories must be identified, normally by observing commonalities and variabilities in the actions. While building the feature model, the same hierarchies are translated to the features of the feature model. For instance, if two user stories describe two similar actions it is likely that both are options of the same general action. This

situation is represented with a (parent) feature representing the common action and two child features of this parent feature.

- Finally, each dependency described in a user story is translated as a dependency between the corresponding features. In Figure 5.5, this fact is illustrated by the arrow 5.

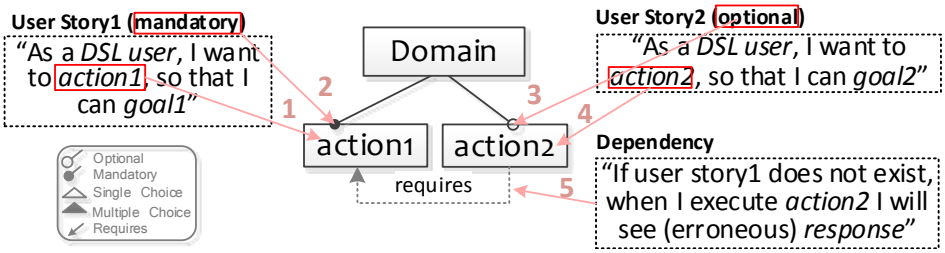


Figure 5.5 Example that illustrates how to create the feature model

For the concepts model, the developers create the entities of the domain, their attributes, and the relationships between concepts (Figure 5.6):

- Entities and their attributes are created from each of the terms written in the action and the goal of the user stories. In order to decide if a term should be a new entity or a new attribute, the developers ask the end-users to explain each term.
- The relationships between entities are also identified by the developers when the end-users explain each term.

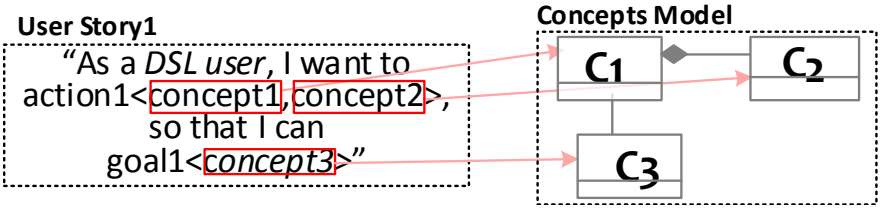


Figure 5.6 Example that illustrates how to create the conceptual model

For the vocabulary, the developers use natural language to define all the entities of the concepts model. This vocabulary may be built simultaneously with the conceptual model.

For the relationships between the feature model and the conceptual model, the developers create the relationships between features of the feature model and entities of the conceptual model (Figure 5.7):

- For each user story, a relationship is created when the action field contains a domain concept.
- As a general rule, if the action of the user story includes several concepts it is created a relationship per each concept. However, depending on the hierarchies of the concepts, sometimes is enough to establish a relationship with the parent concept that contains several concepts.

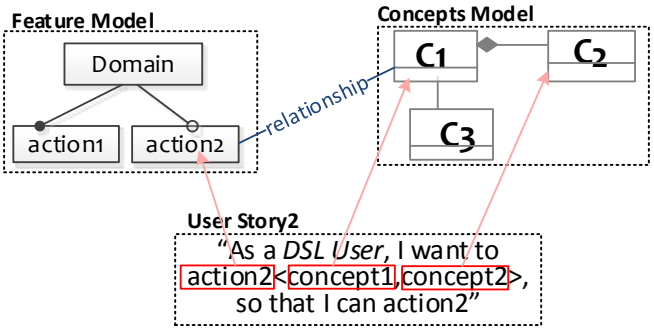


Figure 5.7 Example that illustrates how to relate the feature model and the concepts model

5.2.4 The analysis of the genetic analysis domain

In this stage, we collaborated with geneticists to obtain the domain model that represents the genetic analysis domain.

Regarding the **iteration planning** (Section 5.2.1), Table 5.6 shows a partial DSL backlog that contains the requirements related to the illustrative example (explained in Chapter 4): the requirements already addressed in the previous iterations (Iteration 1 and 2) and the ones to be addressed in the current iteration (Iteration 3).

Table 5.6 Partial DSL backlog of Iteration 3

Classification	Requirements
Previous Iterations (Done)	Annotate Variations with Gene
	Filter Variations by Gene
	Report Variations' Properties
	Report Variations' Gene
Current Iteration (To do)	Read Genotypes of several samples from a VCF File
	Annotate Variations with Transcripts Names
	Annotate Variations with POLYPHEN predicted effect
	Filter Variations by POLYPHEN predicted effect
	Report Variation's POLYPHEN predicted effect

Regarding the **requirements specification** (Section 5.2.2), we formalize the geneticists requirements (end-user requirements) and the requirements of the genetic DSL (DSL requirements) by fulfilling the user story templates and the usage scenario templates (mechanism M1).

Table 5.7 and Table 5.8 show the user story templates that describe the requirement "*Filter Variations by POLYPHEN predicted effect*". Table 5.7 describes the corresponding *end-user requirement*, which details the user story, two acceptance tests, and one dependency. In the first acceptance test, one variation passes the filter but in the second acceptance test, none variation fulfils the criterion and a message is shown. This user story requires the variations to be previously annotated with the POLYPHEN predicted effect. Table 5.8 describes the corresponding *DSL requirement*, which details the language construct, two acceptance tests, and one dependency. The first acceptance test checks the normal behavior of the construct, while the second acceptance test describes an erroneous usage and the corresponding error message. This language construct has a precondition with the construct *annotate variations with POLYPHEN predicted effect*.

Table 5.7 End-user requirement for "*Filter Variations by POLYPHEN predicted effect*"

User Story Filter Variations by Polyphen predicted effect			
Description	As a geneticist, I want to filter the sample's variations by the predicted effect by POLYPHEN (probably_damaging, possibly_damaging, benign), so that I can see only the variations that pass the filter"		
Role	**Mandatory**	**Action**	**Goal**
Geneticist	No	Filter sample's variations by a set of POLYPHEN predicted effects (benign, possibly_damaging, probably_damaging)	Seeing only the variations that pass the filter

Acceptance Test AT1			
Description	As a geneticist, given the variations chr2:g.136438366A>G {}, chr11:g.111959693G>T {probably damaging}, chr17:g.41245471C>T {benign}, when I filter the variations by the POLYPHEN predicted effect possibly damaging I will see the variation chr11:g.111959693G>T		
Role	Input	Action	Response
Geneticist	chr2:g.136438366A>G {} chr11:g.111959693G>T {probably damaging} chr17:g.41245471C>T {benign}	Filter by POLYPHEN damaging	chr11:g.111959693G>T {probably damaging}

Acceptance Test AT2			
Description	As a geneticist, given the variations chr2:g.136438366A>G {}, chr11:g. 76255523 G>T {probably damaging}, chr11:g.111959693G>T{}, chr17:g.41245471C>T {benign}, when I filter the variations by the predicted effect probably damaging I will see a message saying that "None variation has the desired predicted effect"		
Role	Input	Action	Response
Geneticist	chr2:g.136438366A>G {} chr11:g.111959693G>T{} chr17:g.41245471C>T {benign}	Filter by POLYPHEN damaging	"None variation has been annotated by POLYPHEN with the desired predicted effect"

Dependency DP1		
Description	If variations have not been annotated with POLYPHEN predicted effect, when I filter variations by POLYPHEN predicted effects, I will see the error "Variations must be annotated with POLYPHEN predicted effect before filtering"	
Precondition	Action	Error Message
Annotate variations with POLYPHEN predicted effect	Filter variations by a set of POLYPHEN predicted effects	"Variations must be annotated with POLYPHEN predicted effect before filtering"

Table 5.8 DSL requirement for Filter Variations by POLYPHEN predicted effect

User Story Filter by Polyphen predicted effect			
Description	As DSL user, I want to order a filter by a list of POLYPHEN predicted effects, so that variations can be filtered by these predicted effects		
Role	**Mandatory**	**Action**	**Goal**
DSL user	No	Write Filter and a list of POLYPHEN predicted effects	Variations can be filtered by these predicted effects
Acceptance Test AT1			
Description	As a DSL user, given annotate variations with POLYPHEN predicted effect, when I write filter and the POLYPHEN predicted effect probably damaging, I will see the source code that filter the variations by this predicted effect.		
Role	Input	Action	Response
DSL User	Annotate variations with POLYPHEN effect	Write Filter and predicted effect probably damaging	Source code that filters variations by the POLYPHEN predicted effects "probably_damaging"
Acceptance Test AT2			
Description	As a DSL user, when I write filter by the POLYPHEN predicted effect harmful, I will see an error saying that the predicted effect harmful is not a POLYPHEN predicted effect.		
Role	Input	Action	Response
DSL User	Annotate variations with POLYPHEN predicted effect	Write Filter and predicted effect harmful	Error: "The predicted effect harmful is not a POLYPHEN predicted effect. The predicted effects must be benign, possibly damaging or probably damaging"
Dependency DP1			
Description	If annotated with POLYPHEN predicted effect has not been written, when I write filter and a list of POLYPHEN predicted effects, I will see the error "Variations must be annotated with POLYPHEN predicted effect before filtering"		
Precondition	Action	Error Message	
Write Annotate variations with POLYPHEN predicted effect	Write Filter and a list of POLYPHEN predicted effects	"Variations must be annotated with POLYPHEN predicted effect before filtering"	

Table 5.9 describes the usage scenario "*Analyze Diabetes Mellitus Type 2 (Analysis 1)*". This scenario contains, among other requirements, the requirement "*Filter Variations by POLYPHEN predicted effect*" described in Table 5.7 and Table 5.8.

Table 5.9 Usage scenario template to describe one analysis of Diabetes Mellitus Type 2

Usage Scenario	Usage Scenario Diabetes Mellitus Type 2 (Analysis 1)
Description	In order to research the diabetes mellitus type 2 disease: I want to read the genotypes of several samples from a VCF file. I want to annotate the variations with their genes, with all the names of the transcripts that they hit, and the score and predicted effect of POLYPHEN. I want to filter the variations by the diabetes genes "ABCC8, CAPN10,KCNJ11, GCGR, SLC2A2, HNF4A, INS, INSR, PPARG, TCFl2, ADIPOQ, AKT2, PAX4, MAPK81p1, GPD2, MNTR1B", and by "possibly damaging" or "probably damaging" variations according to POLYPHEN. I want to create a report with the variations main properties, their genes, their transcript names, and their POLYPHEN predictions.

The last step of the analysis stage is the **domain modeling** (Section 5.2.3), in which developers obtain the feature model, the concepts model, the vocabulary, and the relationships between the feature model and the concepts model.

Figure 5.8 shows a partial feature model that is related to the illustrative example. This feature model gathers three main features: *Processing Sample Data (Sample Data)*, *Analyze a set of Variations (Variation Analysis)*, and *Report the Analysis Results (Report)*.

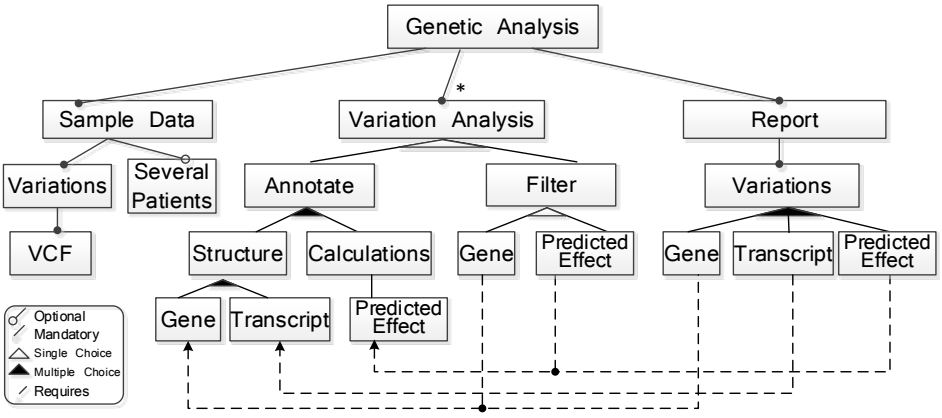


Figure 5.8 Feature model of iteration 3

In order to create this feature model, we applied the model-based guidelines previously described that obtain the necessary information from the user story templates:

- The feature *Genetic Analysis* is the root feature that represents the domain and gathers all the instances supported by the DSL.

- For each user story template, we created a feature whose name is a summary of the *description* field or is extracted from the *action* field. For instance, the features *Filter* and *Predicted Effect* of Figure 5.9 are due to the *action* field of the user story template "*Filter Variations by a set of POLYPHEN predicted effects" (*shown in Table 5.8). At the moment, the relationship between the feature *Filter* and *Predicted Effect* is single choice because the user story is optional.
- For each dependency between the user stories, we have created a *require* link between features. For example, Figure 5.9 shows the correspondence between the *require* link from the feature *Predicted Effect* (child of *Filter*) and the feature *Predicted Effect* (child of *Annotate*), and the dependency *If annotated with POLYPHEN predicted effect has not been written, when I write filter and a list of POLYPHEN predicted effects, I will see the error "Variations must be annotated with POLYPHEN predicted effect before filtering"* , described in Table 5.7.
- In general, we organized the feature hierarchy according to the commonalities and variabilities presented in the different user stories. For example, we created the parent feature *Filter* of Figure 5.9 by grouping the two user stories that described the need of *filtering by gene* (feature *Gene)* and the need of *filtering by predicted effect* (feature *Predicted Effect).*

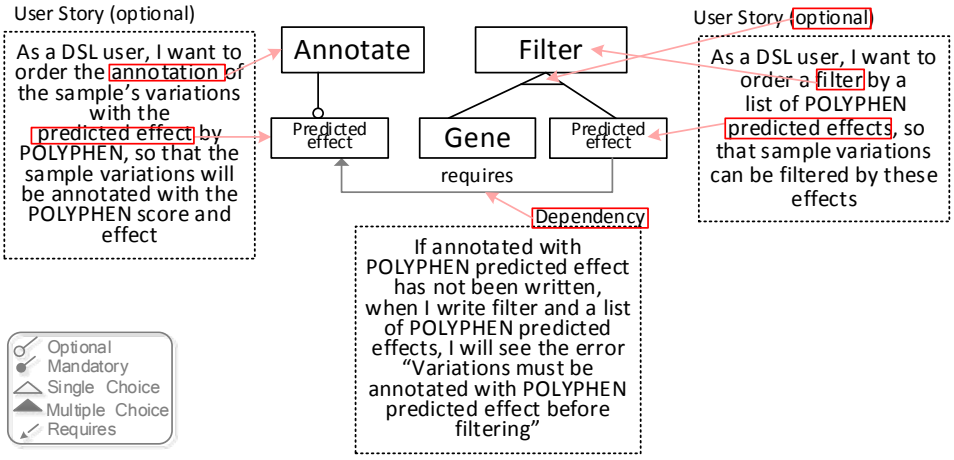


Figure 5.9 Example that illustrates the creation of the feature model

Figure 5.10 shows a partial concepts model that is related to the illustrative example. A *genetic analysis* is performed over a *sample* that gathers a list of genetic *variations*, in which different genetic attributes are identified.

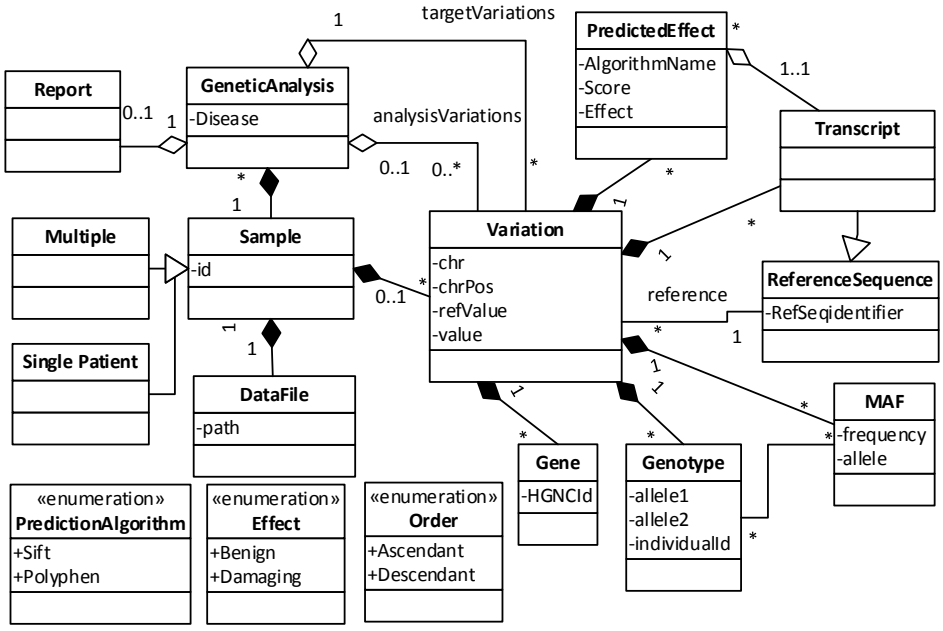


Figure 5.10 Concepts model of the third iteration

In order to create the UML class diagram, we applied the model-based guidelines previously described that obtain the necessary information from the user story templates:

- For each user story, we identify the keywords of the domain that are written in the fields *action* and *goal* of the user stories. For example, Figure 5.11 shows the correspondence between the entities *Sample*, *Variation*, and *Predicted Effect* and the attributes *Algorithm Name*, *Effect,* and *Score* from the user story "*As a DSL user, I want to order the annotation of the sample's variations with the predicted effect by POLYPHEN, so that the sample variations will be annotated with the POLYPHEN score and effect*". The attribute associated with the keyword POLYPHEN is translated to Algorithm name because the specific algorithm used it is a technological decision not a domain concept.

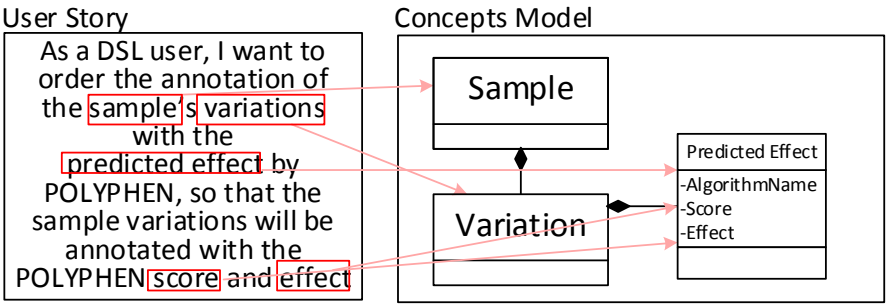


Figure 5.11 Example that illustrates the creation of the concepts model

In order to define the vocabulary, we asked the geneticists to define each of them. The glossary of terms related to the illustrative example are:

- Genetic Analysis: Analysis that is performed to individuals by observing their genetic data.
- Report: Relevant information gathered as a result of a genetic analysis.
- Sample: Object of study to perform a genetic analysis (one or several individuals).
- Single (Sample): When the object of study is a single individual.
- Multiple (Sample): When the object of study are several individuals.
- Datafile: Genetic data of the sample saved in a textual file.
- Variation: Each of the nucleotides that the sample has different in regards to a reference sequence.
- Reference Sequence: A representative sequence of nucleotides that theoretically represents the sequence of a "disease free" human.
- Gene: Functional unit that delimits a subset of nucleotides from the DNA sequence. A gene regulates a function of the body.
- Transcript: Functional structure of the gene that represents the parts that play a role in the transcription of the nucleotides of the genes to proteins.
- Effect Prediction: Result of the execution of a prediction algorithm that assesses the effect of the variation in an individual.
- Genotype: Two alleles of an individual in a position in the chromosome.

Finally, we defined the relationships between the feature model and the conceptual model. Figure 5.12 shows two of the relationships between the models of the illustrative example (Figure 5.8 and Figure 5.10). The feature *Filter->Gene*

and the entity *Gene* and the relationship between the feature *Filter->Effect Prediction* and the entity *PredictedEffect*.

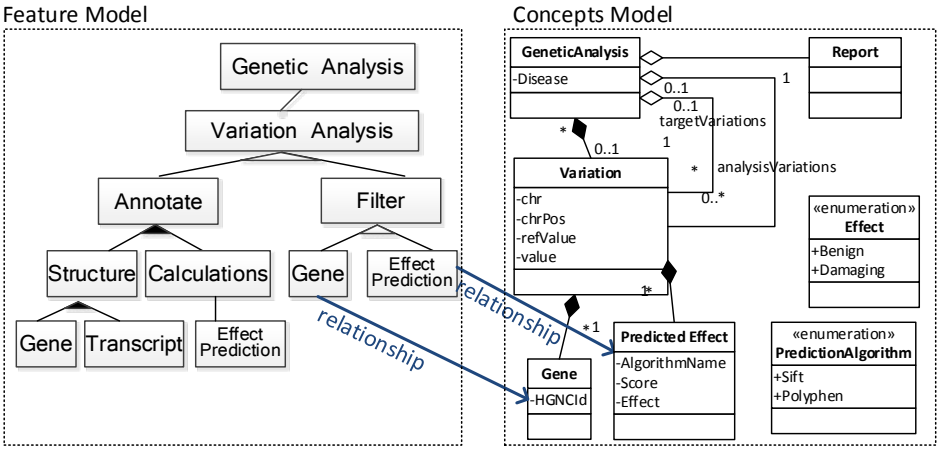


Figure 5.12 Relationships between the models of the genetic analysis example

In order to obtain these relationships, we applied the proposed model-based guidelines:

- For each user story template, we search domain concepts in the field *action*. Figure 5.13 shows the correspondence between the feature *Predicted effect* and the entity *Effect Prediction* of the concepts model. Additionally, this relationship has the cardinality one-to-many due to the keyword *list* of the field *action*.

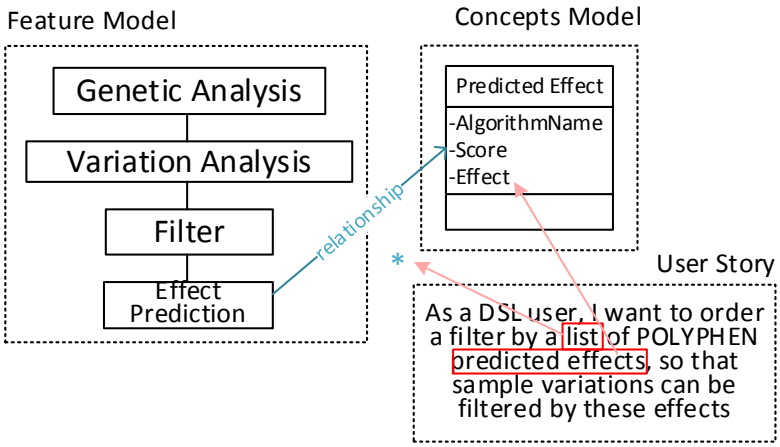


Figure 5.13 Example that illustrates how to relate the feature model and the concepts model

5.3 Conclusion

In this chapter, we have explained the two first stages of the DSL development process: Decision and Analysis. We have explained these stages together because both of them aim to bring closer end-users and DSL developers. In these stages, end-users tell developers what are the particular features of their domain and what are their specific needs. The Decision stage is the first step into this path and when the potential benefits of developing a DSL are clear, further details of the domain are discussed in the Analysis stage.

Regarding the Decision stage, our approach does not provide anything new to the state of the art since we adopted an already existing set of decision patters. Our contribution to this stage is to assert the benefits of this existing proposal by applying it in practice.

Regarding the Analysis stage, our approach contributes to the state of the art by finding the balance between the agile practices that describe requirements and the models that make explicit the domain of the DSL. First, we have established the difference between end-user requirements and DSL requirements and how to address each type. Second, as mechanism M1, we have adopted user stories, acceptance tests, and usage scenarios to engage end-users in describing a few requirements of the system in a structured way. Thanks to this structure (third),

we have proposed a set of guidelines to obtain the domain model of the DSL explicitly in form of a feature model, a concepts model, and a vocabulary. Although full automation of these guidelines is not supported at the moment, these three models can be generated systematically by the developers.

As drawbacks, we have not dealt with non-functional requirements. Although user stories and acceptance tests could be used in principle to define non-functional requirements, we have not studied its application in practice. This is a challenging problem left for our very next future work.

6. Realizing the Solution: The Design and Implementation Stages

Once the developers have acquired and formalized the appropriate knowledge from domain experts, the next step is realizing the DSL that will support the end-users' needs. With this aim, the developers design the language that is going to be created (although the end-users also participate to ensure that their needs are well represented). This design includes the syntax, which describes the structure of the DSL, and the semantics, which describes the underlying behavior of the DSL. In our method, these elements are created in the Design stage.

Once this design is complete, developers implement the technological support of that design. End-users do not participate because implementing only implies to deal with technological concepts nor domain concepts. In our method, this technological solution is created in the Implementation stage.

In this chapter, we explain the Design and Implementation stages and how we applied each of them for developing of a DSL for the genetic analysis domain. We created several versions of the method and the DSL; however, in order to simplify the explanation of the method, we focus only on the method version that

corresponds to the last iteration. Similarly, in order to simplify the explanation of the application of the method to build the DSL, we only provide fragments of the DSL in regards to the illustrative example presented in Chapter 4.

In summary, we start describing the Design stage and how we applied this stage in the real use case and then, we proceed equally to describe the Implementation stage.

6.1 The design stage

The goal of the Design stage is to provide a design of the language to be developed. The goal of this stage is to specify a syntax, which describes the structure of the language by means of language constructs[8], and the semantics, which describes the functional meaning of each syntax construct. On the one hand, the syntax of a language (artefact DA1, Figure 4.4) is defined by means of two artefacts: 1) the abstract syntax, which describes the concepts of the language and the existing relationships among them; and 2) the concrete syntax, which describes the specific symbols, textual or visual, that are used to refer to the concepts and relationships of the abstract syntax. On the other hand, the semantics (artefact DA2, Figure 4.4) is also defined by means of two artefacts: 1) semantics restrictions, which express facts or conditions that should be fulfilled by syntax elements; and 2) behavioral semantics, which describe the meaning of syntax elements in the specific target domain.

Table 6.1 shows the steps to design all these artefacts, the model-based transformations guidelines for developers, and the mechanisms proposed for gathering end-users' input. All these elements were introduced in Section 4.2. Steps 3.1 and 3.2 belong to the syntax design, and steps 3.3 and 3.4 to the semantics design.

[8] A language construct is a set of tokens (or a set of graphical elements) that are syntactically correct according to the rules of a language.

Table 6.1 Overview of the Design stage

Step	Step Description	Artefact	Model-based transformations guidelines	Mechanism for gathering end-user input
3.1	Syntax preferences	Internal/External Decision	-	Interview to identify preferences
3.2	Abstract and concrete syntax design	Syntax Models (DA1): Metamodel and grammar	Transformation guidelines (from the domain model)	Mechanism M2: A questionnaire based on usage scenarios
3.3	Semantic restrictions design	Metamodel constraints and grammar rules (DA2)	Transformation guidelines (from the domain model)	-
3.4	Semantic behavior design	Service model (DA2)	-	Mechanism M3: Template based on service specification and user stories

6.1.1 Syntax preferences

The step **syntax preferences** inspects the decision whether designing an internal DSL, which is using an existing language as a base for the new language, or an external DSL, which is creating a new language with its own syntax [10]. This decision is only addressed in the first iteration and it is not revisited unless the end-users demand major changes in the language.

In order to make this decision, the developers should assess: 1) if the end-users are familiar with any existing language; and 2) the advantages and disadvantages of each option for the specific end-users. According to [90] this decision depends on their preferences regarding:

1. **Availability of a programming context**: The possibility of having programming libraries of another general purposed language while creating DSL specifications.
2. **Syntax flexibility**: The degree of restrictions to express the constructs of the DSL.
3. **Language cacophony**: The necessity to learn a new language.

Hence, in order to involve end-users in this decision, we propose to ask them the following questions:

1. **Existing language**: "Have you ever used a programming language to perform an analytic task about your domain? If so, which language?"
2. **Availability of a programming context**: "When you perform an analytic task about your domain, would you like to have the possibility to use external programming libraries?"
3. **Syntax Flexibility**: "Would you like to use a set of predefined words and symbols to perform your analytic tasks?
4. **Language learning**: "Do you mind to learn a new language?"
5. **Priorities**: "What are your priorities regarding the knowledge of an existing language, availability of a programming context, syntax flexibility and language learning

In order to make the final decision, the developers assess the end-users' responses and decide between internal and external according the end-users' needs and preferences. Table 6.2 shows two examples of this decision-making approach. For end-users 1, the most suitable solution is an internal DSL that is based on a language they already know. They know an existing language, they would like to use existing programming libraries of this language, they don't need a special syntax, they don't want to learn a new language, and the most important for them is to use this language and not to learn a new one. On the contrary, for end-users 2, the most suitable solution is an external DSL. They do not know any programming language, they need syntax flexibility, they don't need to have any programming context available besides the DSL, they don't mind to learn a new language and the most important for them is to have syntax flexibility.

Table 6.2 Examples of internal/external decision

	Language already known	**Programming context**	**Syntax Flexibility**	**New learning**	**Piorities**	**Decision**
End-users 1	Yes (internal)	Yes (internal)	No (internal)	No (internal)	Language known and new learning	Internal
End-users 2	No (external)	No (external)	Yes (external)	Don't mind (internal/external)	Syntax flexibility	External

For practical reasons, initially our method only addresses the development of external DSLs and leaves internal DSLs for future work. The reason for this decision was driven for the thesis context. One of our goals was to provide a complete DSL development approach that could be applied in practice, so we

decided to continue with the next stages of DSL development, instead of focusing only on all of the details of the design stage.

The specific decision of supporting first an external approach was motivated by the other goal of this PhD of building a DSL for the genetics analysis domain. We assessed the preferences of the geneticists from GEM Biosoft and Imegen and the most suitable approach for them was an external DSL since: 1) they knew a language but it was not suitable enough to describe their analyses; 2) they did not needed specific programming libraries; 3) they did not had restrictions in the language syntax as long as it was easy; and 4) they were willing to learn a new language if it was worth.

6.1.2 Abstract and concrete syntax design

Once the decision between internal or external DSL is made, the goal of the step **abstract and concrete syntax design** is to design the syntax models (DA1) that represent the language requirements gathered in the analysis stage and the end-users' preferences.

For the same reason that we chose to support first external DSLs, in this step, we focused on textual syntaxes, and we left graphical syntaxes for future work. Therefore, since the method focuses on textual syntaxes, following the guidelines from Strembeck et al. [12] and Voelter et al. [22], we propose to describe the abstract syntax using a metamodel and the concrete syntax using a grammar.

In order involve the end-users in the syntax design, this step is decomposed into four sub-steps (Figure 6.1): 1) designing the abstract syntax metamodel draft; 2) designing several syntaxes with different structures and styles that are compliant with the abstract syntax metamodel; 3) creating a questionnaire to ask end-users about the abstract syntax and the different concrete syntax options; and 4) refining the analysis models and the syntax models according to the feedback gathered by the questionnaire. Next, we detail these four sub-steps.

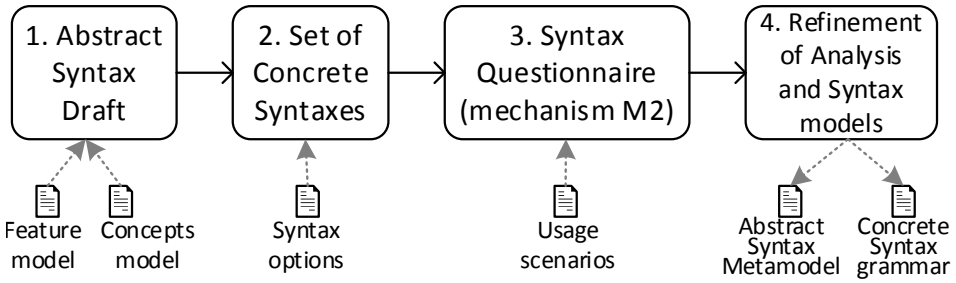


Figure 6.1 Substeps to design the abstract syntax and concrete syntax

In the first sub-step, a draft of the abstract syntax metamodel is designed. Since end-users don't usually have experience designing metamodels, they cannot collaborate in this modelling task, so the developers have to obtain the knowledge from the information gathered in the analysis stage (analysis models). With this aim, we propose a set of guidelines made up of a set of model-based transformations to extract the knowledge from the analysis models (artefact AA3) and represent it into the abstract syntax metamodel (artefact DA1). The guidelines are the following (Figure 6.3):

- The complete feature model is projected as a metamodel. Features become entities of the metamodel and the relationships among these features become composition or specialization relationships. In Figure 6.2, the projection of entities is illustrated by the arrow 1.
- When a relationship among two features of the feature model is mandatory or optional, this information is projected in the metamodel as a composition relationship among the two corresponding entities of the metamodel. In Figure 6.2, this projection is illustrated by the arrows 2 and 3. The mandatory or optional property of a feature of the feature model is represented in the metamodel by establishing the cardinality of the composition relationship. If the feature is mandatory, the cardinality is 1. If it is optional, the cardinality is 0..1. In Figure 6.2, this projection is illustrated by the arrow 3.
- When a relationship among one parent feature of the feature model and several child features is single option or multiple option, these relationships are projected in the metamodel as specializations, one per child feature. If this relationship is single option, the projected specializations are disjoint. On the contrary, if this relationship is

multiple option, the projected specializations are overlapping. Figure 6.2, this projection is illustrated by the arrows 4 and 5.

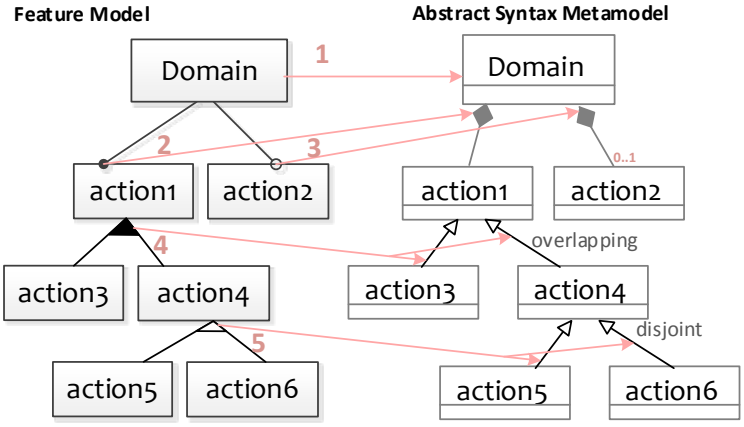


Figure 6.2 Example that illustrates the creation of the abstract syntax metamodel

- Relationships among features of the feature model and concepts of the concepts model (feature-to-concept relationships) are also projected in the metamodel. The concepts involved in a feature-to-concept relationship are projected as new entities of the metamodel and the relationship is projected as an association. In Figure 6.3, this projection is illustrated by the arrow 2.
- Additionally, after projecting a concept of the concepts model in the metamodel, if the concept has a composition relationship, this relationship is also projected in the metamodel as a composition. In Figure 6.3, this projection is illustrated by the arrow 3.

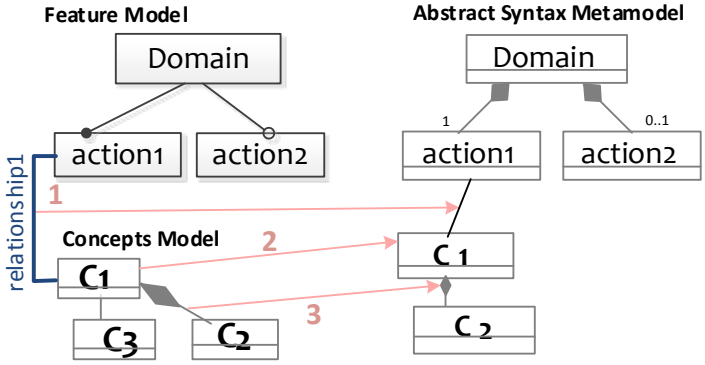


Figure 6.3 Example that illustrates the creation of the abstract syntax metamodel

In the second sub-step, several concrete syntax grammars with different structures and styles are designed to offer the end-users several options to choose. In order to design each of these syntax options, we propose a set of guidelines with model-based transformations to create the grammar that complies with the designed abstract syntax metamodel. These guidelines are the following (Figure 6.4):

- The complete abstract syntax metamodel is projected as a grammar: Entities become non-terminals and relationships between entities become production rules. In Figure 6.4, this projection is illustrated by the arrows 1 and 2.
- Each composition relationship in the metamodel becomes a production rule: the container entity becomes the non-terminal of the left side of the rule and the contained entities the non-terminals of the right side of the rule. If the cardinality of the composition relationship is 0..1, the non-terminal is optional. If the cardinality is 1, the non-terminal is mandatory. In Figure 6.4, this projection is illustrated by the arrows 3 and 4.
- When an entity of the metamodel is specialized into several sub-entities, a production rule is created: the super-entity becomes the non-terminal of the left side of the rule and each sub-entity becomes a non-terminal of the right side of the rule. If the specialization is disjoint, each non-terminal represents an option of the instantiation of the rule. If the specialization is overlapping, each non-terminal is optional.
- The entities of the metamodel that do not have any composition or generalization relationship become the terminals of the grammar. In Figure 6.4, this projection is illustrated by the arrow 6.
- The label of the production rules and terminals are extracted from the concrete syntax selected by the end-users or from the vocabulary of terms from the analysis.

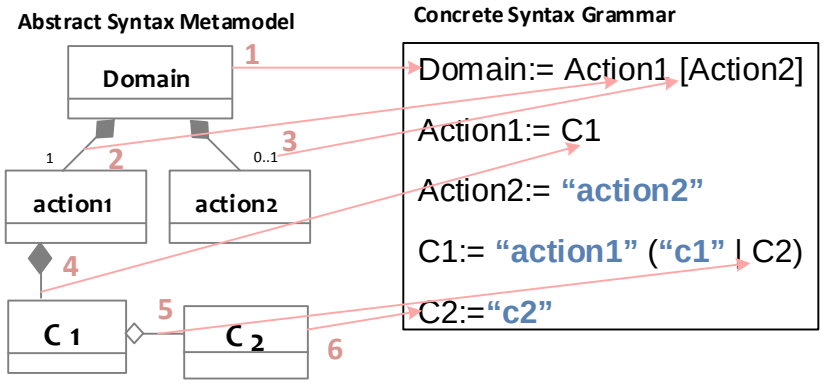


Figure 6.4 Example that illustrates the creation of the concrete syntax grammar

In practice, it is not always necessary for developers to apply these guidelines manually, since there are technological frameworks such as Xtext [91] that implement them already. Specifically, Xtext provides a function that reads a metamodel implemented using EMF [92] and generates a draft of a grammar that complies with that metamodel. Then, developers must only change the tokens of the grammar rules and the terminals in order to customize the concrete syntax grammar. However, we have introduced the guidelines in case of using another implementation approach.

In the third sub-step, in order to illustrate to the end-users both the abstract syntax and the different concrete syntax options, as **mechanism M2**, we propose to use a questionnaire based on usage scenarios (from Scrum). We used questionnaires because it is a very well-known practice to gather information from end-users and we used the agile practice "usage scenarios" to illustrate the different concrete syntaxes with a domain example.

In order to create this questionnaire, the developers chose one of the usage scenarios from the analysis (artefact AA2) and specify this scenario (using a textual editor) with each concrete syntax proposed. Using all these created example specifications, the developers design a set of questions that ask the end-users: 1) to rate each concrete syntax option; 2) to choose the most suitable one; 3) to propose a new concrete syntax (if needed); and 4) to validate the correctness of each syntax construct (abstract syntax).

Figure 6.5 shows an example of a question that asks the end-users to rate one concrete syntax option. In order to rate the suitability of each syntax option we

use a Likert scale [93] from 1 to 5. Also, Figure 6.6 shows a free text question in which end-users may suggest changes in their favorite syntax.

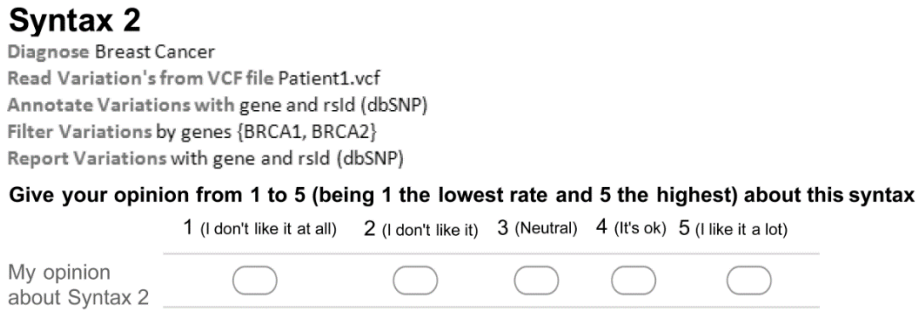


Figure 6.5 Question about the suitability of a specific concrete syntax using a Likert Scale

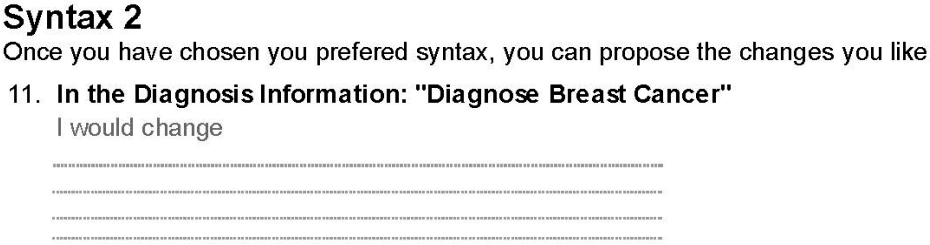


Figure 6.6 Question to suggest syntax changes using free text

Finally, in the fourth sub-step, after the end-users have answered the questionnaire, the developers analyze their responses and create the definitive abstract syntax metamodel and concrete syntax grammar (artefact DA1). First, the developers identify which is the most preferred syntax among the end-users. Then, they design the corresponding grammar rules of the concrete syntax grammar according to this syntax. Finally, they analyze the corrections or changes proposed in the scenario by the end-users and refine the entities, relationships and restrictions of the abstract syntax metamodel accordingly.

6.1.3 Semantic restrictions design

In order to design the semantics, in the step **semantic restrictions design**, the developers incorporate into the syntax models (artefacts DA1) all the domain restrictions that must be ensured when creating domain instances: constraints over the syntax constructs and in the relationships among them. Since we

specified the syntax using a metamodel and a grammar, semantic restrictions are described both as metamodel constraints and grammar rules.

Equally to the syntax design, it is usually very difficult for end-users to contribute to metamodels and grammars. For this reason, we propose a set of guidelines for developers to extract the restrictions represented in the analysis models (the user story templates that describe DSL requirements, artefacts AA2; and the feature model and the concepts model, artefacts AA3) into the abstract syntax metamodel and the concrete syntax grammar. The guidelines to design the semantics restrictions are (Figure 6.7):

- Each acceptance test that describes how to deal with an error in a DSL construct expression is projected as a restriction in the concrete syntax grammar and in the abstract syntax metamodel. This restriction is usually represented using enumerations in the abstract syntax metamodel and using data types in the concrete syntax grammar.
- Each dependency between two features is projected as a restriction in the abstract syntax metamodel. This restriction is represented as an integrity constraint in the metamodel and it is expressed using pseudo-code. In Figure 6.7, this projection is represented by the arrows 1, 2 and 3.
- In order to customize the error messages of the integrity constraints, the messages are extracted from the errors described by the end-users in the acceptance tests of the user story templates. In Figure 6.7, this projection is represented by the arrow 4.

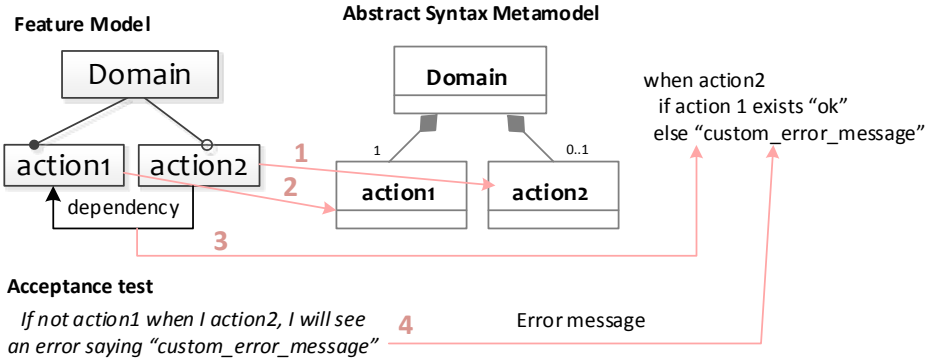


Figure 6.7 Example that illustrates the creation of the semantic restrictions

6.1.4 Semantic behavior design

In the step **semantic behavior design** step, the developers describe the behavior of the different language constructs by means of actions that must be accomplished. For instance, the language construct *upload DNA file using ftp* implies to execute an upload service to a web server using an ftp client.

Following the agile practice *architectural envisioning* from Agile Modeling, we propose to describe the semantics behavior by establishing a mapping between each syntax construct (one or several entities of the abstract syntax metamodel) and a technological artefact that implements its behavior; such as a command line tool, a query to a database, a call to a web service, an execution of a data processing utility, etc.

In this step, the participation of the end-users is essential because they are the experts of the domain and they usually know about the software or tools that could achieve the expected behavior. From their experience, they will identify which technological artefact is the most suitable to provide each corresponding behavior.

In order to specify this mapping between a syntax construct and a technological artefact, as **mechanism M3** we adopted the "Service Abstract Interaction Unit" proposed by [94], which is a template that describes a mapping between a class of a model and a service. First, we analyzed which fields were necessary to describe a semantic mapping between a syntax construct and a service for the DSL context. Then, we adopted the majority of the fields of this template and we discarded the fields: *alias,* because it had no application in the DSL context, and the field *errors*, since the errors of the DSL were already described in the analysis stage by means of acceptance tests.

In order to facilitate the comprehension of this template to the end-users, we added the field *user story* to avoid the inclusion of the notion of a syntax construct. This abstraction is possible because one user story represents a single syntax construct. This way, the end-users establish the mapping between a domain requirement that is being represented by a single user story and a technological artefact, instead of a syntax construct.

Table 6.3 shows the content of a this template: 1) *User story* provides the title of the user story associated with the syntax construct whose semantics are being specified; 2) *Service identifier* provides the name that identifies the technological

artefact that implements the functionality or behavior of the user story; 3) *Service information* provides additional details about the service, such as the service provider, version, prerequisites, etc.; and 4) *Input* and *Output* sections describe the arguments that the technological artefact consumes and produces. Each argument has a *Description* in natural language and a *Type* (a generic data type such a String or Boolean). Inputs contain a flag *Constant*, which indicates if the argument is fixed (value *True*) and a field *Value,* which specifies the fixed content when this is the case. Outputs contain the field *Visibility*, which indicates if showing the output is relevant for the end-users.

Table 6.3 Template to describe semantics behavior

User story	Action of the user story			
Service identifier	serviceId			
Service information	Type of service and service provided			
Inputs	Description	Type	Constant	Value
Input1	description	Type1	No	-
Input2	description	Type2	Yes	Predefined_value
Outputs	Description	Type	Visibility	
Input3	description	Type3	Yes	

6.1.5 The design of the genetic analysis DSL

In this section, we show how we applied the proposal to design a DSL for genetic analysis in collaboration with geneticists. Regarding the **syntax preferences** (Section 6.1.1), we had to decide together with geneticists between implementing the DSL using an internal or external approach. In order to address this decision we met with them to ask about the following features:

- Existing language: The geneticists manifested that programming with a scripting language is the most suitable approach to have full control over their genetic analyses. They have learned how to write scripts, but they manifested that acquiring all these programming knowledge was difficult and teaching all these acquired knowledge to other geneticists is a time consuming task. There are workflow environments like Taverna [51] or Galaxy [52] that aim to avoid the geneticists to learn all this technical knowledge, but the geneticists think that these are still too technical for those who do not have programming expertise. Regarding this feature, they again prefer an external DSL.
- Availability of a programing context: Although the geneticists write scripts to customize their genetic analyses, they do not learn general

programing languages to create new software tools or to customize the existing ones. Also, they do not require any specific programming library to perform a task of their analyses. Regarding this feature, they prefer an external DSL.

- Syntax flexibility: The geneticists don't mind the structure of the language as long as it can be used to specify all the parameters related with their genetic analysis. Regarding this feature, they don't mind if the DSL is external or internal.
- Language cacophony: The geneticists do not want to learn a new language, but they are willing to make the effort as long as it is easy, expressive enough, and if they are provided with a usable editor that guides them. Regarding this feature they don't mind if the DSL is external or internal.
- Preferences: The most important for geneticists was to have a new language that could be used to specify their genetic analysis.

In order to make the final decision, we took into account the work of Cuadrado et al. [95], which compares the internal and external approaches to develop a DSL and concludes that in regards with the target audience: 1) end-users tend to perceive that an internal DSL is more complicated to learn because it implies learning a new general purpose language; 2) an external approach is recommended if the end-users may feel intimidated; and 3) the freedom that is offered by an external DSL may allow to satisfy the end-users when they request changes or new specific constructs.

Together with the feedback provided by the geneticists and the previous remarks, we decided that the best was an external approach. The geneticists will have to learn a new language that has been specially designed for their needs that only contains concepts related with their domain.

Once the syntax approach and the structure were decided, the next step was the **abstract and concrete syntax design** (Section 6.1.2), which gathers four sub-steps. The first sub-step is designing the abstract syntax metamodel. Figure 6.8 shows the set of metamodel entities that support the description of the genetic analysis example. This model describes some of the aspects of the genetic analysis example such as: reading sample data; annotating variations with gene, MAF or transcript; filtering variations by effect; or creating a report.

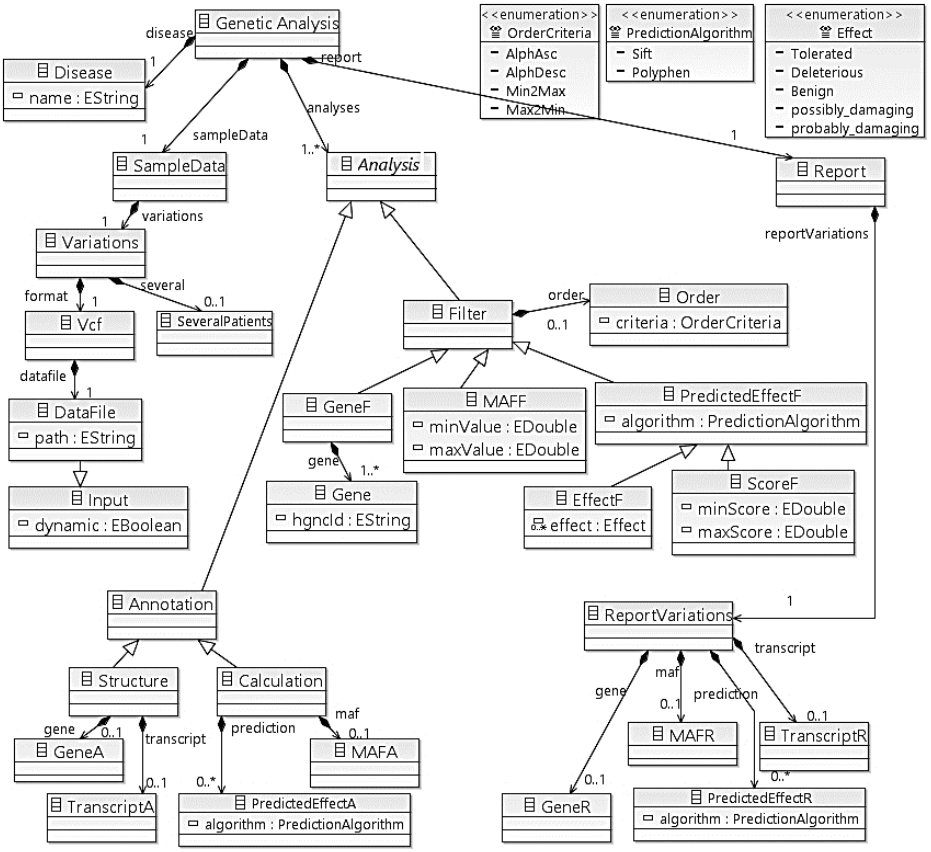


Figure 6.8 Abstract syntax metamodel of the genetic analysis example

In order to create the abstract syntax metamodel, we applied the model-based guidelines to obtain the information from the analysis models (the feature model, the concepts model and the relationships among them) and to create the different entities and relationships of the metamodel:

- Each feature of the feature model is projected as an entity of the metamodel, as well as the hierarchical relationships among them and their cardinalities. For instance, Figure 6.9 shows the correspondence between the features Filter, Gene, and Predicted effect and the metamodel entities Filter, Gene, and Predicted effect. This figure also shows the correspondence between the fact that Gene and Predicted effect are options of the parent feature Filter and the two generalization

relationships in the metamodel of the entities Gene and Predicted effect in relation with the parent entity Filter.

- Each relationship between a feature of the feature model and an entity of the concepts model is projected as a new entity of the metamodel and a composition relationship with the same cardinality than the original relationship. The composite of this relationship is the metamodel entity projected as a result of the feature. For example, Figure 6.9 shows the correspondence between the relationship list_of_predictions (between the feature Predicted effect and the entity Prediction) and the composition relationship of the abstract syntax metamodel between the entity Predicted Effect and the entity Prediction. The cardinality of the original relationship is projected into the new relationship of the metamodel.

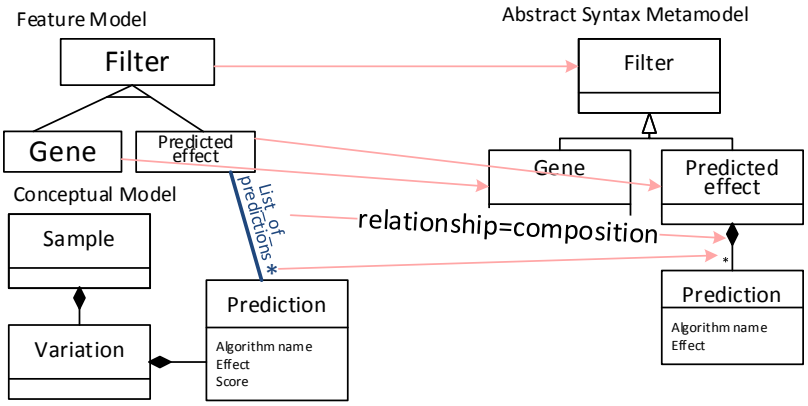


Figure 6.9 Example of application of the guidelines of the abstract syntax metamodel

Once we obtained this metamodel, in order to create the concrete syntax grammar, we used the framework for DSL development Xtext to generate a preliminary draft of the concrete syntax grammar.

The second sub-step is designing several syntaxes with different structures and styles that are compliant with the abstract syntax metamodel. In total, we designed four different syntaxes. Next, in order to show them to the geneticists, we described each of them using the usage scenario of the illustrative example:

1. Descriptive: The genetic analysis is defined by providing the name of an entity of the genetic analysis and the values of their attributes. Figure 6.10 shows the genetic analysis example written with this syntax.
2. Based on natural language: The genetic analysis is defined by giving different orders in natural language. Figure 6.11 shows the genetic analysis example written with this syntax.
3. Object-oriented: The genetic analysis is defined by creating an object of the GeneticAnalysis type and setting their properties by means of class methods. Figure 6.12 shows the genetic analysis example written with this syntax.
4. XML-like: The genetic analysis is defined by creating XML tags and setting the details of the analysis between those tags. Figure 6.13 shows the genetic analysis example written with this syntax.

```
Analyze Diabetes Mellitus Type 2 (Analysis 2)
Read Variations genotypes from VCF file Patient1.vcf
Annotate Variations with gene, transcript, POLYPHEN
Filter Variations by genes {ABCC8, CAPN10,KCNJ11, GCGR, SLC2A2,
HNF4A, INS, INSR, PPARG, TCFl2, ADIPOQ, AKT2, PAX4, MAPK81p1,
GPD2, MNTR1B }
Prioritize Variations by effect prediction {POLYPHEN, damaging} AlphAsc
Report Variations with gene, transcript, POLYPHEN
```

Figure 6.10 Illustrative example written with the descriptive syntax

```
Genetic analysis: Diabetes Mellitus Type 2 (Analysis 1)
Variations Genotypes VCF file: Patient1.vcf
Variations Annotations: gene, transcript POLYPHEN
Analysis Filters: by genes {ABCC8, CAPN10,KCNJ11, GCGR, SLC2A2,
HNF4A, INS, INSR, PPARG, TCFl2, ADIPOQ, AKT2, PAX4, MAPK81p1,
GPD2, MNTR1B}
Analysis Priorizations: by effect prediction {POLYPHEN, damaging}
AlphAsc
Variation report fields: gene, transcript, POLYPHEN
```

Figure 6.11 Illustrative example written with the natural language syntax

```
GeneticAnalysis.Disease(“Diabetes Mellitus Type 2 (Analysis 2)”)
GeneticAnalysis.Sample.Variations.Genotypes ("Patient1.vcf", VCF)
GeneticAnalysis.Sample.Variations.Annotations (gene, transcript,
POLYPHEN)
GeneticAnalysis.Sample.Variations.Analysis.Filter.ByGene(ABCC8,
CAPN10,KCNJ11, GCGR, SLC2A2, HNF4A, INS, INSR, PPARG, TCFl2,
ADIPOQ, AKT2, PAX4, MAPK81p1, GPD2, MNTR1B)
GeneticAnalysis.Sample.Variations.Analysis.Prioritize.ByEffectPrediction.Eff
ect(POLYPHEN, damaging).Order(AlphAsc)
GeneticAnalysis.Sample.Variations.Report.Fields(gene, transcript,
POLYPHEN)
```

Figure 6.12 Illustrative example written with the object-oriented syntax

```
<GeneticAnalysis>
<Disease>Diabetes Type2</Disease>
<SampleData>
  <Genotypes><VCF>Patient1.vcf</VCF></Genotypes>
</SampleData>
<Analyses>
  <Annotate><gene/><transcript/><POLYPHEN/></Annotate>
  <Filter><genes><gene>ABCC8</gene><gene>MNTR1B</gene></genes>
</Filter>
  <Prioritize criteria="effect_prediction" order= “AlphAsc”>
  <POLYPHEN><effect>damaging</effect></POLYPHEN></ Prioritize >
</Analyses>
<Report><Variations><gene/><transcript/><POLYPHEN/></Variations></R
eport>
</ GeneticAnalysis >
```

Figure 6.13 Illustrative example written with the XML-like syntax

The third sub-step is creating a questionnaire to ask the geneticists about the abstract syntax and the different concrete syntax options (syntax questionnaire, mechanism M2). The geneticists were asked to rate each syntax option with a Likert Scale of five levels, being 1 the lowest, when they don't like the syntax option, and 5 the highest, when they like the syntax.

Table 6.4 shows the geneticists' responses about each option, and the syntax that was chosen by each geneticist as their preferred one. Since each of the geneticists chose a different syntax option as its preferred one, in order to identify which was the most preferred syntax among all of them, we had to take into account the geneticists' ratings. Since the scale of rating was ordinal, in order to obtain the representative value of the geneticists' opinion about each syntax, we calculated the median [96]. As a result, the best rated syntax option was the syntax based on natural language, with a median value of 4.

Table 6.4 Geneticists' responses about the different syntax options

	Entity-Based	Natural language	Object-Oriented	XML-like	Preferred
Geneticist 1	5	4	3	2	Syntax 1
Geneticist 2	4	4	5	5	Syntax 3
Geneticist 3	2	3	1	1	Syntax 2
Median	3	4	3	2	-

And finally, the fourth sub-step is representing the gathered feedback into the definitive syntax design. Figure 6.14 shows a fragment of the rules of the concrete syntax grammar of the illustrative example that has been customized with the syntax option chosen by the geneticists.

```
GeneticAnalysis returns GeneticAnalysis:
  'Analyze' disease=disease sampleData=sampleData analyses+=analysis+
report=report;
sampleData returns SampleData:
  'Read'  variations=variations;
variations returns Variations:
  'variations' several=severalSamples  format=vcf;
severalSamples returns SeveralSamples:
  'genotypes'{SeveralSamples};
vcf returns Vcf:
  'from' 'a VCF file'  datafile=dataFile;
```

Figure 6.14 Fragment of the concrete syntax grammar of the illustrative example

Regarding **the semantic restrictions** (Section 6.1.3), we have specified grammar rules and restrictions about the abstract syntax metamodel in natural language. Figure 6.15 shows how we created the restriction *It is mandatory to annotate the POLYPHEN prediction before filtering by POLYPHEN*. In order to specify this semantic restriction, we applied the model-based guidelines previously described that obtain the necessary information from the feature model and the user story templates. We proceeded as follows:

- The dependency between the features Predicted Effect (child of Filter) and Predicted Effect (child of Annotate) is projected as a restriction in the abstract syntax metamodel using natural language: "When the entity PredictedEffectF is created, the entity PredictedEffectA should be present". In the figure, this projection is represented by arrow 3.
- The features that are involved in the dependency are projected in the restriction. The prerequisite feature is projected as the condition to check and it is written after the *if* clause. The dependent feature is projected as the context of the restriction and it is written after the when clause. In the figure, this projection is illustrated by the arrows 1a, 1b, 2a, and 2b.

- In order to customize the error messages of the semantic restriction, the error message is extracted from the field error message of the user story template (Table 5.8): "Variations must be annotated with POLYPHEN predicted effect before filtering". In the figure, this projection is represented by the arrow 4.

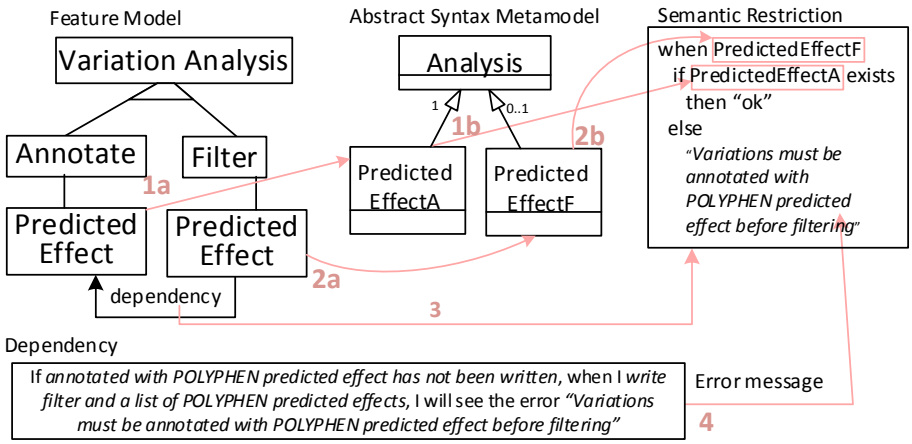


Figure 6.15 Example of application of the guidelines of the semantic restrictions

Regarding the **semantic behavior design** (Section 6.1.4), we defined the semantic templates related to the user stories of the iteration (mechanism M3) in collaboration with geneticists.

In this step, following the agile practice architectural envisioning, together with geneticists we selected the bioinformatics environment Galaxy [52] as the implementation platform. Galaxy is an environment that provides geneticists the possibility to run different biological services and create workflows combining those services (Figure 6.16). This environment can be executed locally, using a web interface, or in the cloud. Galaxy allows geneticists to retrieve local or public data sets (such as the datasets from the USCS database), combine data from independent queries, perform calculations over the retrieved datasets (such as filtering a data set, combining several data sets, and transforming data using a biological service), and visualize the results. Although there are other bioinformatics environments such as Taverna or eBioflow (explained in Chapter 2), or we could have also used Unix scripts, we chose Galaxy as the implementation platform of the DSL for several reasons: its specificity towards

the genetic analysis domain; the extended used of Galaxy among bioinformaticians; and its constant updates regarding functionality and usability.

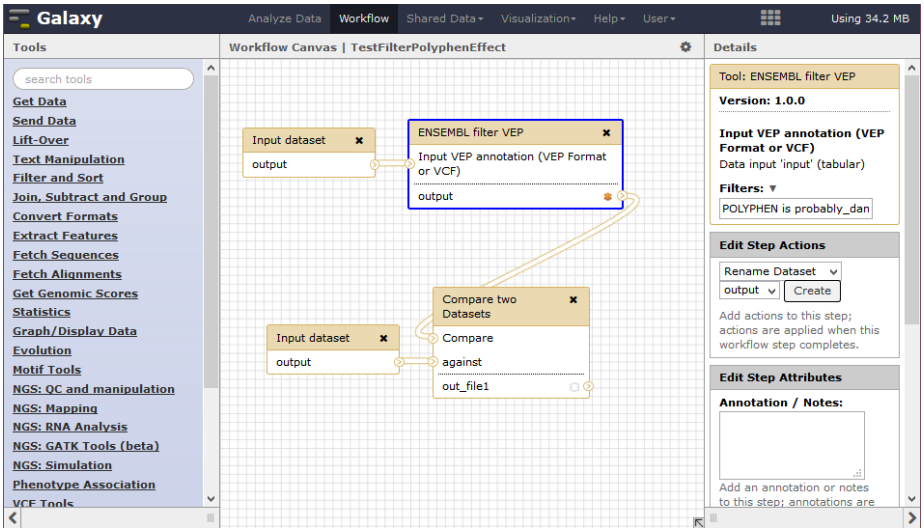


Figure 6.16 Interface of Galaxy

Table 6.5 shows the semantic template fulfilled in collaboration with the geneticists to describe which service from Galaxy can be used to implement the behavior of the user story *Filter Variations by POLYPHEN effect*.

Table 6.5 Semantic template to describe the behavior of the user story *Filter by Polyphen effect*

User Story	Filter Variations by predicted effect POLYPHEN			
Service Identifier	Ensembl Filter VEP			
Source description	Galaxy			
Inputs	Description	Type	Constant	Value
Input	File that gathers the variations	DataFile (VCF)	False	-
FilterCriteria	Evaluation expression that indicates the polyphen criteria to filter	String	False	Examples: "Polyphen is benign" "Polyphen is possibly_damaging"

Outputs	Description	Type	Visibility
annotated_vcf	File that gathers the annotated variations	DataFile (VCF)	True

6.2 The implementation stage

The goal of the *Implementation* stage is to create an executable DSL infrastructure that realizes the design of the language. This DSL infrastructure must support the creation of DSL specifications (which are specifications that are expressed according to the language syntax) and provide the corresponding behavior of those specifications. This DSL infrastructure is formed by a parser, a validator, and a code generator. The parser reads a DSL specification and parses the concrete syntax to obtain the underlying abstract syntax tree[9]. The validator checks the correctness of this abstract syntax tree according to the restrictions of the DSL. The code generator obtains the source code that provides the behavior associated with this abstract syntax tree.

In order to implement this infrastructure, this stage is divided into two steps (Table 6.6): 4.1) creating tests to check the correctness of the syntax and semantics implementation; and 4.2) implementing the DSL infrastructure using both a model-driven development approach (MDD) that takes as input the design models, and a test-driven development approach (TDD) that takes as input the tests specified in the previous step 4.1.

Since the implementation is a highly complex task, end-users do not participate in this stage. Nevertheless, their needs are taken into account because both the models and the tests that are used in the implementation were created (in previous stages) with their collaboration.

Table 6.6 Overview of the Implementation stage

Step	**Step Description**	**Artefact**	**Model-based transformations guidelines**
4.1	Tests specification	Syntax and semantics tests (IA1)	Transformation guidelines (from acceptance tests)
4.2	DSL infrastructure implementation	DSL Infrastructure (parser, validator and code generator) (IA2)	Model-driven development and Test-driven development

[9] An abstract syntax tree is a representation of the abstract syntax structure using a tree in which each node represents one concept of the abstract syntax.

6.2.1 Test specification

In the step **test specification**, the developers address the creation of tests that check the correctness of the syntax and semantics implementation. *Syntax tests* check the parser and *semantics tests* check the validator and the code generator. Besides these tests, the developers also create common unitary tests that check the correctness of end-user requirements. In order to differentiate these last tests from the rest, we named them *target platform tests*, since they check the correctness of the target platform artefacts that are generated by the DSL infrastructure. Eventually, the three type of tests will be used to drive the implementation of the complete DSL infrastructure applying the agile practice test-driven development (TDD) (Figure 6.17).

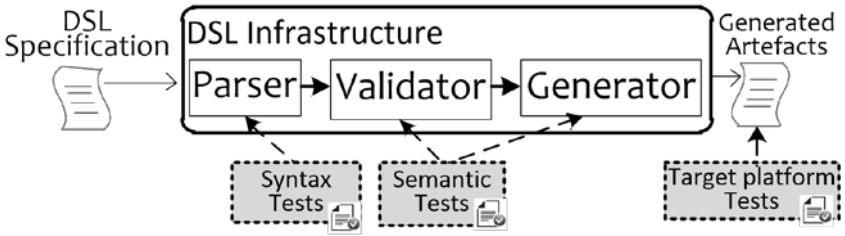


Figure 6.17 DSL infrastructure and tests

In summary, we define three type of tests: syntax tests, semantic tests and target platform tests. As we can see in Figure 6.18, in order to ease the specification of tests, we define a test as an entity that 1) receives two parameters (an *input* and an *assert condition*); 2) executes the artefact to be tested (parser, validator, code generator or generated artefacts) with the parameter *input*; and 3) compares the result of this execution with the parameter *assert condition*.

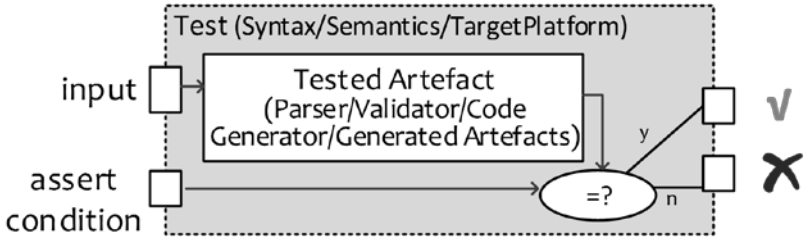


Figure 6.18 Representation of a test

6.2.1.1 Syntax tests

Syntax tests check that the DSL infrastructure parses DSL specifications correctly. There are two type of syntax tests:

- Tests that check that the parser understands the symbols of the language (concrete syntax) and the relationships between them (abstract syntax). The parameter input is a DSL specification and the parameter assert condition is an abstract syntax tree. Figure 6.19 shows an example of a syntax test in which the parameter input is a DSL specification written according to the concrete syntax grammar, the parameter assert condition is the equivalent abstract syntax tree. This syntax test checks that when this input is provided the parser obtains the same abstract syntax tree.

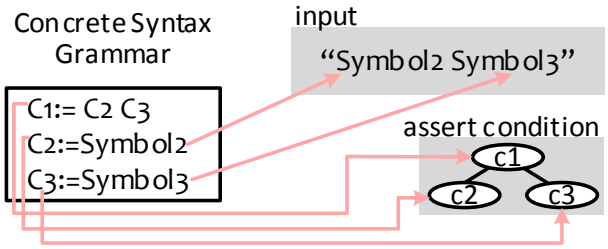


Figure 6.19 Example of a syntax test

- Tests that check that the parser provides an error when an incorrect symbol or an incorrect relationship among symbols is used. Figure 6.20 shows an example of a syntax test in which the parameter input is a DSL specification that contains errors according to the grammar and the parameter assert condition is an error message. This syntax test checks that when this input is provided the parser throws the same error message.

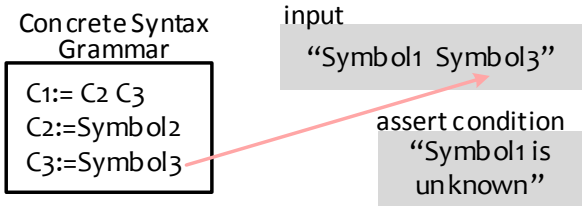


Figure 6.20 Example of a syntax test with errors

In practice, we do not always need to specify this kind of tests because there are technological frameworks (such as Xtext) that generate the parser automatically from the design models. Therefore, when using this approach, it is not necessary to create syntax tests either to check for their correctness or to guide the implementation of the parser.

6.2.1.2 Semantic tests

Semantic tests check that the DSL infrastructure validates the DSL specification and provides the corresponding behavior. There are two types of semantic tests: validator tests, which check semantic restrictions; and code generator tests, which check behavior.

Validator tests check that the validator arises an error when a semantic restriction is violated. The parameter *input* is an abstract syntax tree and the parameter *assert condition* is an error message. Figure 6.21 shows an example of semantic test in which the parameter *input* is an abstract syntax tree that violates the restriction, the parameter *assert condition* is an error message. This validator test checks that when this input is provided, the validator rises the same error message.

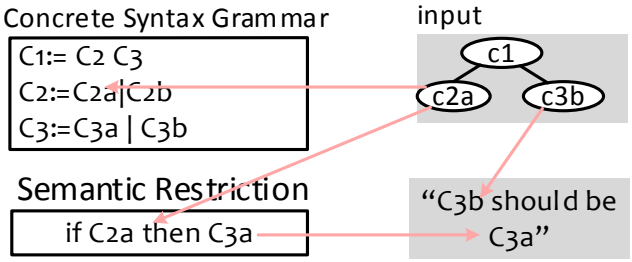


Figure 6.21 Example of a validator test

Validator tests check whether semantics restrictions are implemented correctly. For this reason, in order to create them, we propose to use the semantics restrictions (artefact DA2) and the acceptance tests (artefact AA2) that were used to derive those restrictions (explained in Section 6.1.3). For each pair semantic restriction and acceptance test:

- A new validator test is created to check for the semantic restriction.
- The parameter *input* of the validator test is obtained from the input of the acceptance test. Since the input of the validator test should be an

abstract syntax tree but the input of the acceptance test is a DSL construct, the developers should obtain the equivalent abstract syntax tree. In Figure 6.22, this projection is illustrated by the arrows 1, 2, and 3.

- The parameter *assert condition* of the validator test is obtained from the response of the acceptance test. The *assert condition* is the error message provided by the acceptance test. In Figure 6.22, this projection is illustrated by the arrow 4.

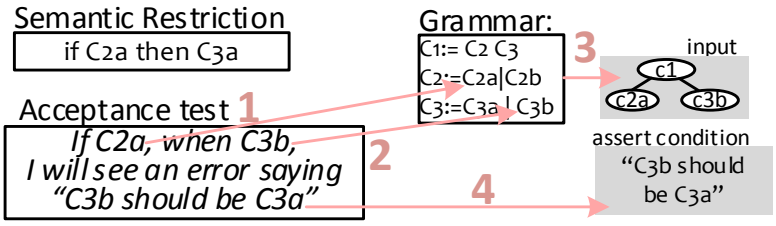


Figure 6.22 Example that illustrates the creation of a validator test

Code generator tests check that the code generator provides the corresponding target implementation artefacts. The parameter *input* is an abstract syntax tree and the parameter *assert condition* is an artefact (usually source code) of the target implementation platform. Figure 6.23 shows an example of a code generator test in which the parameter *input* is an abstract syntax tree that follows the grammar, the parameter *assert condition* is the target platform source code (Java code). This code generator test checks that when this input is provided, the code generator generates the same source code.

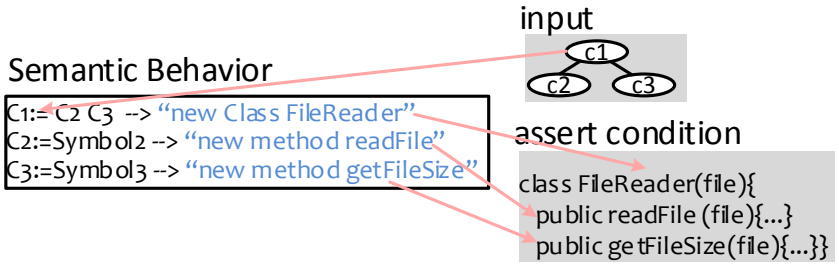


Figure 6.23 Example of a code generator test

Code generator tests check whether semantics behavior are implemented correctly. For this reason, in order to create them, we propose to use the semantics templates (artefact DA2) and the user stories and acceptance tests (artefacts AA2) that were used to fulfill those templates (explained in Section 6.1.4). For each trio semantic template, user story and acceptance test:

- A new code generator test is created for each acceptance test that checks on an expected result.
- The parameter *input* of the code generator test is obtained from the input of the acceptance test. Since the input of the code generator test should be an abstract syntax tree but the input of the acceptance test is a DSL construct, the developers should obtain the equivalent abstract syntax tree. In Figure 6.24, this projection is illustrated by the arrows 1 and 2.
- The parameter *assert condition* of the semantic test is obtained from the response of the acceptance test and the semantic template. Specifically, the *assert condition* is the target platform code equivalent to the response of the acceptance test. In Figure 6.24, this projection is illustrated by the arrow 3.

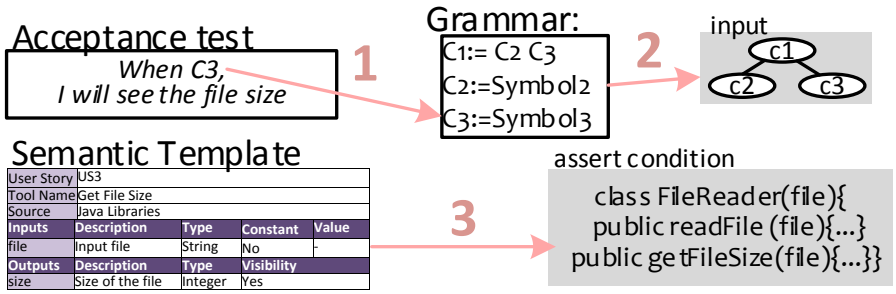


Figure 6.24 Example that illustrates the creation of a code generator tests

As we can see in Figure 6.24, in order to implement code generator tests we need to know the equivalent source code that is being provided as the parameter *assert condition*. Since this equivalent code is not obtained until the next step (step 6.2.2, DSL infrastructure implementation), the specification of this kind of test must be delayed also to the next step.

6.2.1.3 Target platform tests

Target platform tests check that the artefacts generated by the DSL infrastructure work properly according to the end-users' requirements. This kind of tests are common unitary tests that check end-user requirements. The parameter *input* is any data provided by the end-users and the parameter *assert condition* is the expected result. Figure 6.25 shows an example of a target platform test in which the *input* is a file provided by the end-users *TestFile.txt*, the *assert*

condition is the expected result *Size= 20kb*. This target platform test checks that the generated artefacts (an executable made by a set of Java classes) are able to produce the expected result.

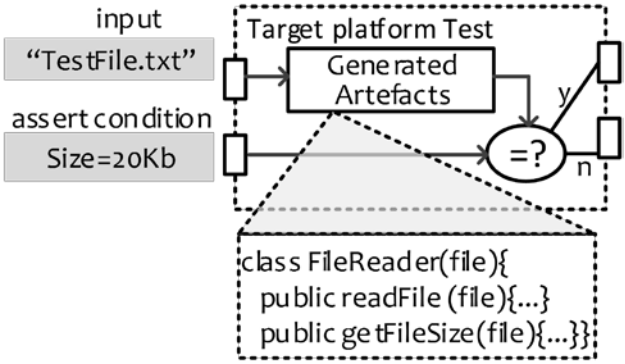


Figure 6.25 Example of a target platform test

Target platform tests check whether the final artefacts that will be used by end-users work as expected. For this reason, in order to create them, we propose to use the user stories and acceptance tests (artefacts AA2) that describe end-user requirements (explained in Section 5.2.2). For each acceptance test:

- A new target platform test is created.
- The parameter *input* of the target platform test is obtained from the input of the acceptance test. In Figure 6.26, this projection is illustrated by the arrow 1.
- The parameter *assert condition* of the target platform test is obtained from the response of the acceptance test. In Figure 6.26, this projection is illustrated by the arrow 2.
- The generated artefact to be tested by the target platform test is the result of parsing, validating, and applying the code generator to a DSL specification that describes the user story. In Figure 6.26, this projection is illustrated by the arrow 3 and 4.

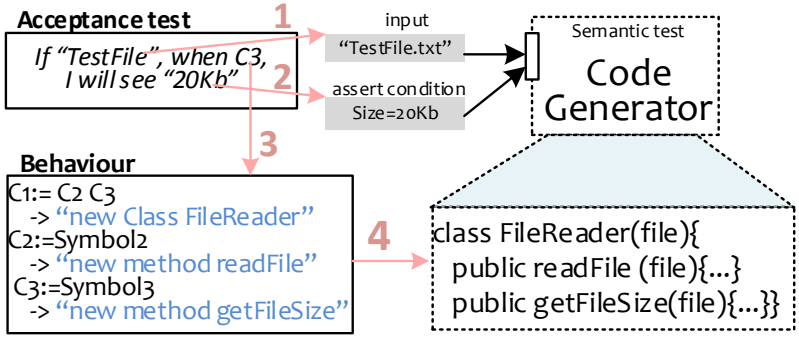


Figure 6.26 Example that illustrates the creation of the code generator tests

6.2.2 Implementation of the DSL infrastructure

After all the tests are created, in the step **DSL infrastructure implementation,** the developers apply both model-driven and test-driven development to implement the complete DSL infrastructure: the parser, the validator and the code generator.

6.2.2.1 The parser

The parser is a program that reads DSL specifications expressed with the concrete syntax and identifies the underlying abstract syntax. When the specification is compliant with the syntax, the parser obtains an equivalent representation such an abstract syntax tree.

In this method, the parser is implemented by applying a model-driven approach using the models crated in the design stage. Specifically, the parser source code is generated automatically by using the abstract syntax metamodel and the concrete syntax grammar (artefacts DA1, Section 6.1.2).

In order to apply this approach, we can use the framework for DSL development Xtext, which takes as input a metamodel specified in the Ecore language [92] and a grammar specified using an EBNF-like syntax [97] and generates automatically the DSL parser in the Java language. Therefore, it is necessary to implement the abstract syntax metamodel using the Ecore language and the concrete syntax grammar using the EBNF-like syntax.

6.2.2.2 The validator

The validator is a program that reads the equivalent representation of a DSL script created by the parser (such as an abstract syntax tree) and checks whether it fulfils all the semantic restrictions of the DSL. The validator is made by a set of validation rules that describe each of these restrictions and the error messages that must be provided when some of these restrictions are violated.

In this method, the validator is implemented by applying test-driven development using the validator tests (artefacts IA1, Section 6.2.1.2). In order to apply TDD, the developers start running all tests. The first time, these tests are expected to fail. Then, the developers choose one of the tests that has failed, program the necessary source code to implement the validation rule, and run the test again. If the test still fails the developers check again the source code. These two last tasks go on until this test succeeds. When the test suceeds, the developers go back to the start and check whether all the tests succeed already or it is still necessary to modify the source code of the validator. Eventually, all tests will succeed simultaneously and all the validator rules will be implemented (Figure 6.27).

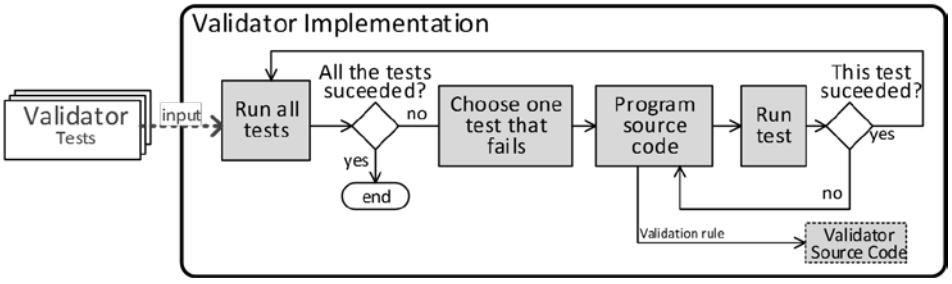


Figure 6.27 Approach to implement the validator

6.2.2.3 The code generator

The code generator is a program that reads the equivalent representation of a DSL script created by the parser and checked by the validator (such as an abstract syntax tree) and obtains the equivalent target platform artefacts that provide the expected behavior. The code generator is made by a set of transformation rules that read the different fragments of the DSL representation (such as fragments

of the abstract syntax tree) and obtain the corresponding source code in the target implementation platform.

In this method, the code generator is implemented by applying both model-driven development using the abstract syntax metamodel and test-driven development using the target platform tests and the semantic templates. The proposed approach includes four sub-steps (Figure 6.28):

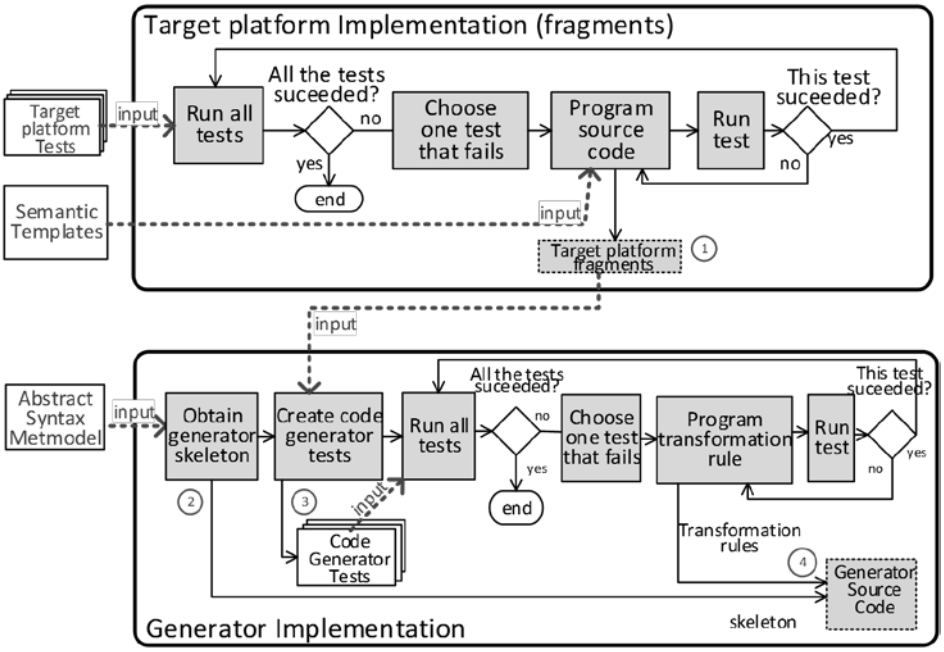


Figure 6.28 Approach to implement the target platform fragments and the code generator

- *Implement target platform fragments*. The aim of this step is to obtain fragments of source code of the target implementation platform that represent a specific behavior. For example, a fragment of a Java class, a fragment of a UML model, or a fragment of a XML file. These fragments are created to be used afterwards to implement the code generator; specifically, to infer the transformation rules. These fragments are implemented by applying test-driven development using the target platform tests (Section 6.2.1.3). The approach to implement them is similar to the validator approach. The developers run all the tests and expect them to fail the first time. Then, they

choose one of the tests that has failed and program the source code fragment that makes the test to succeed. When this test succeeds, the developers go back to the start and check whether all the tests succeed already. Eventually, all tests will succeed simultaneously and all the target platform fragments will be implemented. The main difference with the validator is that when programming source code fragments, the developers use the semantic templates (artefact DA2, section 6.1.4), which describe the details of the technological artefact that is needed to provide the corresponding behavior. Developers know which semantic template to use because they must use the template whose use story matches the user story of the target implementation test that is being addressed.

- *Obtain the generator skeleton*. The skeleton of the generator is the set of classes where developers will embed all the transformation rules to generate the target platform implementation source code. This skeleton is implemented by applying model-driven development using the abstract syntax metamodel that was created in the design stage (artefact DA1, section 6.1.2).
- *Specify code generator tests.* Once all the fragments of source code of the target platform are obtained, the developers are able now to specify the code generator tests that validate that the code generator works as expected. Hence, the developers must follow the guidelines that were explained in Section 6.1.2.
- *Program the transformation rules of the code generator*. The transformation rules of the code generator are implemented by applying test-driven development using the code generator tests that were created in the previous sub-step. The approach to implement them is similar to the validator approach. The developers run all the tests and expect them to fail the first time. Then, they choose one of the tests that has failed and program the transformation rules (embedded into the generator skeleton) that makes the test to succeed. When this test succeeds, the developers go back to the start and check whether all the tests succeed already. Eventually, all tests will succeed simultaneously and the complete set of transformation rules will be implemented.

6.2.3 The implementation of the genetic analysis DSL

In this section, we applied the proposal to implement the DSL infrastructure of the DSL for genetic analysis. The goal of this infrastructure is to provide geneticists with an editor in which they can describe their genetic analysis using the syntax of the DSL (Section 6.1.5). After describing one genetic analysis, the DSL infrastructure must generate an executable workflow that supports the equivalent functionality (the specific genetic analysis) and configures the execution details of the underlying technological software artefacts.

According to the geneticists' feedback, we chose the Galaxy platform as the target execution environment, which is an environment that integrates biological functionality and allows the specification of workflows (further details were provided in Section 6.1.5). The DSL infrastructure generates workflows that are compliant with the Galaxy platform (written using their proprietary workflow syntax). Geneticists can upload the generated workflow into a Galaxy server and execute it with their genetic data files. This way, geneticists will avoid the technological details of designing a genetic analysis workflow such as deciding which software tool to use, configuring each tool parameters, and dealing with the interoperation among the different tools.

In order to implement the DSL infrastructure, we used the framework Xtext. Specifically, Xtext generates three Java projects to deal with the different aspects of the DSL infrastructure:

- diagnosis.it.mysdl: This is the main project. In this project we specified the abstract syntax metamodel, the concrete syntax grammar, the rules of the validator, and the transformation rules of the code generator. This project contains an executable workflow that compiles this and the rest of the projects. After executing this workflow, the complete infrastructure is generated in Java source code.
- diagnosis.it.tests: This project is used to specify the tests that check for the correctness of the complete DSL infrastructure. Specifically, we specified the validator tests and the code generator tests.
- diagnosis.it3.ui: This project is used to specify the aspects of the user interface. Specifically, we specified content assistance, syntax coloring, and quickfixs for the editor.

The implementation approach starts with the step **test specification** (Section 6.2.1). First, we specified the validator tests using the project created by Xtext and the framework for test specification JUnit[10]. For each pair of semantic restriction (created in the design stage) and acceptance test (created in the analysis stage), we specified a JUnit test that checks that when a semantic restriction is violated, an error arises and the error arisen is the correct one.

Figure 6.29 shows the JUnit test that checks if the restriction *It is mandatory to annotate the POLYPHEN prediction before filtering by POLYPHEN* (explained in Figure 6.15) is well implemented in the validator. First, we used the clause @Before to specify an example of DSL specification that violates this restriction. Then, we used the clause @Test to create a test that checks whether the validator is arising the error "You should annotate the prediction before filtering/priotizing by prediction".

```
@Before
def void testSetupOnce() {
DiagnosisPackage.eINSTANCE.eClass();
  diagnosis = parser.parse ('''Diagnose DiabetesMellitus
  Read variations genotypes from a VCF file from input
  Annotate variations with gene
  Filter variations by Sift effect tolerated
  Filter variations by Polyphen effect benign
  Report variations with gene Sift Polyphen ''')
}
@Test
def checkValidationAnnotatePolyphen(){
  diagnosis.assertError(DiagnosisPackage.Literals.EFFECT_F,
  "PredictionNotAnnotated",
  "You should annotate the prediction before filtering/prioritizing by
  prediction")
  diagnosis.assertError(DiagnosisPackage.Literals.PREDICTION_R,
  "PredictionNotAnnotated", "You should annotate the prediction before
  reporting by prediction")
}
```

Figure 6.29 Example of a semantic test that tests the validator

Second, we specified the target platform tests for testing the geneticists' requirements. In this case we used the environment Galaxy to create these tests, since this is the target implementation environment that was chosen by geneticists to run their genetic analyses.

[10] Framework for test specification JUnit http://junit.org/

The Galaxy environment does not support the specification of tests, however, we used the workflow canvas to specify them. We created a workflow for each target platform test. This way, in order to check the correctness of the target platform artefacts we had to run a workflow. In order to specify each test, we created a Galaxy workflow with two inputs files (parameters input and assert condition of the test) and a tool that compares those files (comparator of the test). If the two files are equal, the test will succeed, otherwise it will fail. This test always fails upon their creation because the target platform code (the equivalent Galaxy workflow) is not implemented.

Figure 6.30 shows a Galaxy test for testing the end-user requirement *Filter by Polyphen effect* (described in the user story template Table 5.7). First, we saved the content of the field *input* of the acceptance test in the text file *4VariantsAnnotatedPoly.vcf* and the content of the field *response* in the file *1VariantFilterPoly.vcf*. This is illustrated by the arrows 1 and 2. Then, we created a workflow in Galaxy with two inputs. The first input is the parameter *input* of the target platform test, the second input is the parameter *assert condition*, and the tool Compare Two Datasets is the comparator of the test. The files *4VariantsAnnotatedPoly.vcf* and *1VariantFilterPoly.vcf* are uploaded to Galaxy and introduced in the workflow as the first input and the second input respectively. In summary, this workflow checks how the input *4VariantsAnnotatedPoly.vcf* is processed to obtain a response that is equal to the content of the *file 1VariantFilterPoly.vcf*. At the beginning, this workflow does not run any biological service since these are included in the DSL infrastructure implementation step.

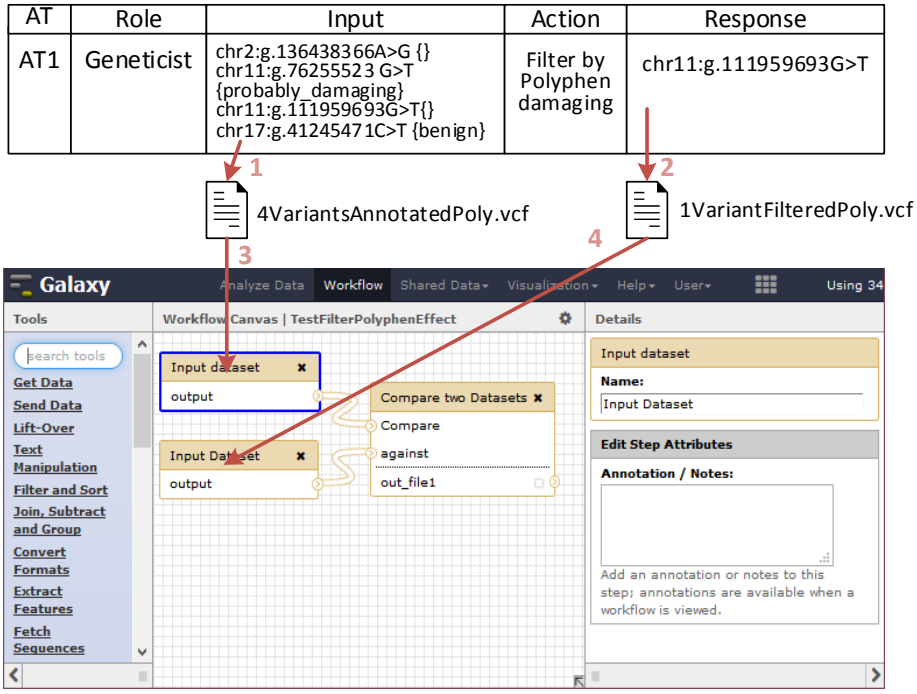


Figure 6.30 Example of a target platform test using Galaxy

Regarding the **implementation of the DSL infrastructure** (Section 6.2.2), we used the Java projects created by Xtext to implement the parser, the validator and the code generator.

In order to implement the parser, we implemented the abstract syntax metamodel (Figure 6.8) in Ecore and the concrete syntax grammar using the EBNF-Like syntax proposed by Xtext. Then, we generated the parser automatically.

In order to implement the validator, we used an Xtend[11] class that was automatically created by Xtext into the java package *validator* of the main project diagnosis.it.mydsl. Xtend is a statically typed programming language sitting on top of Java that, among other uses, can be used to specify different aspects of the DSL easily and more readable than using Java code. In order to generate the complete infrastructure Xtext provides the compilation workflow that generates the equivalent Java code.

[11] http://www.eclipse.org/xtend/

Figure 6.31 shows the Xtend code of the validation rule that checks for the fulfillment of the restriction *It is mandatory to annotate the POLYPHEN prediction before filtering by POLYPHEN* (explained in Figure 6.15). The clause @Check indicates the existence of a validation rule that must be invoked by the validator. The Xtend method *checkAnnotatePredictionBeforeFilteringByPrediction* contains Xtend code that checks the semantic restriction and shows the corresponding error. Specifically, when the DSL specification contains a filter by effect prediction but not the annotation of the effect prediction, the validator shows the error "*You should annotate the prediction before filtering by prediction*" (with the error code *PredictionNotAnnotated)*. After implementing this validation rule correctly, the validator test explained in Figure 6.29 succeeded.

```
@Check
def checkAnnotatePredictionBeforeFilteringByPrediction(Diagnosis
diagnosis){
  for(filter:diagnosis.analyses.filter(PredictionF)){//for each filter
    var annotationFound=0 //Initialize variable->not found
    for(annotation:diagnosis.analyses.filter(Annotation)){ //search
    annotation
      if(annotation.prediction!=null)
        for(annotationPred:annotation.prediction){//check same prediction
        algorithm
          if(annotationPred.algorithm.equals(filter.algorithm)){
            annotationFound=1//Annotation found
          }
        }
    }
    if(annotationFound==0){//Annotation not found->show error
          error('You should annotate the prediction before filtering by
          prediction', filter, null, PredictionNotAnnotated)//Error
          message, construct and error code
    }
  }
}
```

Figure 6.31 Example of Validator method that checks a semantic restriction

Finally, in order to implement the code generator, we followed the four sub-steps explained in Section 6.2.2.3 (Figure 6.28).

The first sub-step is to *Implement target platform fragments.* The goal of this sub-step is to obtain fragments of Galaxy workflows that implement the behavior of each user story. In order to obtain each of these fragments we used the target platform tests specified as Galaxy workflows and we applied TDD. For each of these workflows, we run the workflow. Each of the workflows failed the first time

because the behavior was not implemented. Then, we completed the workflow by adding biological services to the workflow until it succeeded. In order to know which biological services we had to add to the workflow and how to configure them, we used the semantic templates (artefact DA2, explained in Section 6.1.4). When each workflow succeeded, we exported the created workflow to a text file. This way, we obtained the fragment of the Galaxy workflow that implements the specific behavior of the user story. Eventually, these fragments will be used to infer the transformation rules of the code generator.

Figure 6.32 shows the Galaxy workflow that implements the user story *Filter Variations by predicted effect Polyphen*. First, we selected the corresponding acceptance test of the user story template (Table 5.7) and the corresponding semantic template (Table 6.5). Specifically, we included the Galaxy tool *Ensembl Filter VEP* and configured the parameters *input* as the input of the workflow and the parameter *filter_criteria* as the string *POLYPHEN is probably_damaging*.

User Story	Filter Variations by predicted effect POLYPHEN			
Service Identifier	Ensembl Filter VEP			
Source description	Galaxy			
Inputs	Description	Type	Constant	Value
Input	File that gathers the variations	DataFile (VCF)	False	-
FilterCriteria	Evaluation expression that indicates the polyphen criteria to filter	String	False	Examples: "Polyphen is benign" "Polyphen is possibly damaging"
Outputs	Description	Type		Visibility
annotated_vcf	File that gathers the annotated variations	DataFile (VCF)		True

Role	Input	Action	Response
Geneticist	**chr2**:g.136438366A>G {} **chr11**:g.76255523 G>T {probably_damaging} **chr11**:g.111959693G>T{} **chr17**:g.41245471C>T {benign}	Filter by Polyphen damaging	chr11:g.111959693G>T

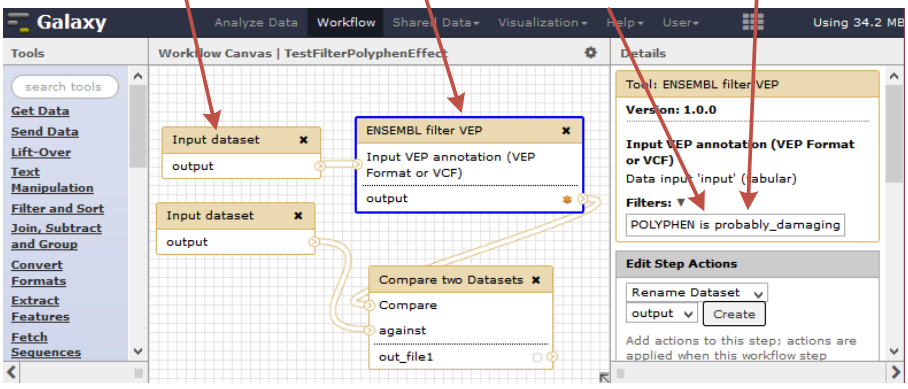


Figure 6.32 Galaxy workflow to pass the test *TestFilterByPolyphen*

When the Galaxy test succeeded (the two files under comparison were equal), we exported the workflow and obtained the corresponding Galaxy workflow fragment. Figure 6.33 shows a simplified fragment of the Galaxy workflow that implements the user story *Filter Variations by predicted effect POLYPHEN*. From this fragment, we are mostly interested in the attributes *tool_id*, which indicates the identifier of the tool, and *tool_stage*, which configures the parameters of the tool. Specifically, in this fragment the attribute *tool_id* is the Galaxy tool *filter_vep* and the parameters of the field *tool_stage* are *filterField* and *input*. Eventually, this fragment will be used to infer the transformation rule of the code generator that implements the behavior of the user story *Filter Variations by predicted effect POLYPHEN*.

```
"galaxy_workflow_step": {
  …
  "tool_id": "filter_vep",
  "tool_state": "{
      \"input\": \"input1\",
      \"filterField\": \"\\\"POLYPHEN is probably_damaging \\\"\"
  }",
  …
}
```

Figure 6.33 Simplified fragment of a Galaxy workflow

The second sub-step is *Obtain the generator skeleton*. In order to create the code generator, we also used Xtend classes. Xtext automatically generates an Xtend class in the main project *diagnosis.it.mydsl* in which developers can implement the code generator. However, instead of programming all the generator rules in this Xtend class, we designed a different structure based on the entities of the abstract syntax metamodel. We created four packages: patientdata, analysis, report, and galaxy. The first three packages contain the classes generated according to the structure of the abstract syntax metamodel. The fourth contains two Xtend classes that deal with the specific details while creating Galaxy workflows. Figure 6.34 shows the complete skeleton of the code generator.

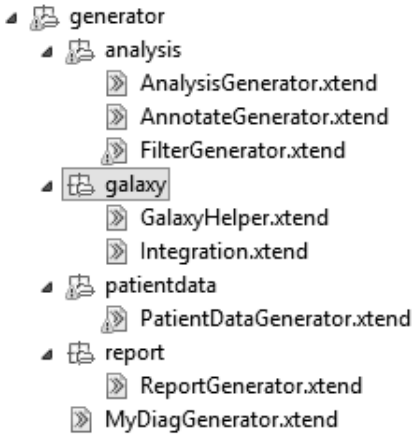


Figure 6.34 Xtend classes that represent the Generator skeleton

The third sub-step is *Specify code generator tests*. After obtaining all the different fragments of Galaxy workflows in the first sub-step, it was then possible to create the code generator tests. Equally to validator tests, we used Xtext and JUnit to specify generator tests. For each trio of semantic template (and corresponding Galaxy workflow fragment), user story and acceptance test, we specified a JUnit test. The *input* of this test is the DSL construct described in the acceptance test and the *assert condition* is the fragment of the Galaxy workflow associated with the corresponding semantic template.

Figure 6.35 shows the JUnit test *testFilterPolyphenEffect*(), which tests the behavior of the code generator in relation to the user story *Filter Variations by Polyphen effect*. First, in the clause @Before, we wrote the usage scenario *Diabetes Mellitus Type 2 (Analysis 1)* using the DSL syntax and we run the parser and the generator. Then, we used the clause @Test to check whether the fragment generated is equal to the fragment of the parameter *assert condition*. If they are different, the test shows an error message. In order to compare both fragments, we created a method that obtains a *Galaxy workflow*, a *Galaxy workflow fragment* and a *toolId* and checks in the complete workflow if the fragment corresponding to this *toolId* is equal to the fragment saved in the file.

```
@Before
def void testSetupOnce() {
    DiagnosisPackage.eINSTANCE.eClass();
    diagnosis = parser.parse (
        '"Diagnose Diabetes Mellitus Type 2 (Analysis 1)
        Read Variations genotypes from VCF file Patient1.vcf
        Annotate Variations with gene, transcripts, polyphen
        Filter Variations by genes {ABCC8, CAPN10, KCNJ11, GCGR,
        SLC2A2, HNF4A, INS, INSR, PPARG, TCFI2, ADIPOQ, AKT2, PAX4,
        MAPK81p1, GPD2, MNTR1B}
        Filter Variations by predicted effect polyphen damaging
        Report Variations with gene, predicted_effect"')
    fsa= new InMemoryFileSystemAccess()
    generator.doGenerate(diagnosis.eResource, fsa)
    filecontent=fsa.getTextFiles().values().iterator().next().toString();
}
@Test
def test FilterPolyphenEffect(){
     assert.assertTrue("The workflow fragment of filterByPolyphenEffect and
    the generated one are different",
    GalaxyGenerator.checkGeneratorGalaxy(filecontent,
    "Galaxy_Fragment_PolyphenEffect.txt", "filter_vep"))
}
```

Figure 6.35 Example of a JUnit test that checks the correctness of the generator

The fourth sub-step is *Program the transformation rules of the code generator*. The goal of this sub-step is to implement the rules that transform the abstract syntax representation (such as an abstract syntax tree) that is created by the parser into a Galaxy workflow. These transformation rules were placed inside the methods of the Xtend classes that were created in the second sub-step (code generator skeleton). For each generator test, we identified the corresponding Xtend class and we programmed the transformation rule until the generator generator was able to transform the abstract syntax tree from the parameter *input* into the Galaxy workflow fragment from the parameter *assert condition*.

Figure 6.36 shows the transformation rule that generates code to fulfil the user story *Filter Variations by Polyphen effect*. This transformation rules generates a fragment of a Galaxy workflow that executes the tool "filter_vep" with the parameters indicated by the entity *PredictionF*. This transformation rule is used to generate the fragment of a Galaxy workflow shown in Figure 6.33 and makes the test *FilterPolyphenEffect* of Figure 6.35 to succeed.

```
def dispatch filterVariations(PredictionF filter)'''«filterWithVEP(filter)»'''
def filterWithVEP(Filter filter)'''
 "«step»": {
...
"tool_id": "filter_vep",
"tool_state": "{
    \"filterField\": \"\\\"«expressionVEP(filter)»\\\"\",
    \"input\": \"input1\
 "}",
   ...
  }'''
def dispatch expressionVEP(EffectF prediction)'''«
  FOR effect:prediction.effect SEPARATOR ' or ' »«
  » POLYPHEN is «effect.vepEffect»«
  »«ENDFOR»'''
```

Figure 6.36 Example of transformation rule of the generator

After finishing the implementation of the parser, the validator and the code generator, we compiled all the Xtext projects and run the implemented DSL infrastructure. Figure 6.37 shows an Eclipse-based interface in which the geneticists can create their genetic analysis pipelines using the DSL. They write the pipeline using a textual editor that understand the DSL syntax and when they save this file, the DSL infrastructure parses the files, applies the validation rules, and generates the corresponding workflow.

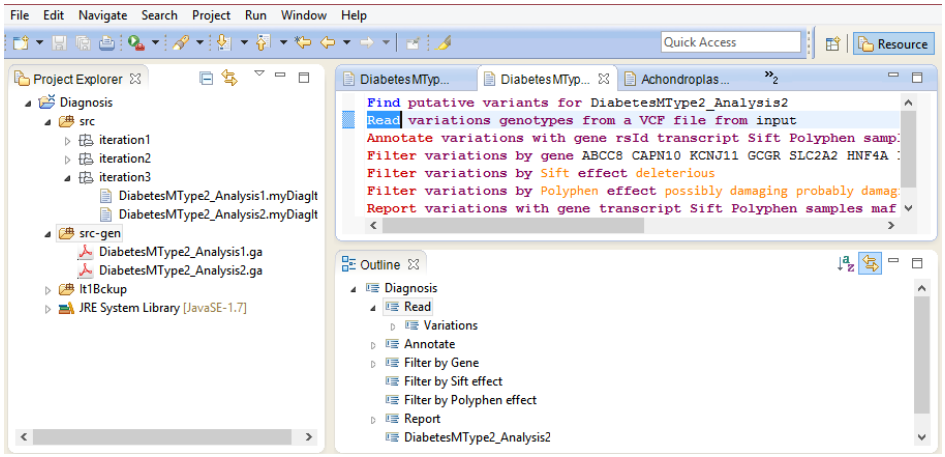


Figure 6.37 Interface for using the DSL infrastructure

The complete code is available in the following repository URL: https://github.com/mvillanueva/GeneticAnalysisDSL

6.3 Conclusion

In this chapter, we have explained the stages Design and Implementation. We have explained these two stages together because both of them aim making the DSL a reality. In the Design stage, it is planned how the DSL is going to be developed, and then, in the implementation stage, this plan is followed in order to provide a technological support for the DSL.

Regarding the Design stage, our approach contributes to the state of the art by adopting agile practices to involve end-users in the definition of the design models. In the syntax design, as mechanism M2, we have proposed a questionnaire that allows the end-users to choose the concrete syntax that is most suitable and that shows them a set of domain examples so they can contribute in the concrete syntax grammar and in the abstract syntax metamodel. In the semantics design, as mechanism M3, we have adapted an existing template for describing services to facilitate to the end-users the description of the behavioral semantics of the DSL.

However, as drawbacks, the method does not support the definition of internal or graphical DSLs yet. Also, the approach to define the behavioral semantics requires the domain to have pre-existing executable services that encapsulate different domain functionalities and the end-users to know these executable services. This was the case of our illustrative example thanks to the Galaxy environment, which provided a set of biological services that could be used to specify the semantics of the DSL.

Regarding the Implementation stage, our approach contributes to the state of the art by combining a model-driven development approach (MDD) with a test-driven development approach (TDD) to implement the different artefacts of the DSL infrastructure. First, we have defined the set of different tests that must be created to apply TDD. Then, we have explained which artefacts can be generated automatically using MDD (which is not a contribution of this PhD) and how to generate the rest using the tests that have been defined. Although, at the moment, it is still not possible to automate generation of the complete DSL infrastructure (parser, validator, code generator) from the design models, TDD allows the developers to systematize the implementation of the artefacts that cannot be generated automatically applying model transformations.

7. Releasing the Solution: The Testing, Deployment and Maintenance stages

Once the developers have implemented the DSL infrastructure of the iteration, the next step is releasing the solution to the end-users to check whether it is appropriate to satisfy their needs. First, the end-users check whether the different aspects of the current DSL are correct: abstract syntax, concrete syntax, semantic restrictions, and behavioral semantics. During this experience, developers gather all the feedback provided by the end-users with the aim to improve the DSL in the next iterations. In the method proposed in this PhD, this assessment is done in the Testing stage.

Second, when a version of the DSL infrastructure has been tested and the end-users consider this version suitable enough for using it, the developers prepare a production environment so that the end-users can use the current version of the DSL infrastructure by themselves. In the method proposed in this PhD, this release is prepared in the Deployment stage.

Third, after the end-users have used the DSL infrastructure for a long enough period, they provide their feedback about the DSL released. In the method proposed in this PhD, this assessment is done in the Maintenance stage.

In this chapter, we explain the Testing, Deployment, and Maintenance stages and how we applied each of them for developing of a DSL for the genetic analysis domain. We created several versions of the method and the DSL; however, in order to simplify the explanation of the method, we focus only on the method version that corresponds to the last iteration. Similarly, in order to simplify the explanation of the application of the method to build the DSL, we only provide fragments of the DSL in regards to the illustrative example presented in Chapter 4.

7.1 The Testing stage

The goal of the *Testing* stage is to assess if the DSL infrastructure implemented in the iteration fulfils end-users' requirements and needs. In order to accomplish this goal, this stage is divided into two steps (Table 7.1): 5.1) Demonstrating the DSL infrastructure of the iteration to end-users; and 5.2) testing that DSL infrastructure.

Table 7.1 Overview of the testing stage

Step	Step Description	Artefact	Mechanism for gathering end-user input
5.1	Demonstration of the DSL infrastructure of the iteration	Demonstration (TA1)	The definition of done and a demonstration based on usage scenarios (mechanism M4)
5.2	DSL infrastructure testing	Iteration Feedback report (TA2)	A questionnaire and a set of activities based on usage scenarios (mechanism M5)

7.1.1 Demonstration

In the **demonstration step**, the developers compose a functional DSL infrastructure with the requirements addressed in the iteration and demonstrate it to the end-users. In order to compose this release, **as mechanism M4**, we adopted the agile practices *the definition of done* and the *customer demo* from Scrum.

According to the *definition of done* practice, the developers inspect each requirement one by one assessing which are done, which are not, and which are the problems found that explain the lack of completeness of the iteration. In order to classify a user story as done, we propose the acceptance criteria to be: *a user story is considered done when all their acceptance tests can be written in the DSL infrastructure and the generated executable code obtains the result described in the acceptance test*.

According to the *customer demo* practice, in order to show the end-users the current state of development, we propose to perform a live demo of the DSL release to specify one usage scenario from the analysis. This demonstration will consist on:

1. Showing the description of the usage scenario to be demonstrated;
2. Using the DSL editor to create the corresponding DSL specification with the DSL syntax.
3. Executing the code generator that translates the DSL specification into the target platform.
4. Showing end-users the execution generated by the code generator.

Before the demonstration, the end-users are provided with a summary of the demonstration and they are encouraged to write down their impressions to be discussed after. When the demonstration is finished, the developers and the end-users discuss the written impressions and then, new requirements, changes, and comments are added to the DSL backlog.

7.1.2 DSL infrastructure testing

Once the DSL release has been demonstrated, in the **DSL infrastructure testing step**, the end-users have the opportunity to use the DSL by themselves. The goal of this activity is that the end-users assess whether the different aspects of the DSL (abstract syntax, concrete syntax, semantic restrictions, and semantic behavior) are well designed and well implemented.

In order to guide the end-users in the testing of the different aspects of the DSL, as **mechanism M5,** we propose a questionnaire that asks about three dimensions proposed by Visser [86] to assess a DSL: expressivity, coverage, and completeness.

- Expressivity: This dimension analyses if the language abstractions support a concise expression of the domain. This dimension can be applied to assess the syntax and the semantic restrictions of the DSL.
- Coverage: This dimension analyses if the abstractions of the language are adequate for developing applications in the domain. This dimension can be applied to assess the correctness of DSL requirements and syntax.
- Completeness: This dimension analyses if the language implementation creates a complete target implementation or is it necessary to write additional code. This dimension can be applied to assess the behavioral semantics.

In order to facilitate the participation of end-users in the assessment of the DSL regarding those dimensions, we created a questionnaire to be answered by end-users. Table 7.2 shows the set of questions proposed to assess each aspect of the DSL related with the analyzed dimensions.

Table 7.2 Questions to test different DSL aspects

Questions for testing Requirements	
Coverage	Did you find any erroneous step/instruction?
	Did you find in the language any step that contains come erroneous aspect?
	Did you miss any essential step/instruction?
Questions for testing Syntax	
Expressivity	Would you add, change, remove or reorder any word of the language?
	Is the language easy to understand?
	Is the language intuitive to use?
Coverage	Did you find a combination of words that were incorrect but they could be written with the DSL?
Questions for testing Semantic restrictions	
Expressivity	Did you find any error message that you did not understand?
Coverage	Did you find a combination of constructs that were incorrect but they could by written with the DSL?
	Did you find any step that was dependent of another one but it could be written without satisfying that dependency?
Questions for testing Behavioral Semantics	
Completeness	Do you know any new software that suits better to implement a step/instruction?
	Did you find any error after executing the generated artefact?

While assessing the DSL, the end-users are free to use it. However, in order to ensure that all the aspects of the DSL are assessed, we propose a set of activities that will guide them to incrementally learn how to use the DSL editor and will help in the assessment of all the DSL aspects: abstract syntax, concrete syntax, semantic restrictions, and semantic behavior. The activities proposed are:

1. Write the same usage scenario of the demonstration: The end-users already know the DSL syntax of the usage scenario, so they only have to write it again using the DSL infrastructure. This activity is proposed to facilitate the end-users the familiarization with the DSL syntax and the DSL editor.
2. Write another usage scenario: End-users choose another usage scenario and use the DSL editor to write the corresponding specification. They are provided with all the usage scenarios written in natural language, so the task is learning how to express them using the DSL syntax. To

facilitate the task, they are provided with the DSL syntax specification (the grammar). In addition, the DSL editor provides code completion and on-live syntax checking in order to aid the end-users to wirite the DSL specification.

3. Test the technological implementation generated by the code generator: After the end-users finish the usage scenario specification using the DSL infrastructure, the code generator generates the corresponding technological implementation. After that, the end-users test whether the generated artefacts fulfill their requirements. If the target artefacts are executable, the end-users can run them and test their behavior.

After answering the questionnaire, the developers process them to create the iteration feedback report (TA2), which is used to update the product backlog with new requirements and changes for the next iterations. Then, end-users and developers overview together the changes to be adopted in the next iteration.

7.1.3 The testing of the genetic analysis DSL release

In this stage, we collaborated with geneticists to test the current state of the implementation of the DSL for supporting genetic analysis.

For the **demonstration** (Section 7.1.1), we created a video to show to the geneticists the current state of the DSL (mechanism M4). This video presented the following contents:

1. Demonstration of one usage scenario: The video showed step by step how to write in the DSL editor the usage scenario "*Diabetes Mellitus Type 2 (Analysis 1)*" (Table 5.9).
2. Description of editor help and shortcuts: The video showed the syntax highlight feature and a set of shortcuts. Figure 7.1 shows the keywords "Read" or "Annotate" colored in red, which indicate the main keywords of the DSL constructs. The figure also shows the shortcut "Filter- Add complete instruction", which writes a template that contains the different elements of the "Filter" construct.

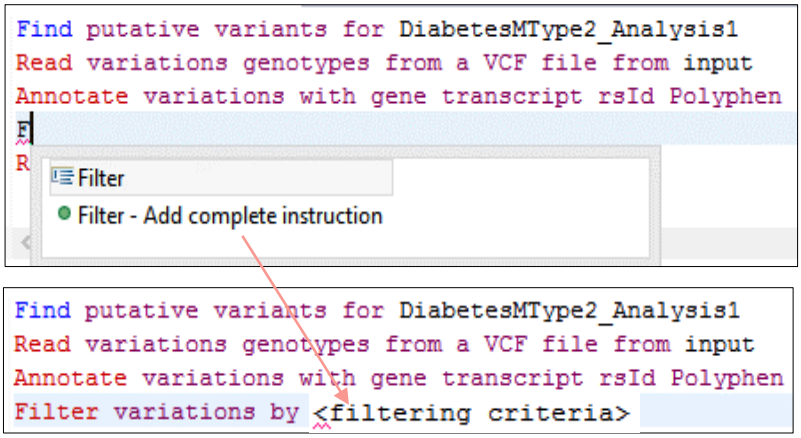


Figure 7.1 Example of a DSL syntax shortcut

3. Explanation of error messages retrieved by the DSL infrastructure: The video showed the errors that appear when the DSL syntax is not correctly used according to the DSL syntax or a restriction is violated. Figure 7.2 shows the error messages that appear when the restriction "*Annotate POLYPHEN effect before filter by POLYPHEN effect*" is violated.
4. Generation of the Galaxy workflow: The demonstration showed how to generate a functional equivalent Galaxy workflow and how to deploy it. Figure 7.3 shows how the pipeline "*Diabetes Mellitus Type 2 (Analysis1)*" is automatically generated and saved into the source folder and how the generated workflow can be deployed in Galaxy.

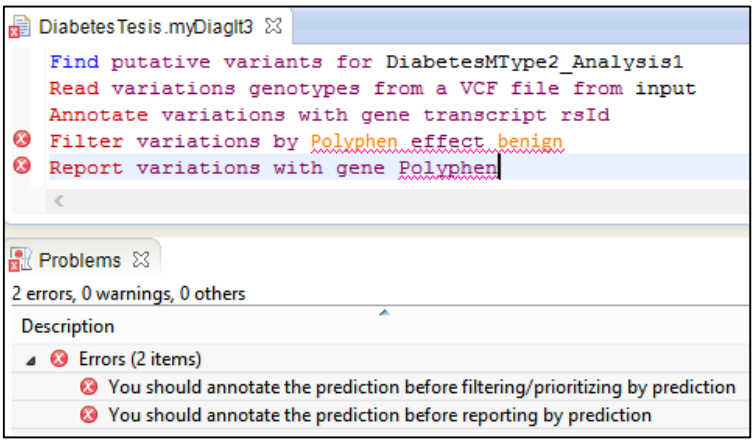


Figure 7.2 Example of several DSL error messages

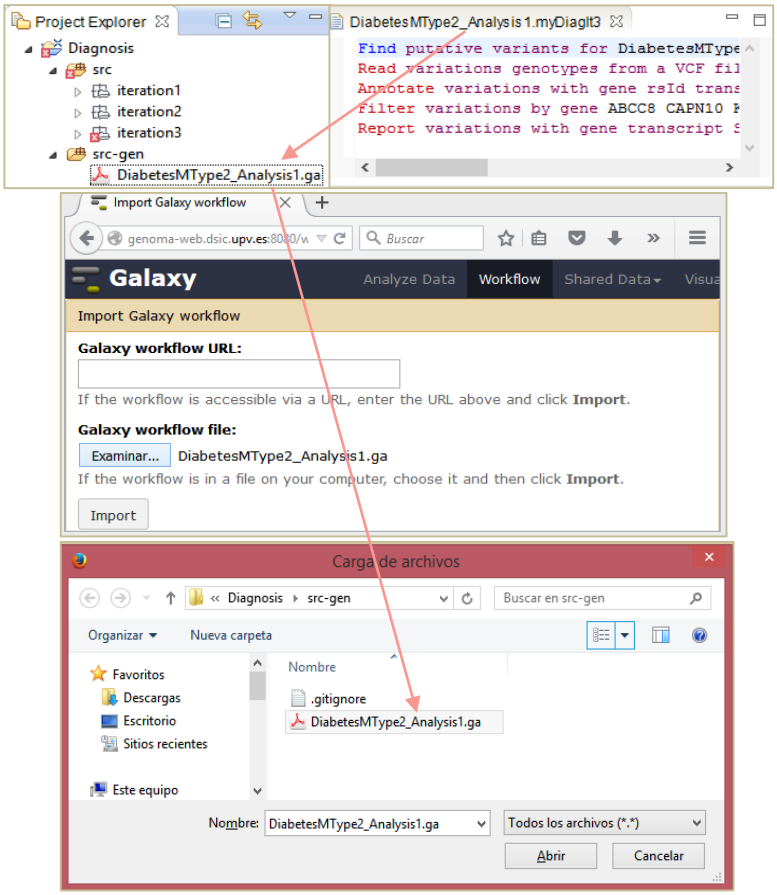


Figure 7.3 Generation and deployment of a Galaxy workflow.

After the demonstration, in the **DSL infrastructure testing step** (Section 7.1.2), the geneticists executed a set of activities with the DSL infrastructure and answered the testing questionnaire (mechanism M5). The activities were the following:

1. Writing the usage scenario of the demonstration: The developers provided the example "*Diabetes Mellitus Type 2 (Analysis 1)*", which was already written with the DSL syntax (Figure 4.8), and asked the geneticists to repeat on their own the same process that was shown in the demonstration. Geneticists wrote the same scenario in the DSL editor, generated the Galaxy workflow, and checked for the workflow correctness.

2. Write another usage scenario: The developers provided the description of the complete syntax of the DSL and all the usage scenarios of the iteration written in natural language (as they were specified in the Analysis stage). The geneticists checked the syntax, used the shortcuts that were shown in the demonstration, and resolved the syntax errors that appeared during the specification of the genetic analysis. When the usage scenario was finished and it had no errors, the geneticists generated the corresponding Galaxy workflow, imported it into Galaxy and execute it to check for its correctness.

After finishing these activities, the geneticists answered the testing questionnaire. Table 7.3 shows a question that asked geneticists about the coverage of the DSL.

Table 7.3 Example of the geneticists' responses to the testing questionnaire

Did you missed any essential step/instruction? How important is it for the usage scenario?		
Geneticist 1	Geneticist 2	Geneticist 3
"I missed an instruction about how to write the Sift score"	"Filter instructions by gene should let import a gene list from a file (e.g. what happens if I want to filter by 200 genes?)	"Always"

7.2 The Deployment stage

The goal of the Deployment stage is to release a stable DSL to be freely used by end-users, i.e, without any sort of developer's supervision. In order to accomplish this goal, this stage consist of one step (Table 7.4): 6.1) Installation of the DSL release.

Table 7.4 Overview of the Deployment stage

Step	**Step Description**	**Artefact**	**Model-based transformations guidelines**	**Mechanism for gathering end-user input**
6.1	DSL release installation	DSL release	-	-

In the **DSL release installation step**, the developers release the current state of the DSL to be used by the end-users in their own environment. Examples of deployments can be, for instance, a server and a web interface that offers a DSL editor or a local installation of the DSL in the end-users' workspace.

It is worth to notice that this step is not performed in each iteration of the cycle. The reason for this decision is to avoid end-users to use DSL releases that could contain unsolved errors. For this reason, when a DSL is stable enough and it is considered ready to use, the DSL infrastructure is deployed to be used by end-users freely. The goal of this release is to find errors in a working scenario that cannot be detected in the testing stage.

End-users do not participate in this stage for two reasons: 1) because installing the DSL does not require end-users' feedback; and 2) because installing the DSL infrastructure could require some technical knowledge.

In order to deploy the DSL for genetic analysis, since we implemented the DSL using Xtext, we can use the feature of the Eclipse Environment "Export"-> "Deployable plugins and fragments". As a result, we obtain a folder that contains the set of plugins that support the DSL infrastructure. This way, the generated plugins can be installed in a local Eclipse instance.

At the moment, we did not deliver this executable to the geneticists. The two first iterations were more focused on assessing the method than developing the genetic DSL. For this reason, the third iteration was the first iteration in which a higher number of requirements were addressed and the resulting DSL addressed more complex genetic analyses. Still, the DSL implemented in this iteration was not mature enough. As a consequence, the geneticists were not provided with the DSL for their use. This means that we did not carried out the Deployment stage for this use case.

7.3 The maintenance stage

The goal of the Maintenance stage is to gather information about the DSL after being freely assessed by the end-users. In order to accomplish this goal, this stage consist of the step 7.1) testing the DSL release.

Table 7.5 Overview of the Maintenance stage

Step	Step Description	Artefact	Model-based transformations guidelines	Mechanism for gathering end-user input
7.1	DSL release testing	Iteration Feedback report (TA2)	-	Mechanism M5:A questionnaire and a set of activities based on usage scenarios

Once the DSL has been made accessible to the end-users and they have used it for a while, in the **testing of the DSL release step**, the end-users provide feedback about the DSL release. The goal is both to assess the different aspects of the DSL and gather information about their preferences or new requirements. As a way to provide this feedback, the end-users answer a similar questionnaire to the questionnaire of the testing stage but with additional questions about their experience while using the DSL. Table 7.6 shows these questions classified by the DSL aspect to assess and the assessment dimension.

Table 7.6 Complementary questions to test different DSL aspects after deployment

DSL aspect	Dimension	Question
Requirement	Coverage	Were you able to express every application that you needed in your domain?
		Was there any step/instruction you never used for expressing your applications?
Syntax	Expressivity	Did you notice any improvement of using the DSL instead of your previous approach?
Semantic restrictions	Expressivity	Did you find any abnormal error message while using the DSL?
Behavioral Semantics	Completeness	Were all the generated applications working properly?

At the moment, we did not release the DSL to the geneticists for their free use. This means that we neither carried out the Maintenance in practice.

7.4 Conclusion

In this chapter, we have explained the three last stages of DSL development: Testing, Deployment, and Maintenance. We have explained these three stages together because the three of them aim to release the DSL to the end-users. In these stages, the end-users try the implemented DSL infrastructure and developers make this assessment possible. The Testing stage comes first because in this stage the DSL is tested by the end-users under a controlled environment. In the Deployment stage, the developers transfer this DSL into the real context of the end-users so they can try it under real conditions. In the Maintenance stage, the end-users provide feedback about their continuous experience using the DSL.

Regarding the Testing stage, our approach contributes to the state of the art by adopting agile practices to facilitate the testing of the DSL release by the end-users. As mechanism M4, we have proposed that developers demonstrate one usage scenario of the DSL to the end-users. As mechanism M5, we have proposed a questionnaire to guide the end-users in the assessment of the different aspects of the DSL.

As drawbacks, although preparing a demonstration of the DSL can be time consuming for developers, the effort is worth because end-users have the opportunity of learning how to use the DSL by example and ask questions during the demonstration.

Regarding the Deployment and Maintenance stages, our approach contributes by proposing a questionnaire to guide the end-users to provide feedback about the DSL, as it was made during the Testing stage. As drawbacks, we did not apply these stages in practice because we did only carried out three iterations and the DSL release was not mature enough to be deployed and used in real conditions. In conclusion, in the near future work, a further analysis should be made for these two stages in order to become part of the stable design of the method.

8. Validation

Empirical assessment of a software development product is essential to ensure that a new product can be really used for what it was originally designed. In the context of software engineering, the same applies to new methods and processes. For this reason, we have conducted an experiment to validate whether the mechanisms of the method proposed in this thesis can be used to involve end-users in DSL development.

In order to carry this validation, we have carried out an experiment of the type "Researching expert opinion" [25] and applied the method proposed to involve geneticists in the design of a DSL for supporting genetic analysis. We have selected the genetic analysis domain; first, due to our close collaboration with research groups involved in genetic analysis; and second, because geneticists usually lack a solid background in computer science and they are good candidates for validation purposes.

This chapter starts explaining the methodology used to conduct the experiment. Then, we explain the motivation of the experiment (scoping), the experimental design (planning), and the execution of the experiment (operation). After conducting the experiment, we analyze the results obtained from the experiment (data analysis), and we provide a critical discussion about those results

(interpretation). Finally, we discuss the final conclusions obtained from this empirical assessment activity.

8.1 Experiment methodology

In order to validate the mechanisms of the method for involving end-users we have carried out a controlled qualitative experiment with geneticists in the development of a DSL for supporting genetic analysis. According to the classification of Wieringa [25], we carried out an experiment of the type "Researching expert opinion". This type of evaluation tests artefacts using experts of the area but does not involve statistical analysis with the aim of estimating significant data; rather, it is an attempt to get early information from real users.

Specifically, our experiment researches geneticists' opinion about the mechanisms of the method while applying the method to develop a DSL for supporting genetic analysis. We chose the genetic analysis domain because the complexity of this domain requires the participation of geneticists to ensure the comprehension of DSL requirements by developers. Moreover, since geneticists do not usually have software development knowledge, we can check how regular users understand the proposed mechanisms. Both reasons justify this scenario as being suitable for applying our method.

The participants of the experiment were three geneticists from the INCLIVA. We had access to these geneticists thanks to previous research collaboration between our institutions.

The experiment was designed according to the guidelines for empirical research proposed by Wohlin [98] and Juristo et al. [99], which are two widely accepted evaluation frameworks in the Software Engineering community that describe the general structure to scope, plan, design, and conduct any kind of experiment in Software Engineering.

8.2 Goal

The goal of this experiment is to assess whether the **mechanisms (M1-M5)** proposed in the method are suitable to involve end-users in the DSL development

process. Our aim is to assess these mechanisms from both the end-users' and the developers' perspective to eventually understand whether these mechanisms can be used to collect the end-users' domain knowledge and their preferences about the DSL. Additionally, we are interested in knowing how long it takes to apply these mechanisms.

The experiment includes a real implementation of a DSL from scratch. The developers apply all the stages and steps of the method, create all the artefacts of the method, and apply the different mechanisms to involve the end-users. However, data measurement, data analysis, and the extraction of conclusions are only focused on the mechanisms for involving end-users. The assessment of the suitability and benefits of the stages, steps, and artefacts of the method (AA1, AA2, AA3, DA1, DA2, IA1, IA2, TA1, and TA2 in Figure 4.4) is outside the scope of this experiment.

8.3 Experimental subjects

Two types of subjects participated in this experiment: end-users and developers.

For end-users, the population that we wanted to test is a set of domain experts (who did not have any DSL development knowledge) of a complex domain that needed a DSL. For the recruitment process, we used "*convenience sampling*" [100], which chooses the subjects that are the easiest to recruit instead of applying a random selection of subjects among the population. We recruited three geneticists that we already knew from a previous collaboration.

The three geneticists had experience in DNA analysis from 1 to more than 10 years, but they did not have experience in DSL development (Table 8.1). It is worth mentioning that despite the existing collaboration with our research center, these geneticists did not have any knowledge about the proposed method since the previous collaboration had been performed in a different context.

Table 8.1 Subjects' profiles

	Genetic experience	DSL experience	Method knowledge
Geneticist G1	1 year	0 years	None
Geneticist G2	3 years	0 years	None
Geneticist G3	More than 10 years	0 years	None

In our experiment, the subjects are indeed representative of the population even though the sample size of this experiment is small. All of them are geneticists: 1) with genetic analysis experience; 2) currently working in the genetic field; 3) with knowledge about the most common technologies used by the genetics community; 4) with software development problems that could be solved with a DSL; and 5) without prior DSL development experience.

For developers, we recruited one developer who had knowledge in DSL development, extensive knowledge about the method, and basic notions of genetics. This subject played both the developer role and the experimenter role.

8.4 Research questions and hypothesis formulation

In order to determine whether the method facilitates end-user participation, we analyzed the satisfaction of end-users and developers as well as the time spent using the proposed mechanisms. We used the definition of satisfaction from IEEE: the contentedness and positive attitudes towards product use [101]. For end-users, we operationalized satisfaction as how at ease the end-users are while using the mechanisms of the method to provide feedback. For the developers, we operationalized satisfaction as how at ease the developers are while using the mechanisms of the method to gather the end-users' feedback and to represent it within the DSL.

To study satisfaction and time, we proposed the following research questions:

RQ1. Are end-users satisfied with the feedback provided through the involving mechanisms? The hypothesis to check in order to answer RQ1 is H_1: End-users are satisfied with the involving mechanisms of the method.

RQ2. Are developers satisfied with the feedback gathered through the involving mechanism to build the DSL? The hypothesis to check in order to answer RQ2 is H_2: Developers are satisfied with the involving mechanisms of the method.

RQ3. How long does the application of the mechanisms for involving end-users take? There is no hypothesis for this research question since we only aim to know the time needed to apply the five mechanisms.

8.5 Factors and treatments

To check the above hypotheses, we used the term *factors* to describe the data that is predefined and controlled and the term *treatments* to describe all the concrete values that the factor takes in the experiment.

In this experiment, we have one factor and one treatment. The **factor** was *the method to involve end-users in DSL development*. The **treatment** was *the set of mechanisms* that have been proposed to involve end-users in DSL development *(*Chapters 5, 6, and 7*)*: M1) User stories, acceptance tests, and usage scenarios; M2) a syntax questionnaire based on usage scenarios; M3) semantic stories based on user stories; M4) a demonstration; and M5) a testing questionnaire. The reason for not having another treatment as a control case for comparison is that the goal of this experiment is to validate the proposal at the conceptual level. Our goal is to elicit the opinions of the experts about the mechanisms of the method and use their feedback to check their usability, their limitations, and their potential improvements.

8.6 Response variables and metrics

Response variables describe the feature to be measured in the experiment from which conclusions are drawn. Table 8.2 summarizes the response variables, metrics, and measurement procedures that were used in the experiment to gather data about the research questions.

Table 8.2 Summary of RQs, hypotheses, and response variables

RQs	Hypotheses	Response Variables	Metric	Measurement procedure
RQ1	H_1	End-users' Satisfaction	PEOU and PU	Satisfaction Questionnaire
RQ2	H_2	Developers' Satisfaction	Comprehension questions, degree of agreement, and undetected errors.	Observation, recording, and analysis of subjects' feedback and anecdotes.
RQ3	-	Time	Minutes	Measurement of time spent

RQ1 requires a response variable to measure the satisfaction of end-users. To measure this satisfaction, we followed the method evaluation model (MEM) proposed by Moody [102], which proposes a framework to evaluate model quality in terms of the following metrics:

- Perceived Ease of Use (PEOU): This is the degree to which a person believes that using a particular method would be free of effort.
- Perceived Usefulness (PU): This is the degree to which a person believes that the intended objectives will be achieved by using a particular method.
- Intention to use (ITU): This is the extent to which a person intends to use a particular method.

In the context of our experiment, we adopted only the PEOU and PU metrics. We discarded the ITU metric because end-users are not responsible for deciding whether the method and the mechanisms are going to be used.

Following the MEM framework, the procedure to measure end-user satisfaction was a questionnaire with a 5-point scale (with the levels strongly disagree, disagree, neutral, agree, and strongly agree). The questionnaire had three questions for assessing the PEOU metric and two questions for assessing the PU metric. Following the advice of Moody, half of the questions were defined in a negative way to stimulate the attention of subjects. Also, all the questions were ordered randomly to avoid boredom.

In total, we created five satisfaction questionnaires, one for each mechanism. All of them contained the same number of questions and asked about the same aspects (3 questions for PEOU and 2 for PU). However, the questions of each of questionnaire were customized for the mechanism that they were assessing:

- For the assessment of the perceived Ease of Use:
    - I found difficult to apply <mechanism>(negative redaction)

    - Applying <mechanism> took me an adequate amount of time (positive redaction)
    - Overall, I found all the process activities clear and easy to understand (positive redaction)
- For the assessment of the perceived Usefulness:
    - I found useless applying <mechanism> (negative redaction)
    - Overall, I found that the process activities engaged my participation (positive redaction)

As an example, the question to ask about the PEOU of mechanism M1 was "Did you find it difficult to review the usage scenarios, user stories, and acceptance tests?"

In order to create these questionnaires we used Google Drive Forms[12]. This way, participants could answer the questionnaire online and their responses were recorded automatically. As this software supports shuffling questions, we only had to write the questions and the questions were offered to each participant randomly ordered. All these questionnaires are gathered in Annex C.

RQ2 requires a response variable to measure the satisfaction of developers. To measure this satisfaction, we also followed the method evaluation model (MEM), although this time, we did not used subjective Likert scales because the developer is the same who developed the approach. Specifically, we measured the perceived usefulness of the developers through the following metrics:

- Comprehension questions: These are the doubts and complaints that end-users ask the developers about each mechanism of the method. The developers measured this metric after applying each mechanism by counting the total number of questions that geneticists asked. Moreover, the developers recorded the specific questions and comments in order to be able to identify the cause of the doubt.
- Degree of agreement: This is the level of agreement achieved after comparing and aggregating the feedback provided by all of the end-users for each mechanism. The developers obtained the number of items in which the majority of the geneticists agreed, the number of items in which the majority of the geneticists disagreed, and the number of items provided individually. Feedback was captured separately for each

[12] https://www.google.es/intl/es/forms/about/

geneticists while applying each mechanism; for this reason, (depending on the agreement achieved), the developers decided whether or not to incorporate the new feedback into the DSL.

- Undetected errors: These are the errors in the DSL artefacts (concrete syntax grammar, abstract syntax metamodel, etc.) that are missed by the end-users despite applying the mechanisms. The developers measured this metric after geneticists tested the DSL by counting the number of errors that geneticists detected in the Testing stage and by identifying which mechanism should have helped the geneticists to detect those errors earlier.

RQ3 requires a response variable to measure the time spent by end-users to apply the mechanisms of the method. To measure the time spent, the developers observed how many minutes each subject spent carrying out the activities proposed for each of the mechanisms.

8.7 Experiment design

We chose the experimental design "one factor, one treatment". Since we run an experiment of the type "Researching expert opinion", our aim was to obtain information only about the mechanisms of our method. This design was compatible with our restriction of having a small number of subjects (only three geneticists), but this restriction hindered the possibility of selecting a subset, creating several groups, or applying randomization. Hence, only one treatment was applied to all of the subjects, therefore there were no *block variables* and *balancing* was not needed.

In order to apply the treatment, we had to organize several sessions in which we applied one mechanism of the method at a time. This restriction was imposed by the configuration of the method itself. As we explained in the previous chapters, in order to create the different DSL artefacts according to the end-users' needs, the method mixes development activities to be carried out by the developers (creation of DSL artefacts such as conceptual models) with mechanisms for gathering the end-users' feedback. After applying each mechanism (and before continuing with the next one), the developers need to process the feedback gathered from the end-users and to develop the

corresponding DSL artefact(s). For this reason, it was not possible to apply all the mechanisms of the method in the same session.

This design avoided the following threats: 1) the reduction of the sample size since we did not divide the geneticists into groups; and 2) the influence of the problem addressed since all of the geneticists developed the same problem. However, this design had other threats: 1) the generalization of our results to any development since we only used one object due to the small sample size; and 2) a learning effect in the last mechanisms since there were several sessions.

8.8 Experimental objects

Since the subjects of the experiment were geneticists from an industrial environment, we had the opportunity to create experimental objects that represent a real domain problem instead of a toy example. In order to acquire some knowledge about their work, the developer met with the geneticists before the experiment and the geneticists gave a brief overview of their work. As a result of these meetings, the developer created the experimental objects.

The experimental objects were the set of requirements that were going to be supported by a DSL. Hence, in order to configure an achievable experiment with short sessions (1 or 2 hours), the developer had to choose a subset from all of the requirements given by the geneticists. In the end, four requirements were selected for the experiment:

EO1. "Read the polymorphisms[13] of several patients from a file".

EO2. "Retrieve the effects of the polymorphisms from a prediction algorithm[14]".

EO3. "Filter polymorphisms by the effect types [15] calculated by the prediction algorithm".

---

[13] A polymorphism is a change in the DNA sequence of an individual with respect to a "DNA reference sequence". The "DNA reference sequence" is a representative sequence of all the individuals from a species, which is agreed upon by the genetic community.

[14] An effect prediction algorithm predicts how the polymorphism changes the DNA and therefore the corresponding protein. As a result of this analysis, it predicts whether this change is benign or damaging for the patient.

[15] The effect types represent a benign or a damaging effect, however, depending on the algorithm further types can be predicted.

EO4. "Calculate the frequency of the polymorphisms[16]".

Therefore, the goal of the experiment was to gather information about the mechanisms (M1-M5) while applying the method to create a DSL that supports these four requirements.

8.9 Instruments

This section describes the instruments that were used to run the experiment:

- **Artefacts of the method**: These are the artefacts that have been proposed in the method to develop the DSL: 1) the DSL backlog (artefact AA1); 2) the requirements (artefact AA2); 3) the domain model (artefact AA3); 4) the syntax models (artefact DA1); 5) the semantic models (artefact DA2); 6) the tests (artefact IA1); 7) the DSL infrastructure (artefact IA2); 8) the demonstration (TA1); and 9) the DSL feedback (artefact TA2).
- **Mechanisms of the method**: These are the mechanisms that have been proposed in the method to involve end-users in the development of the DSL (mechanism M1-M5).
- **Instruments of the experiment**: These are the questionnaires that have been designed (instruments E1-E5) to assess the end-users' satisfaction (RQ1) for each mechanism (M1-M5).
- **Guidelines of the experiment**: These are the documents for guiding end-users through the different experiment activities. The *Overview_Guideline* summarizes all the activities to be performed by geneticists in the experiment. The guidelines *T1_Guideline*, *T2_Guideline*, *T3_Guideline*, *T4_Guideline*, and *T5_Guideline* explain the session and the training, and the *Experimenter_Guideline* helps the

[16] The frequency of a polimorfism is the number of times that this polymorphism appears in the patients divided by the total number of patients.

experimenter in the conduction and execution of the complete experiment plan.

8.10 Experiment procedure

This section describes the experiment that was executed. Figure 8.1 presents an overview of the meetings between the developer and the geneticists before starting the execution of the experiment. In meeting 1, the geneticists from INCLIVA explained their domain and their daily work to the developer. This meeting was necessary in order to know their working context and to be able to define real experimental objects. After this meeting, the developer selected the experimental objects (explained in Section 8.8).

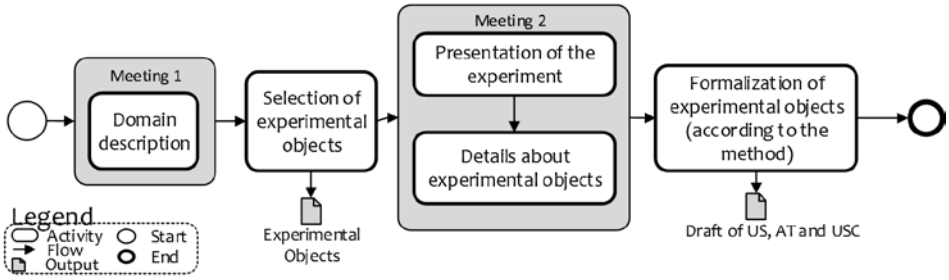
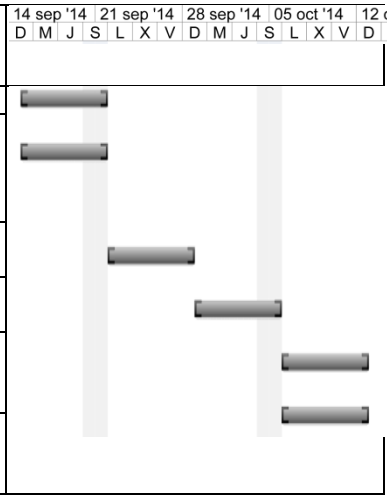


Figure 8.1 Experiment overview

Once the experimental objects were selected, in meeting 2, the developer presented to the geneticists the goal of the experiment, a proposal for the schedule, and the general structure of the experimental sessions. In order to provide freedom to the geneticists to choose the most suitable days for them to participate in the experiment, we scheduled each session in different weeks, one session per week (See Table 8.3). The sessions started on time (the second week of September), however, due to miscellaneous reasons (holidays and work restrictions), the next sessions were postponed a few weeks, and the experiment finished one month later (second week of November).

Table 8.3 Preliminary schedule

Activity	Estimated time (mins)	Start	End
T0. Presentation	10 mins	15/09/14	21/09/14
T1. Review Usage Scenarios, User stories and Acceptance tests	40 mins	15/09/14	21/09/14
T2. Syntax Questionnaire	15 mins	22/09/14	28/09/14
T3. Semantic Templates	40 mins	29/09/14	05/10/14
T4. DSL Prototype Demonstration	10 mins	06/10/14	12/10/14
T5. DSL Prototype Assessment	15 mins	06/10/14	12/10/14



After the geneticists had all of the experiment information and agreed to the schedule, the developer fulfilled the DSL backlog with the list of requirements to be addressed (artefact AA1) and geneticists described further details about these requirements. Using this information, the developer formalized the geneticists' requirements in the form of *User stories (US), acceptance tests (AT), and usage scenarios (USC)* (artefact AA2).

After these two meetings and the formalization of requirements, we executed the experiment sessions. The experiment had 4 sessions of approximately one hour each, which were held once a week (4 weeks in total). All of the sessions were carried out face-to-face with the developer, except session 2. Session 2 consisted of completing two questionnaires; since the developers' presence was not imperative for answering these questionnaires, we offered them the chance to complete them on their own.

Each session had the same structure: 1) a training of the activities to be performed in the session; 2) the application of the mechanism of the method being validated; and 3) a questionnaire for assessing the geneticists' satisfaction regarding the mechanism. The training was performed with all of the geneticists together, and the application of the mechanism was done individually by each geneticist with the help of the developer. The objective of the individual meetings between the developer and each geneticist was to get the most personalized feedback possible. This decision was made to avoid the threat of losing important feedback, which is a common issue when an experiment is performed

simultaneously with several subjects. Additionally, individual meetings avoid the learning effect among end-users when they talk.

Below, we describe the four sessions of the experiment as well as the tasks performed by the developer after each of these sessions, a.k.a post-sessions. In each experiment session, the developer and the geneticists collaborated to apply the mechanisms of the method, and in the post-sessions, the developer processed the geneticists' feedback to create the corresponding DSL artefacts of the method (Figure 8.2):

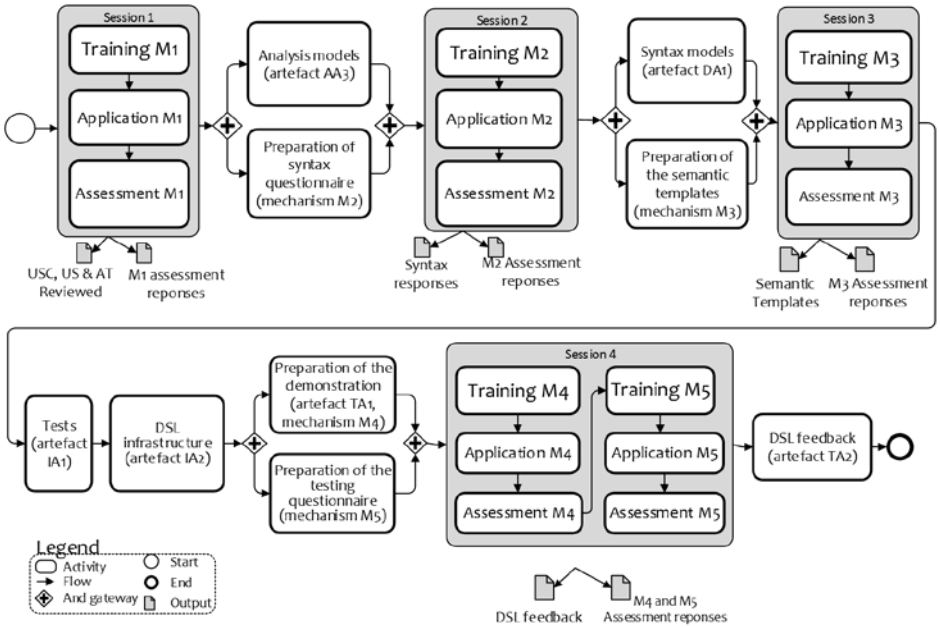


Figure 8.2 Overview of the steps of the experiment

- **Session 1**: The goal of this session was to apply and gather information about mechanism M1. This session included the following tasks:
  1. Training of mechanism M1: The developer presented the activity to be carried out during this session and provided the guidelines of the method that explain the structure of the user stories (US), the acceptance tests (AT), and the usage scenarios (USC).
  2. Application of mechanism M1: The geneticists together with the developer reviewed the subset of the US, AT, and USC descriptions (artefact AA2) that had been formalized by the developer before starting the experimental sessions (Figure 8.1, meeting 2). During

this activity, the developer wrote down all of the comments and the time spent.

3. Assessment of mechanism M1: The geneticists answered the assessment questionnaire about mechanism M1 (instrument E1).

- **Post Session 1**: After applying mechanism M1, the developer created the analysis models of the DSL (artefact AA3), designed the abstract syntax metamodel and several concrete syntax options (artefact DA1), and prepared the syntax questionnaire (mechanism M2).
- **Session 2**: The goal of this session was to apply and gather information about mechanism M2. The geneticists performed this session on their own. This session included the following tasks:
  1. Training of mechanism M2: The developer sent to the geneticists by email an explanation of the activity to be carried out and the guidelines of the method that explain the syntax questionnaire.
  2. Application of mechanism M2: The geneticists answered the syntax questionnaire to indicate their preferences about the DSL syntax and to refine the abstract syntax. Since the developers were not present during this activity, the questionnaire included two additional questions to gather their comments and measure the time spent.
  3. Assessment of mechanism M2: The geneticists answered the assessment questionnaire about mechanism M2 (instrument M2).
- **Post Session 2**: After applying mechanism M2, the developer analyzed the responses about syntax preferences and designed the definitive syntax models (artefact DA1). Additionally, the developer prepared the semantic templates (mechanism M3).
- **Session 3**: The goal of this session was to apply and gather information about mechanism M3. This session included the following tasks:
  1. Training of mechanism M3: The developer presented the activity to be carried out during this session and provided the geneticists with the guidelines of the method that explain the structure of the semantic templates.
  2. Application of mechanism M3: The geneticists together with the developer completed the templates to specify the DSL semantics (artefact DA2). During this activity, the developer recorded all of the comments and the time spent.

3. Assessment of mechanism M3: The geneticists answered the assessment questionnaire about mechanism M3 (instrument E3).

- **Post session 3**: After applying mechanism M3, creating the set of tests (artefact IA1) and implementing the complete DSL infrastructure (artefact IA2), the developer prepared the demonstration (artefact TA1, mechanism M4) and the questionnaire to assess the DSL infrastructure (mechanism M5).
- **Session 4**: The goal of this session was to apply and gather information about mechanism M4 and mechanism M5. In order to reduce the number of the experiment sessions and avoid the geneticists to meet for another experiment session, we decided to combine the assessment of both mechanisms into a single session of two hours. This session included the following tasks:
  1. Training of mechanism M4: The developer presented the activity to be carried out during the first part of this session (session 4a) and provided the guidelines of the method that describe the demonstration structure.
  2. Application of mechanism M4: The developer demonstrated the DSL and encouraged geneticists to provide comments. During this activity, the developer recorded all of the comments and the time spent.
  3. Assessment of mechanism M4: The geneticists answered the assessment questionnaire about mechanism M4 (instrument E4).
  4. Training of mechanism M5: The developer presented the activity to be carried out during the second part of this session (session 4b) and provided the guidelines of the method that describe the testing questionnaire.
  5. Application of mechanism M5: The geneticists used the DSL infrastructure freely and answered the testing questionnaire. During this activity, the developer recorded all of the comments and the time spent.
  6. Assessment of mechanism M5: The geneticists answered the assessment questionnaire about mechanism M5 (instrument E5).
- **Post session 4**: After applying mechanism M5, the developer analyzed the feedback obtained from the geneticists in order to improve the DSL infrastructure (artefact TA2).

8.11 Evaluation of validity

Before running the experiment, we needed to ensure that the complete experiment design was suitable to achieve the experimental goal. Specifically, we wanted to ensure its validity to assess the satisfaction and efficiency of the mechanisms of the proposed method.

The experiment design was validated in terms of four criteria: internal validity, external validity, construction validity, and conclusion validity. To assess each type of validity, we followed the work of [98] and discussed all the potential threats that apply to the context of our experiment:

1. **Internal Validity**: Internal validity assesses the causality relationships of the experiment between the treatment and the outcome of the experiment:
    a. *Maturation* threat is the appearance of different subjects' reactions as time passes. In our experiment, besides the method restrictions, we organized the experimental task in different sessions, with an approximate duration of up to two hours to keep geneticists from getting bored. In order to ensure that the satisfaction of geneticists for each mechanism was captured equally, geneticists were asked about the mechanisms of the method at the end of each session. This was done to avoid problems recalling their opinion about the mechanisms they applied first.
    b. *Testing* threat is the appearance of different responses when a test is applied several times because it has been learned by subjects. Our experiment did not have this threat because each mechanism and the corresponding assessment instruments were only applied once. Moreover, the mechanisms were different from each other since they tackled different aspects of DSL development. By using one mechanism, the geneticists did not acquire further knowledge that could affect the usage of the following ones.
    c. *Instrumentation* threat can appear if instruments are not well designed. In order to mitigate this situation, we conducted a pilot experiment with two Master students that had a biotechnology degree and expertise in bioinformatics. With their feedback, we solved some errors and clarified some terminology and concepts that were needed for the experiment.

   d. *Hierarchical relationships among experimenter and subject*s can affect the planning of the experiment and the results obtained. The geneticists participated voluntarily (for research purposes), so the developer was not in full control of the experimental environment because she was not in a hierarchical position to give orders. Therefore, this threat was not avoided.
   e. *Hierarchical relationships among subjects* can affect the freedom of subjects to provide their opinion. In our experiment, since one of the geneticists supervised the other, these two geneticists might have waited for feedback provided by their supervisor and hid their real opinion. We avoided this threat by meeting with geneticists individually.

2. **External Validity**: External validity assesses the ability to generalize the experiment results to industrial practice:
   a. *Inadequate subjects* is the threat of having subjects that are not representative of the population that is going to be generalized to. We avoided this threat since the chosen subjects were geneticists who were experts in genetic disease diagnosis and who perform genetic analysis in their daily work. Moreover, they fit the expected profile because only geneticists with that domain experience were suitable to be involved in the development of a DSL for the genetic analysis domain. However, we did not avoid this threat for the developer because the developer and the method creator were the same person. This situation hindered our ability to get information about the learnability of the mechanisms of the method for developers since the developer already knew the details of the mechanisms before starting the experiment. In order to reduce the subjectivity when measuring the developer's satisfaction using the method, we proposed objective metrics (comprehension questions, degree of agreement, and undetected errors).
   b. *Inadequate objects* is the threat of having objects that are not representative of industrial practice. We avoided this threat since the experimental objects were selected from the daily work requirements of the geneticists. Hence, the target DSL was not a toy DSL because it addressed a problem that occurs in a real environment.

   c. *Inadequate environment* is the threat of conducting the experiment in a specific environment or on a special day or time that affects the subjects. We avoided this threat by conducting the experiment in the geneticists' workplace and during the days and times that were the most suitable for them.

3. **Construction Validity**: Construct validity assesses whether the experiment design is able to measure the effect of the treatment over the experimental objects and ensure the reliability of the observed outcome:
   a. *Inadequate preoperational explication* constructs is the threat of having constructs that are not sufficiently defined before they are translated into measures or treatments. We avoided this threat by adopting the metrics and instruments of the Method Evaluation Model (MEM), which have already been used in several experiments of the literature and have been validated by several authors.
   b. *Mono-operation bias* is the threat to experiments with one single factor or treatment in which the experiment underrepresents the construct. In order to mitigate this threat, the Method Evaluation Model (MEM) [102], proposes a set of metrics to assess satisfaction and a questionnaire that is specially designed to measure them. To ensure the validity of the questionnaire, the author includes redundant questions, describes questions in positive and negative style, and excludes the items that fail a correlation analysis (Chronbach's alpha [103]). In our experiment, we adopted the same metrics and adapted the questionnaire to our context. We also included redundant questions and we described them in positive and negative style. However, we were not able to run Chronbach's alpha analysis due to the small number of subjects.
4. **Conclusion Validity**: Conclusion validity assesses the ability to extract conclusions from the results of the experiment:
   a. *Low-statistical* power is the threat of not having enough data to be able to identify a true pattern. Our experiment had this threat due to the small sample. Since we were not able to mitigate this threat, we could not conclude that the results were statistically significant. However, we conducted the experiment with this issue in mind with the following goals: 1) to demonstrate the feasibility of the proposal;

and 2) to obtain preliminary insights of the success of the method to involve end-users in real practice.

b. *Fishing for a result* is the threat that occurs when the experimenter seeks a specific result. We did not avoided this threat because the experimenter was the same person that designed the method and nobody else was available to conduct the experiment.

8.12 Data analysis

As a result of the experiment, we obtained data from three sources: 1) the responses from geneticists' satisfaction questionnaires; 2) the geneticists' feedback recorded by the developer; and 3) the time spent by each geneticist while applying each mechanism of the method. All these data are gathered in Annex C.

For the **end-users' satisfaction** response variable, we gathered five data sets (one per mechanism). It should be noted that: 1) we negated some questions to stimulate the attention of the geneticists; 2) we combined questions that assess two metrics (perceived ease of use and perceived usefulness) in the same questionnaire; 3) we used several questions to ask about the same metric to ensure robustness of responses; and 4) we used a 5-level Likert Scale to assess each metric. Hence, in order to correctly analyze this response variable, we must pre-process the obtained data sets so that they can be compared. To do this, we proceeded as follows:

- **Standardization of responses**: Since we narrated the questions using positive and negative adjectives to attract the attention of the geneticists, we had to standardize the responses so they could be compared. Therefore, we changed the responses of the negative questions to their opposites (e.g., when the response was "strongly disagree" it was changed for "strongly agree"). For example, Table 8.4 shows the original responses of the questionnaire of the second session (Syntax Questionnaire Assessment, instrument E2) and Table 8.5 shows how the responses to the negative questions Q1 and Q4 have been changed for their opposites; while the responses of the positive questions Q2, Q3, and Q5 remain unchanged.

Table 8.4 Responses about the assessment of the syntax questionnaire

	Q1	Q2	Q3	Q4	Q5
Geneticist G1	Disagree	Agree	Strongly Agree	Disagree	Agree
Geneticist G2	Strongly disagree	Strongly Agree	Agree	Disagree	Agree
Geneticist G3	Neutral	Agree	Neutral	Disagree	Neutral

Table 8.5 Responses about the assessment of the syntax questionnaire after standardization

	Q1	Q2	Q3	Q4	Q5
Geneticist G1	Agree	Agree	Strongly Agree	Agree	Agree
Geneticist G2	Strongly agree	Strongly Agree	Agree	Agree	Agree
Geneticist G3	Neutral	Agree	Neutral	Agree	Neutral

- **Separation of responses into two data sets, one per metric (PEOU and PU)**: Since the questionnaires asked about two different metrics, the responses could not be compared and had to be analyzed separately. For each mechanism (M1- M5), PEOU was measured through three questions (Q1, Q2 and Q3) and PU was measured through two questions (Q4 and Q5). Hence, we separated the responses into two datasets (one per metric). As an example, Table 8.5 is divided into one dataset that contains the responses to questions Q1, Q2, and Q3 (which asked the geneticists about PEOU) and one dataset that contains the responses to questions Q4 and Q5 (which asked about PU). These datasets are shown in Table 8.6 and Table 8.7, respectively.

Table 8.6 Ease of Use responses about the assessment of the syntax questionnaire

PEOU	Q1	Q2	Q3
Geneticist G1	Agree	Agree	Strongly Agree
Geneticist G2	Strongly agree	Strongly Agree	Agree
Geneticist G3	Neutral	Agree	Neutral

Table 8.7 Usefulness responses about the assessment of the syntax questionnaire

PU	Q4	Q5
Geneticist G1	Agree	Agree
Geneticist G2	Agree	Agree
Geneticist G3	Agree	Neutral

Once we obtained the two datasets with each geneticist's responses separated by metric (PEOU and PU), we aggregated their responses to obtain further information about the geneticists' opinions. We focused our analysis on the following questions:

- **What is each geneticist's opinion about each mechanism by metric?** In order to answer this question, we aggregated the responses of each

geneticist by metric (PEOU and PU). Since the question scale is ordinal, in order to aggregate the responses, we calculated the median value (according to [96]). To do this, we associated each Likert level with a number; since there were five categories, the numbers ranged from 1 to 5 ("strongly disagree"=1, "disagree"=2, "neutral"=3, "agree"=4 and "strongly agree"=5). As an example, Table 8.8 shows the equivalent numbers of the responses provided by geneticist G1 about the PEOU of the mechanism M2 (syntax questionnaire): "4", "4", and "5", representing "agree", "agree", and "strongly agree", respectively. These numbers correspond to questions Q1, Q2, and Q3 that were shown in Table 8.6. In order to obtain the opinion of geneticist G1 about the PEOU of mechanism M2, we calculated the median of these numbers, whose result was "4".

Table 8.8 Geneticist G1's opinion about the PEOU of Mechanism M2

Mechanism M2				
PEOU	Q1	Q2	Q3	Median (opinion about mechanism M2)
Geneticist G1	4	4	5	4

- **What is each geneticist's opinion about the treatment by metric?** In order to answer this question, we aggregated the responses of each geneticist about mechanisms M1-M5 by metric (PEOU and PU) (from Table 8.8). As before, since the scale of the responses was ordinal, we calculated the median to aggregate the responses. As an example, Table 8.9 shows the opinion of geneticist G1 about the metric PEOU for the mechanism M1-M5, whose numbers are "5", "4", "4", "5", and "5", respectively. In order to obtain the opinion of geneticist G1 about the treatment, we calculated the median of these numbers, whose result is "5".

Table 8.9 Geneticist G1's opinion about the PEOU of the treatment

PEOU	**M1**	**M2**	**M3**	**M4**	**M5**	**Median (opinion of geneticist G1 about the treatment)**
Geneticist G1	5	4	4	5	5	5

- **What is the general opinion about the treatment by metric?** In order to answer this question, we aggregated the geneticists' responses about the treatment (from Table 8.9)**.** As before, since the scale of the responses was ordinal, we calculated the median. As an example, Table 8.10 shows the opinion of the geneticists G1, G2, and G3 about the metric PEOU

for the treatment, whose numbers are “5”, “5”, and “4”. In order to obtain the general opinion of geneticists about the PEOU of the treatment, we calculated the median of these numbers, whose result is “5”.

Table 8.10 General opinion about the PEOU of the treatment

PEOU	**Treatment**
Geneticist G1	5
Geneticist G2	5
Geneticist G3	4
Median (opinion of all of the geneticists about the treatment	5

For the **developers' satisfaction** response variable**,** the developer gathered feedback from the geneticists about all of the mechanisms of the method during the experiment sessions. In order to analyze all this data, we classified it into one of the three metrics of the developer's satisfaction: comprehension questions, degree of agreement, and undetected errors. As described in the Section 8.6, all these metrics were measured through quantitative values, such as the number of questions, the percentage of agreement, disagreement, and individual feedback, and the number of errors.

For the **time** response variable**,** the developer measured the time spent by the geneticists to apply each mechanism. In order to know the total time spent by geneticist, we added the individual times to apply each mechanism. Additionally, in order to know the average time that geneticists needed to apply each mechanism, we calculated the average of the times spent by the three geneticists.

8.13 Results

In this section, we discuss about the experiment results for each of the response variables: 1) end-users' satisfaction; 2) developers' satisfaction; 3) and time to apply the mechanisms.

For the **end-users' satisfaction** response variable, we obtained information about: 1) the geneticists' individual satisfaction for each mechanism (Table 8.11); 2) the geneticists' individual satisfaction for the treatment (Table 8.12); and 3) the general satisfaction for the treatment (Table 8.13).

Table 8.11 shows the values that represent the geneticists' opinion about each mechanism. For the *PEOU* metric, all of the geneticists mostly “agree” or “strongly agree” that the proposed mechanisms are easy to use, with only one

geneticist (geneticist G3) being neutral about the mechanism M2 (syntax questionnaire) and the mechanism M3 (semantic templates). For the *PU* metric, the geneticists' opinion is similar and we observe values ranging from "neutral" (3) to "strongly agree" (5); being higher values more frequent.

Table 8.11 The opinion of each geneticist about each mechanism

	Perceived Ease of Use (PEOU)					**Perceived Usefulness (PU)**				
	M1	M2	M3	M4	M5	M1	M2	M3	M4	M5
Geneticist G1	5	4	4	5	5	4	4	4,5[17]	5	5
Geneticist G2	4	5	4	5	5	5	4	3	5	5
Geneticist G3	4	3	3	5	5	3	3,5[17]	5	5	5
1="Strongly Disagree", 2="Disagree", 3="Neutral" 4="Agree", 5="Strongly Agree".										

Table 8.12 shows the values that represent the geneticists' opinion about the treatment. For the *PEOU* metric, we observe that the median values that wre calculated for each geneticist (the five mechanisms of the method) are always equal to or greater than 4 ("agree"). This means that the geneticists' perception about the ease of use is high. For the *PU* metric, we observe that the values are always greater than 4, which means that the geneticists' perception of usefulness is also high.

Table 8.12 The geneticists' opinion about the treatment

	Perceived Ease of Use (PEOU)	**Perceived Usefulness (PU)**
	Treatment	Treatment
Geneticist G1	5	4,5
Geneticist G2	5	5
Geneticist G3	4	5

These results are consistent with Table 8.13, which shows the values that represent the general opinion about the treatment. We observe that the aggregated opinion of all geneticists is 5 for both metrics (PEOU and PU), so we can say that, in general, the three geneticists that participated in the experiment were satisfied with the mechanisms of the method.

Table 8.13 Population's opinion about the treatment

	Perceived Ease of Use (PEOU)	**Perceived Usefulness (PU)**
	Treatment	Treatment
All geneticists	5	5

For the **developers' satisfaction** response variable**,** we obtained information about: 1) the comprehension of the mechanisms (Table 8.14); 2) the degree of

[17] This median contains decimal places because it is calculated from two values.

agreement among the geneticists (Table 8.15); and 3) the undetected errors (Table 8.16).

Table 8.14 shows the amount of *comprehension questions* asked by the geneticists and the total number of questions asked per mechanism. The total number of questions asked were low, which means that, in general, the geneticists did not have doubts about the mechanisms. They correctly understood how to review user stories, acceptance tests, and usage scenarios (mechanism M1) since only one question arose during the application of the mechanism. Specifically, geneticist G1 did not understand what the abbreviation US and AT meant, so as a solution, we propose adding the definitions of the abbreviations of the user stories, acceptance tests, and usage scenarios in the guidelines of the mechanism. Additionally, the geneticists indicated that they understood the acceptance tests particularly well because they reflected examples of their domain, with specific and real inputs and specific and real outputs.

Table 8.14 Comprehension questions that were asked by the geneticists

	Comprehension Questions				
	M1	M2	M3	M4	M5
Geneticist G1	1	0	1	0	2
Geneticist G2	0	0	1	0	0
Geneticist G3	0	0	1	0	2
Total	1	0	3	0	4

Good comprehension was also indicated for the syntax questionnaire (mechanism M2) and the demonstration of the DSL infrastructure (mechanism M4), since the geneticists did not ask any questions. In contrast, several questions were asked for the semantic templates (mechanism M3) and the testing questionnaire (mechanism M5). For mechanism M3, all three geneticists asked about the source field and they indicated that this field was not clear. As a solution, this field should be changed for two new fields: authors/creators, which defines the stakeholders that created the artefact; and link to the service provider, which indicates if the artefact is obtained through a website, a framework, or locally from the file system.

Similarly, for mechanism M5, the geneticists asked several questions. Geneticist G1 left two questions of the testing questionnaire unanswered, and geneticist G3 asked for clarification about two other questions. These comprehension questions showed that this questionnaire needs to be improved. Also, the developer observed that the relationship between the activities suggested for the testing and the questions of the questionnaire was not clear for

the geneticists. As a solution, the questions will be reviewed to remove ambiguities and the testing activities will be redesigned in a way that end-users see the relationship between activities and questions. We propose to interleave both activities and questions in a way that end-users are asked to try one aspect of the DSL and then they are asked about it immediately afterwards.

Table 8.15 shows *degree of agreement* among the geneticists' feedback. Mechanisms M3 and M4 showed a 100% of agreement among the three geneticist, while M1, M2, and M5 showed different distributions of agreement, disagreement and individual feedback. For mechanism M1, all of the geneticists agreed that one of the user stories was ambiguous (3 comments, 20%), but each of them provided different comments and proposed changes (12 changes, 80%) in the other user stories, acceptance tests, and usage scenarios. This means, that mechanism M1 was useful for detecting big mistakes but not so powerful for detecting small ones.

Table 8.15 Percentages of agreement from the geneticists' feedback.

	Degree of Agreement				
	M1	M2	M3	M4	M5
Agreement	20%	33%	100%	100%	66%
Disagreement	0	50%	0	0	22%
Individual feedback	80%	13%	0	0	12%

For the syntax questionnaire (mechanism M2), lots of different feedback was provided. All of the geneticists agreed on the suitability of two of the proposed syntaxes (2 agreements, 33%), but they disagreed on the suitability of the other two, and, eventually, they completely differed on their preferred syntax (3 disagreements, 50%). Moreover, only one geneticist proposed a change in the syntax (1 change, 13%). As a consequence, the developer had to weight their answers to try to choose the best rated syntax option. This means that mechanism M2 should be improved by getting additional feedback about all the syntax options to ensure that the final result satisfies all the end-users as much as possible.

For both the semantic templates (mechanism M3) and the demonstration (M4), all the results provided by the three geneticists were identical. Since all of the geneticists completed the template with the same information (agreement of 100%), and no feedback was provided (agreement of 100%) for the mechanism M4, there is no evidence that suggests that this mechanisms should be improved.

In the testing questionnaire (mechanism M5), the geneticists agreed on 27 responses (66%), disagreed on 9 (22%), and provided additional feedback in 5

responses (12%) (e.g., the correction of the syntax keyword possibly_damaging or the addition of a new user story *filter by list of genes from file*). In general, there were many coincidences by the geneticist, even though full agreement between all of the geneticists was not achieved in some cases. Individually, some of them gave suggestions that were not given by the others. Nevertheless, no contradiction arose, which means that mechanism 5 was suitable to understand the general opinion about the DSL.

Table 8.16 shows the *undetected errors* that should have been detected earlier during the application of the mechanisms. The geneticists missed four errors related to mechanism M1 (review of user stories, acceptance tests, and usage scenarios) and mechanism M2 (syntax questionnaire). For mechanism M1, geneticist G2, indicated the need of including a new user story named *filter by list of genes from file* (1 error). For mechanism M2, all the geneticists proposed new changes to the syntax, which means that the questions of mechanism M2 were not effective in gathering end-user requirements. Specifically, geneticists G1 and G3 suggested transforming the enumeration *AlphAsc, AlphDes, Min2Max, and Max2min* into *Ascendant and Descendant*" (2 errors), and geneticist G2 suggested adding an underscore to the keyword *possibly_damaging* (1 error). These undetected errors revealed that mechanism M1 should be improved to detect additional requirements that are highly related to existing ones and mechanism M2 should be improved to encourage end-users to provide feedback about the specific details of the abstract and concrete syntax. In order to improve the syntax questionnaire, we propose: 1) removing all the questions with free-text answers; 2) adding a question that shows the entire scenario as a whole (instead of splitting it up into several questions); 3) coloring different syntax elements to highlight the different constructs; 4) and providing more engaging questions about each syntax construct.

Table 8.16 Undetected errors that were observed by geneticists in the testing stage

	Undetected Errors				
	M1	M2	M3	M4	M5
Geneticist G1	0	1	0	0	-
Geneticist G2	1	1	0	0	-
Geneticist G3	0	1	0	0	-
Total	1	3	0	0	-

For mechanisms M3 and M4, the geneticists did not find any error. For mechanism M5, we cannot measure the undetected errors unless we run an

additional iteration and detect the errors that could have been missed in the application of this mechanism (testing questionnaire).

For the **time** spent to apply the mechanisms of the method, Table 8.17 shows that the mechanisms that required more time were M1, M3, and M5, and the mechanisms that required less time were M2 and M4. In general, the geneticists spent similar times applying the mechanisms, although there was greater variety among the geneticists when applying mechanisms M1, M2 and M3. In total, geneticists G1 and G2 spent approximately 2 hours while geneticist G3 spent 3 hours.

Table 8.17 Time spent by each geneticist

Time(minutes)	M1	M2	M3	M4	M5	Total
Geneticist G1	24	7	25	14	57	127
Geneticist G2	30	4	26	14	60	134
Geneticist G3	49	12	46	14	60	181
Average	34,3	7,7	32,3	14	59,0	147,3

8.14 Threats to validity

In this section, we discuss the experiment experiences related to the threats to validity that arose during the experiment.

With regard to **internal validity**, we detected a usability problem in the measurement instrument of the mechanism M1 (instrument E1) during the first session. The geneticists indicated that they were having doubts about choosing which response represented their opinion. The questions asked for agreement or disagreement for several statements, but they doubted between choosing *agree* or *disagree* when the question statement used a negative adjective such as difficult or tedious.

For practical reasons, we decided not to repeat the questionnaire of this first session. Instead, in order to ensure the validity of the responses already provided, we analyzed them looking for inconsistencies. As we explained in Section 8.6, the geneticists were asked several questions to measure the same metric: 3 questions to measure the PEOU metric, and 2 questions to measure the PU metric. Thanks to this, we were able to compare the responses that assess the same metric keeping in mind that some answers could be expressing the opposite of what the

participants wanted to answer. For instance, geneticist G1 answered three questions regarding the PEOU metric for mechanism M1. In two of the questions, this geneticist indicated a high PEOU but a very low PEOU in the other question. When asked to review the three questions, the geneticist detected that one of the questions was not correctly answered and changed the response given. In total, we detected two inconsistencies that we verified with the corresponding geneticists.

Once we detected this problem in instrumentation (and before executing the rest of the sessions), we analyzed the other instruments E2, E3, E4, and E5 to check if they had the same problem. Since the questions were worded similarly, we changed some labels to improve usability. For instance, when the question statement asked about how difficult a mechanism was, we changed the label of the responses to clarify it. As a result, the labels were *agree (difficult)* and *disagree (easy)*.

We did not observe any other inconsistency in the other sessions; however, in order to ensure the validity of the instruments for future experiments, a thorough analysis should be done to assess the understanding of end-users about the labels *agree* and *disagree* and the clarifications that we added.

Another threat related to internal validity was that the social aspects between the developer and the geneticists affected the planning of the experiment. During Session 1, one of the geneticists received a phone call to solve a work matter and left the experiment activity (temporarily). The developer could not keep the geneticists from leaving; as a consequence, the geneticist's perception could have been affected and no action was available to avoid this threat.

With regard to **external validity**, we had a potential change in the experimental environment. Session 4 was planned as a session to be performed with all of the geneticists on the same day; however, due to work restrictions, we had to schedule several sessions: one for geneticists G1 and G2, and another one for geneticist G3. Initially, we believed that the day a session is held could affect the experimental environment, but since all the geneticists accomplished all the tasks of the session normally, there is no evidence that the environment was compromised. Geneticist G3 carried out the activities without any noticeable issues. Had Subject 3 attended the original session, the fact that the work problems were not solved could have affected his attitude and motivation to participate in the experiment.

8.15 Discussion

As a result of this experiment, we have found that the set of mechanisms of the method are suitable for involving end-users in the context of DSL development since the results show that the mechanisms are satisfactory to both end-users and developers. We now discuss the usability problems, the limitations, and the potential improvements in the mechanisms of the method.

From the end-users' perspective (**the end-users' satisfaction variable**), the high levels of perceived ease of use and perceived usefulness lead us to think that the mechanisms of the method are a good approach for end-users to communicate their needs and preferences and do not have any usability problem. However, from these measures, we are not able to extract further conclusions about the *limitations* or *potential improvements* of the method.

From the developers' perspective (**the developers' satisfaction variable**), the low values for the comprehension questions metric, the high values for the degree of agreement metric, and the low values for the undetected errors metric lead us to think that developers perceive all of the mechanisms of the method as a good approach to gather end-users' feedback and represent it in the DSL. The feedback and anecdotes that gathered developers from the geneticists during the application of the mechanisms has provided a big picture about developers' satisfaction, but it has been also useful for detecting some usability problems, limitations, and points of improvement.

Thanks to the comprehension questions that the geneticists asked during their experiment and the existence of undetected errors, we have detected *usability* problems in mechanisms M1, M2, M3, and M5: for mechanism M1, some abbreviations were not clear; for mechanism M2, some questions were tedious and too open; for mechanism M3, a field from the template was ambiguous; and for mechanism M5, some questions were similar or they contained ambiguous expressions.

Additionally, we have detected a *limitation* in mechanism M2 due to the low degree of agreement in the geneticists' responses. The geneticists were asked to rate each syntax proposed and eventually to choose their favorite. We expected to find variability in their preferences, but we also expected a slight convergence since all the geneticists are experts in the same area and they accomplished the

same tasks (and used the DSL for the same purpose). However, since each of the three geneticists chose a different syntax, the developer found it very difficult to determine the most suitable one. This situation has revealed that the geneticists were asked about few aspects of concrete syntaxes.

As a solution for these usability problems and limitations, we propose the following *potential improvements*: for mechanism M1, improving the guidelines to clarify all the abbreviations; for mechanism M2, performing a study of usability aspects about concrete syntaxes and using the results to improve the syntax questionnaire and to create a weighting system that obtains the most suitable concrete syntax for end-users; for mechanism M3, removing ambiguities from the template; for mechanism M4, encouraging end-users to provide feedback during the demonstration; and for mechanism M5, removing ambiguities in the questions and improving the relationship between activities and questions.

Finally, for the time spent (**the time response variable**) to apply the mechanisms of the method, the geneticists invested up to three hours to apply the five mechanisms, which seems to be a reasonable amount of time to be spent on participating in the development of a DSL. With this information, we have not been able to detect usability problems in the mechanisms; however, the time measures have revealed which mechanisms are the most costly for end-users (mechanisms M1, M3, and M5). For mechanism M5, we believe that the reason could be the difficulties of learning the DSL and how the DSL application is deployed, which actually means that there is a *limitation* in mechanism M4. If the geneticists had problems getting familiar with the DSL while applying mechanism M5, this could mean that the demonstration (mechanism M4) was not effective. As a *potential improvement*, we propose to define a demonstration session that is more interactive and encourages the end-users to ask questions about the DSL, even if the session is interrupted.

In summary, we believe that mechanism M1 was well accepted among the geneticists because of the easy structure of the templates based on user stories, acceptance tests, and usage scenarios and also because of the use of natural language. Similarly, using questionnaires (mechanism M2 and M5) is the most common way to gather end-user feedback. Therefore, we believe that our questionnaires have been well received among the geneticists because they were short and easy to answer and they used domain examples in the questions. Similarly, mechanism M3 has been well accepted because the templates

structured the data in different sections and they were not extremely difficult or large. Mechanism M4 has been well accepted because the demonstration was visual, and it did not require any effort from the geneticists. This is in contrast to the testing questionnaire, which is the most complex because the geneticists had to carry out a set of activities with the DSL infrastructure and find errors in the DSL.

It is worth mentioning that these conclusions are the experimental results of applying only one iteration of the method, which addresses four requirements with three geneticists. As we already mentioned in Section 8.11, these preliminary results assess the feasibility of the proposal and provide preliminary insights about the acceptance of the involving mechanisms by both end-users and developers. However, having such a small number of subjects limits our ability to generalize these results to the population.

Besides the limitations of the method, during the experiment, we also learned some lessons that are related to the experimentation design and execution. First, asking end-users to perform a task during their free time it is not productive. We asked the geneticists to answer the syntax questionnaire (mechanism M2) in their free time, but after a week, only one geneticist had done it, and we had to insist several times. Since practitioners have a busy working schedule, it is difficult for them to accommodate an experimental task in their free time. Second, in contrast, we learned that spending time with practitioners is worthwhile since the feedback they provide is more oriented to industry. Third, we were able to get a lot of individual feedback and avoid social validity threats because our experiment was especially designed for subjects to participate individually. Since we only had three subjects, applying the different method mechanisms individually was feasible.

8.16 Conclusions

In this chapter, we have validated the mechanisms for involving end-users with an experiment of the type "Researching expert opinion" with three subjects from an industrial environment. The experiment focuses on evaluating the users' participation in the DSL development process with respect to end-users' and developers' satisfaction while applying the involving mechanisms of the method. As a conclusion of the experiment, we found that the use of agile practices has

helped to involve end-users in DSL development since the data measured in the experiment showed high levels of end-user satisfaction and developer satisfaction with the involving mechanisms. We also found that these mechanisms can be applied in a reasonable amount of time.

As we already mentioned, the results obtained from this experiment cannot be generalized due to their sample size. However, the findings are very valuable because the data originates in experts from an industrial environment and the experimental objects are based on a subset of their real problems. Another relevant aspect of this experiment is the benefit obtained from observing the participants individually. We gathered very detailed feedback about the experiment and the method; a kind of feedback that is more difficult to observe with a bigger sample.

Despite the lack of generalization, this experiment has been useful to learn some lessons about the method, although it was useful as well to learn how to improve the planning of a future experiment. Regarding the method, we have detected usability issues, limitations, and potential improvements in several mechanisms of the method. Regarding the experiment, we have detected threats to validity that were not considered during design and problems in the guidelines, the measurement instruments, and the schedule.

In summary, despite the limitations of the experiment, its impact is twofold. On the one hand, we have proved the feasibility of the proposal by applying the method in a real context. This means that, using the method, we have been able to involve end-users in DSL development and we have obtained positive feedback from end-users about the involving mechanisms. On the other hand, we have gathered a big amount of feedback about the mechanisms of the method to improve it for the next method version.

9. Conclusions

This PhD thesis focuses on the field of DSL development. We identified the need of involving end-users within the process and we analyzed the state of the art to identify which issues remain unsolved. As a solution, we proposed a method to involve end-users in DSL development and we validated the proposal with geneticists in a real DSL development context.

In this chapter, we start summarizing the contributions of this PhD thesis. Next, in order to justify the value of these contributions, we present the list of publications, their relevance, and their coverage in regards to each contribution. Then, we discuss the lessons that we have learned while conducting the research of this PhD. Finally, we present the future research lines to improve this work.

9.1 Contributions

Contribution 1: A discussion of the need of involving end-users in DSL development.

In order to illustrate the need of involving end-users in DSL development we chose the genetic analysis domain as example. We collaborated with geneticists

from two different industrial environments to characterize their problems with genetic analysis software tools: Imegen (in the context of the Project Diagen[18]) and INCLIVA.

We analyzed the possibility to propose a DSL for geneticists; but as a result of this analysis, we realized of the complexity of this domain and the need of involving them in the development process to ensure the suitability and correctness of this DSL. This contribution is presented in Chapter 2.

Contribution 2: State of the art of DSL development approaches for involving end-users.

In order to justify that the motivation of the PhD thesis has not been previously addressed, we searched for works in the literature that address the participation of end-users in the development of DSLs. First, we analyzed the most relevant works in the literature related to DSL development and observed how they take into account end-users. Second, we searched for works that focused on involving end-users in DSL development and we compared them in regards to process completeness and end-user involvement. Since none of them provided a DSL development method that besides involving end-users also covers the complete DSL development lifecycle, we identified a gap in the state of the art. This contribution is presented in Chapter 3.

Contribution 3: Design and implementation of a method to involve end-users in DSL development.

As a solution to the gap that we identified in the state of the art, we proposed a new DSL development method. In order to design this method, we studied approaches for DSL development, model-driven practices in the context of DSL development, and agile practices that focus on enhancing the participation of end-users. After this analysis, together with the geneticists and bioinformaticians from Imegen, GEM Biosoft and INCLIVA, we proposed a DSL development method that combines model-driven and agile practices to improve the DSL development efficiency and to enhance end-user participation.

Additionally, this method is the proof of concept of combining into a single proposal the practices of such different development paradigms as model-driven

[18] "Modulo de carga para genes especificos en una base de datos del genoma humano con el objetivo de facilitar la busqueda de variaciones en secuencias genomicas y su interpretacion fenotipica", Instituto De Medicina Genomica, S.L.

development and agile development. Model-driven development focuses on creating rigorous and complete representations and use these representations to create the corresponding software implementation, while agile principles are more focused on producing working software so it can be early delivered to end-users. This method has achieved a balance between those paradigms focusing on delivering software to end-users as soon as possible but enhancing the creation of models throughout the process and guidelines to transform those models into working software.

This contribution is presented in chapters 4, 5, 6, and 7.

Contribution 4: A DSL for supporting genetic analysis.

After designing this method, again together with the geneticists from Imegen, GEM Biosoft, and INCLIVA, we applied it to develop a DSL for supporting genetic analysis. This DSL provides domain abstractions to allow geneticists to specify their pipelines for genetic analysis. At the moment, this DSL is a prototype that only supports a subset of constructs, but it can still be used to specify a set of genetic analysis of diseases such as *Breast Cancer* or *Diabetes Mellitus Type 2*. This contribution is presented in chapters 4, 5, 6, and 7. Further details are provided in the Annex B and in the technical report [83].

Contribution 5: Validation of the proposal in a real environment

In order to assess the whether the proposal is a suitable approach to enhance end-user involvement in DSL development, we conducted an experiment of the type *Researching expert opinion* with three geneticists from INCLIVA. In this experiment, we gathered information from the end-users' and the developers' perspective about the end-user involving mechanisms. During this experiment, we also identified the current limitations and the potential improvements to the method. This contribution is presented in Chapter 8.

9.2 Research publications

The contributions of this PhD thesis have been presented to the software engineering and the bioinformatics communities through the following publications:

[1] Oscar Pastor, Ana M. Levin, Matilde Celma, Juan Carlos Casamayor, Luis Eraso, Maria José Villanueva and Manuel Perez-Alonso, "Enforcing Conceptual Modeling to Improve the Understanding of Human Genome", *International Conference on Research Challenges in Information Science (RCIS 2010)*. IEEE Computer Society, pp. 85-92, 2010.

[2] Maria José Villanueva, Francisco Valverde, Ana M. Levin and Oscar Pastor, "Diagen: A Model-driven Framework for Integrating Bioinformatic Tools", *Forum of the International Conference on Advanced Information Systems Engineering (CAiSE Forum)*, pp. 105-112, 2010.

[3] Maria José Villanueva, Francisco Valverde, Ana M. Levin and Oscar Pastor, "Diagen: A Model-Driven Framework for Integrating Bioinformatic Tools", *CAiSE Forum (Selected Papers), Lecture Notes in Business Information Processing,* Springer-Verlag, Heidelberg, ISBN 978-3-642-29748-9, vol. 107, pp. 49-63, 2012.

[4] Maria José Villanueva, Francisco Valverde and Oscar Pastor, "Applying Conceptual Modeling to Alignment Tools One Step towards the Automation of DNA Sequence Analysis", *International Conference on Bioinformatic models, methods and algorithms* (*BIOINFORMATICS)*, pp. 137-142, 2011.

[5] Maria José Villanueva, "An agile model-driven approach for simplifying the development of genetic analysis tools", *International Conference on Advanced Information Systems Engineering (RCIS)*, pp. 1-6, 2012.

[6] Maria José Villanueva, Francisco Valverde and Oscar Pastor, "Involving End-users in Domain-Specific Languages Development -Experiences from a Bioinformatics SME", *International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, pp. 97-108, 2013.

[7] Maria José Villanueva, Francisco Valverde and Oscar Pastor, "Involving End-Users in the Design of a Domain-Specific Language for the Genetic Domain", *International Conference on Information Systems Development: Improving Enterprise Communication (ISD)*, pp. 99-110, 2013.

[8] Maria José Villanueva Francisco Valverde and Oscar Pastor, *"Como diseñar pipelines científicos sin tener que aprender programación ni comandos Linux*", *I Congreso Biomedicina Predocs (CONBIOPREVAL)*, 2014.

[9] Francisco Valverde and Maria José Villanueva, *"Applying Capability Modelling in the Genomics Diagnosis Domain: Lessons Learned*",

International Workshop on Capability-oriented Business Informatics (CoBI), CEUR workshops Proceedings, 2015.

[10] Maria José Villanueva, Francisco Valverde, Ignacio Panach and Oscar Pastor, "Involving end-users in the development of domain- specific languages (DSLs): Researching expert opinion". Submitted to *Journal of Software and Systems*.

Table 9.1 overviews these publications and shows the relevance of each of them, indicating the type of communication (publication in editorial, workshop, forum, short paper at conference, regular paper at conference, or article), whether they have been published in a conference or editorial with international reach, and whether this conference or an editorial is classified according to any relevant ranking (such as CORE or JCR). This table also shows the contents of the publication, indicating which contributions of the thesis (motivation, state of the art, solution, and validation) have been covered.

Publications [1], [2], [3], [5], [6], [7], and [9] have been published in forums related with software engineering to present the five contributions. Publications [4] and [8] have been published in forums related with biotechnology and bioinformatics in order to present the contributions 1 and 4, that is, the motivation related to the genetic analyses domain and the DSL for supporting genetic analysis.

Table 9.1 Summary of publications

	Relevance			Coverage				
	Communication type	International	Ranking	Contrib.1 (Motivation)	Contrib. 2 (State of the Art)	Contrib. 3 (Solution)	Contrib. 4 (DSL)	Contrib. 5 (Validation)
[1]	Regular paper	✓	Core B	✓	-	-	-	-
[2]	Short paper at forum	✓	CORE A (forum)	✓	-	-	-	-
[3]	Selected paper for publication	✓	-	✓	-	-	-	-
[4]	Short paper at conference	✓	-	✓	-	-	-	-
[5]	Doctoral consortium	✓	Core B (doctoral)	✓	-	-	-	
[6]	Regular paper	✓	Core B	✓	✓	✓	✓	✓
[7]	Regular paper	✓	Core A	✓	✓	✓	✓	✓
[8]	Oral communication	×	-	✓	-	-	✓	-
[9]	Workshop paper	✓	-	✓	-	-	-	-
[10]	Article (submitted)	✓	JCR	✓	✓	✓	✓	✓

9.3 Discussion

In this section, we discuss the research conducted in this PhD thesis, analyzing the benefits and the limitations of the proposed method and the developed DSL.

Regarding the **investigation of the problem**, we found that developing a DSL is difficult because it requires coming up with the right domain abstractions for the users of the DSL. Since the developers are the ones who know how to provide these abstractions but not the ones that hold the knowledge about the domain, this task is harder as the complexity of the domain increases. In order to illustrate this complexity, as an example we chose the genetic analysis domain, a domain

with very complex concepts that has traditionally struggled with software tools because of the huge existing gap between domain experts and developers.

We met and interviewed geneticists who work daily performing genetic analyses. We were able to precisely analyze their problems because they were very interested and willing to answer all of our questions. Thanks to their enthusiasm, they showed us real examples of their genetic analyses to illustrate their problems. Because of this collaboration, we were able to observe the real complexity that involves developing a DSL for this domain. This complexity justifies both the need of advanced software engineering practices during the development process and the need of involving geneticists from the beginning.

In this PhD thesis, we analyzed the genetic analysis domain and we found that developing a genetic DSL required involving geneticists. However, we believe that this problem is not specific to the genetic analysis domain and it can we generalized to other domains, especially to complex ones whose domain concepts are not closer to the background knowledge of a software engineer. We are aware that each domain has their own particularities; however, we believe that eliciting requirements and end-users' preferences for a DSL is always difficult (no matter the domain). Also, trying to understand the semantics of the different specific concepts of a domain is also hard for developers that do not have the proper domain knowledge.

Regarding the **state of the art**, our main source was a systematic mapping of the current state of DSL development, and we also analyzed the most relevant and most cited DSL development proposals focusing on their approaches for involving end-users. In order to find additional proposals that involve end-users in some activities of DSL development, we have been constantly looking for novel proposals in journal and conference proceedings using academic search engines. We analyzed the most relevant ones by characterizing their benefits and their still unsolved issues. Although we did not perform a systematic review, we are positive that there is no method for involving end-users in DSL development that supports the whole development lifecycle (from decision to maintenance).

Regarding the **proposed method**, we have described a method for involving end-users in DSL development that combines model-driven development and agile practices. The proposed method covers all the stages of the development lifecycle by detailing the steps that must be followed and the artefacts that must be created in each step; ensures the process efficiency, through the adoption of

model-driven practices and the definition of model-based transformations to systematize the development; and involves end-users in the definition and assessment of different DSL artefacts through five agile-based involving mechanisms applied in the stages analysis, design, testing and maintenance.

In order to design this method, we collaborated with geneticists from Imegen, GEM Biosoft and INCLIVA. This collaboration was very valuable to ensure that the ideas and feedback from the end-users of a complex domain were reflected in the method. We collaborated with Imegen and GEM Biosoft to design and improve three versions of the method and we carried out an empirical validation with geneticists from INCLIVA. The benefit of designing and validating the proposal in different environments is reducing the chances of having a method highly dependent on the preferences of the end-users that collaborated in the method design. A set of end-users provided feedback to design the method and another set of end-users provided feedback once the method was designed.

Additionally, in order to propose a set of mechanisms to involve end-users throughout the development process, we studied agile methods and selected agile practices that focused on involving end-users. We selected and analyzed each practice and we customized them to fit the specifics of the DSL development context. From this experience, we proposed a set of involving mechanisms, which simplified some development tasks by using closer language to end-users or real examples.

Nevertheless, during the design of the method, we found a set of issues that we could not addressed in the context of this PhD. We have described these issues at the end of each method chapter, in Sections 5.3, 6.3, and 7.4. Examples of these issues are the inattention to usability requirements for the DSL, the lack of support of internal DSLs, or not having applied the stages deployment and maintenance in real practice. Next, in Section 9.4, we discuss the ideas that we have for future work in order to solve each of these issues.

Regarding the developed **DSL for supporting genetic analysis**, we applied the proposed method with geneticists from Imegen, GEM Biosoft, and INCLIVA to develop a DSL for their complex domain. This DSL aimed to provide a friendly infrastructure to specify genetic analysis without worrying about selection, configuration, and integration of software tools.

The current version of this DSL prototype supports the specification of a set of pipelines to perform genetic analysis of genetic diseases such as Lactose Intolerance, Breast Cancer, and Diabetes Mellitus Type 2. At the moment, this DSL only supports a small set of constructs since we only executed three iterations of the method and the two first were focused on being a proof of concept for the method. Nevertheless, although the generated specifications still do not support all of the details to specify a complete genetic analysis, the geneticists from INCLIVA provided positive feedback about the DSL. Next, we describe some of the insights that we gathered from this experience.

The DSL is useful for geneticists who do not have technological skills. The geneticists from INCLIVA are currently specifying their analysis using and configuring by themselves existing software tools. They acquired computer skills, and they perceive that by using the DSL they lose control and flexibility over their pipelines for genetic analysis. For instance, the DSL has a construct to annotate the variations' gene, but the geneticists are used to work with specific software tools and they need to know which tool is performing the annotation, which parameters are configuring the tool, and even which version of the tool is being used. In contrast, these geneticists admit that acquiring all this technical knowledge and experience is not easy and the DSL could be very helpful for geneticists who do not have this background.

The DSL offers the opportunity to embrace a new paradigm for genetic analysis research. Complementing the previous thoughts, this DSL can offer geneticists (including as well the geneticists who have a high technical knowledge) the possibility to decouple the genetic analysis from existing software tools. Nowadays, when geneticists design a genetic analysis they focus on the tool selection and configuration instead of thinking about the overall process. As a solution, the DSL provides a clear separation of concerns. On the one hand, geneticists can use the DSL to design and specify the genetic analysis they want to perform. On the other hand, they can deal separately with the mapping between DSL constructs and the most suitable software implementation. In this scenario, geneticists deal with the domain complexity, whereas software engineers (or by geneticists with bioinformatics skills) deal with technological complexity.

The DSL needs to be very expressive but easy to understand. In order to convince geneticists to embrace this new paradigm, we need the DSL to be expressive enough to support any analysis that geneticists can think of. This concern is a

challenge in this domain because of its constant evolution. DSLs usually target domains with well-established concepts; for this reason, at the moment, we have oriented this DSL to support well-established genetic analysis procedures for medical diagnosis. Therefore, this DSL can be used to save and reuse the specification of pipelines that are used in the every-day diagnosis or for teaching purposes to help novice geneticists to learn how to create different genetic analysis pipelines.

Finally, for the **validation,** we carried out an experiment with geneticists from INCLIVA, and although only three geneticists participated in the experiment, we obtained valuable information from experts about the involving mechanisms. As a result, we learned that some of the mechanisms had some issues and we gathered information from the geneticists about how to improve them. We have described these issues in the validation chapter, concretely in Sections 8.13 and 8.15. Examples of these issues are the need of clarifying the abbreviation used in the mechanism M1, the need of running a further usability study to improve the mechanism M2, or the need of removing an ambiguous field in the mechanism M3. Next, in Section 9.4, we discuss our ideas to solve each of them as well as how some of them have been already solved.

9.4 Future work

In summary, in this PhD we have proposed a method that supports all the stages of the DSL lifecycle, adopts model-driven practices to benefit from MDD efficiency, and adopts agile practices to involve end-users. For practical reasons, we prioritized covering all the stages of the development process over covering different design possibilities. As a consequence, we have left for future work the following research lines:

- *The mechanisms of the method do not gather feedback in regards to usability or performance concerns*. The mechanisms of the method have been designed especially to gather end-user feedback regarding domain knowledge and functional requirements. Our priority was to gather end-user's requirements, preferences about the DSL syntax, information about DSL semantics, and feedback about each DSL release. However, we did not designed the involving mechanisms to gather non-functional

requirements such as usability or performance. Since, this kind of requirements are also important for the adoption success of the DSL, as future work, we aim to modify or add new involving mechanisms to take into account these aspects. In the state of the art, there is a proposal [104] that performed an analysis of the usability concerns to take into account during DSL development and proposed a questionnaire to ask end-users about these concerns in the Testing stage. As future work, we will analyze the possibility to integrate their proposal in our DSL development method. Using this approach, we could improve the mechanisms of the Analysis, Design, and Testing stages to take into account usability concerns.

- *The method only supports the development of external DSLs*. The design of the syntax and semantics was restricted to support external DSLs because covering all the DSL development lifecycle was a priority. Among the decision to support an external or an internal DSL, we chose to support external DSLs first because the scenario of application that was used in this PhD (a DSL for supporting genetic analysis) follows this approach. This decision was made according to the geneticists needs, but this does not mean that the method could not support the development of internal DSLs as well. As future work, we will analyze which artefacts are needed to design the syntax and semantics of an internal DSL and how to extend the existing language infrastructure to support internal DSLs.
- *The method only supports the development of textual DSLs*. The design of the syntax was restricted to support textual DSLs because, as we already explained, covering all the DSL development lifecycle was a priority. Among the decision of supporting graphical or hybrid syntaxes, we chose to support textual syntaxes first because the use case that was developed in this PhD (a DSL for supporting genetic analysis) follows this approach. Also, developing a graphical editor for such DSL is a time consuming implementation task and we needed an editor that could be shown to the end-users as soon as possible. As a future work, we will design the artefacts that are needed to design a graphical syntax (such as, defining the notation of the graphical syntax using GMF), as well as the mechanisms to show graphical and hybrid syntaxes to the end-users to rate and to choose their preferred one. We would add a questionnaire to ask about their preferences before proposing several syntaxes, or adopt

the ideas of one of proposals of the state of the art (such as [71] or [72]) in which end-users can draw their preferred graphical shapes for the syntax. Additionally, we will analyze how to implement the parser that recognizes graphical and hybrid syntaxes.

- *The method only supports the specification of operational semantics.* Although there are several approaches to specify the semantics of a language: operational, denotational, and axiomatic, we only support the specification of semantics through the specification of a mapping between a DSL construct and a service that is implemented by a technological artefact. The rationale of this decision was the easiness of understanding for end-users and the easiness of specification and implementation for the developers. The drawback is that this approach requires a set of ready-to-use services already available, which can be a limitation for some domains. As future work, we will explore the alternatives that can be applied for domains where such services are not available yet.
- *The method does not have tool support*. Although we used several tools like textual editors and frameworks for DSL development, we do not have a single tool to manage all the steps and artefacts from the method and guide its application. In order to implement this tool support, as future work we will analyze the possibility to apply the proposal of Cervera et al. [105]. This work allows software developers to create a support environment for their method, which integrates under a single environment all the software tools that are used in the method steps and includes the method steps to guide the developers through the development. Using this work, we could implement a tool that supports the interaction of the method with both the developers and the end-users.
- *The stages Deployment and Maintenance have not been assessed with end-users, applied in practice, or validated.* The method includes the stages Deployment and Maintenance and a mechanism to involve end-users in the maintenance stage. However, these stages were designed in the last iteration of the method and they still need a further analysis and assessment of the artefacts to be created. Moreover, we did not include these stages in the experiment with the geneticists since we considered that the DSL release was not stable enough to be deployed and delivered. As future work, we plan to perform a further analysis of the artefacts of

these stages as well as to develop a more stable version of DSL so we can assess these stages with geneticists.

- *The experiment focused on validating the mechanisms of the method.* We validated the main contribution of the PhD with an empirical experiment. The benefits of MDD (such as efficiency) in the context of DSL development and the different artefacts of the method have been already validated in existing works of the state of the art. Nevertheless, as future work, we believe that it is necessary to validate with developers and end-users the entire method working as a whole: steps, artefacts, guidelines, and involving mechanisms.
- *The validation experience identified potential improvements in the involving mechanisms*. During the experiment, thanks to the feedback gathered from the geneticists, we detected some issues in the mechanisms. We solved some of the issues by modifying the artefacts, guidelines, and mechanisms of the method. In order to solve the ambiguity of the mechanism M1, we changed the guidelines and redefined each abbreviation that was unclear. In order to solve the ambiguity found in mechanism M3, we changed one of the fields and explained this change in the corresponding guideline. In order to improve the comprehension of the DSL release for end-users in the mechanism M4, we changed the developer guidelines to encourage end-users to participate during the demonstration. Finally, in order to improve the comprehension of the questionnaire of mechanism M5, we rewrote some of the questions that were ambiguous and joined the ones that were repetitive. For future work, we left some of the limitations detected. In order to improve mechanism M2, we will run a further study about the different aspects to take into account when designing a concrete syntax for end-users. Then, we will use the conclusions of this analysis to improve the questionnaire of the mechanism M2. Also, in order to improve mechanism M5, we will analyze how to establish a better link between the activities to test the DSL release and the testing questionnaire.

Besides the research lines identified in regards to the method, as a result of collaborating with the geneticists, we also identified future work related to the DSL for supporting genetic analysis:

- *The textual syntax may not be the best for every geneticist*. Using the method, the questions that were asked to geneticists revealed that using a textual syntax was the most suitable approach for geneticists. This was due to the fact that the geneticists from INCLIVA are used to program their own pipelines using scripts, so they are comfortable with a textual syntax. This syntax was the best for the geneticists involved; however, since the goal of the DSL is to specify pipelines, it is possible that a hybrid approach (graphical and textual) could facilitate geneticists the description of the data and control flow. As future work, we will assess the advantages and drawbacks of these alternatives and review the method questions that are used to decide between an external and internal approach.
- *Validate a more stable version of the DSL with an empirical experiment.* Although the geneticists provided informal comments about their opinion about the DSL, we did not provide empirical data that demonstrates whether this DSL could be a solution to their current problems with software tools, or whether this DSL will improve their experience in contrast with existing bioinformatics pipeline development environments. As future work, we want to run further iterations of the method to obtain a more stable version of the DSL and validate with an empirical experiment whether a DSL is a suitable solution to improve the geneticists' efficiency. Our plan is to compare how efficient is creating a pipeline specification with the DSL in comparison with creating a pipeline with a bioinformatics pipeline development environment such as Galaxy [52].

9.5 Final thoughts

DSLs are the most natural and simple way to encapsulate source code, tasks, or knowledge and improve the efficiency of their users. Since I started doing my research on DSLs, I also started to see DSLs everywhere in my daily routine. They were already there, but I had not seen it before. For example, the DSL for writing music when I was at band practice.

What defines a DSL is not always clear since it is a matter of defining the boundaries of a domain; boundaries that are not predefined and are open to interpretation. What is indeed clear is that DSLs have no other aim than facilitate

descriptions or tasks. That is why I found DSLs interesting and worth to study for a PhD.

After being in contact with geneticists (and their software problems) for several months, I wanted to know whether a DSL could be a suitable solution for their problems. However, I realized that I was not able to respond to this question. I could not develop this DSL since I was not a geneticist. Geneticists could not develop the DSL either. We had to do it together.

I researched how DSLs were developed, but they did not mentioned much the role played by the domain experts; mostly because the developers designed DSL for technical domains, closer to their knowledge. There, I had found the research problem of my PhD: I wanted to propose a method to help software developers as myself to have guidance and tools to develop such a useful, but difficult-to-develop, kind of languages. Besides proposing this method, from the beginning, I wanted to go a step further and apply this method to create a real DSL; to create the DSL for supporting genetic analysis.

The proposed method is not reinventing the wheel. On the contrary, a lot of interesting work has been done towards the development of DSLs and the method presented in this PhD thesis aims to get the best of these works together (both methodologies and technologies), fill the gaps so that future developers can use it altogether, and support the participation of end-users seamlessly.

There is still work to do for this method to become into a method that covers all the decisions that a DSL developer would face. However, I strongly believe that this proposal goes a step further towards that direction.

We did not released the DSL for genetic analysis into real practice. However, after seeing the geneticists' responses to the first DSL release and after seeing that their problems remain as the time passes, I also believe that the definition of a DSL to specify and customize genetic analysis software tools is a project that can provide huge benefits for geneticists in terms of efficiency and organization.

DSLs are here to stay since, in a figurative way, everything is a DSL

.

10. References

[1] The Standish Group, "Chaos Manifesto 2013", 2013.

[2] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett and others, "The State of the Art in End-User Software Engineering", *ACM Computing Surveys (CSUR),* vol. 43, no. 3, pp. 21:1-21:44, 2011.

[3] A. Van Deursen, P. Klint and J. Visser, "Domain-Specific Languages: An Annotated Bibliography", *SIGPLAN Notices,* vol. 35, no. 6, pp. 26-36, 2000.

[4] M. Fowler, "Domain-Specific Languages", *Addison-Wesley Professional*, 2010.

[5] L. Nascimento, D. Viana, P. Silveira Neto, D. Martins, V. Garcia and others, "A Systematic Mapping Study on Domain-Specific Languages", *The Seventh International Conference on Software Engineering Advances (ICSEA)*, 2012.

[6] M. J. Villanueva, "Diagen: Modelado e Implementación de un framework para el análisis personalizado del ADN", *Master Thesis*, Universitat Politècnica de València, 2011.

[7] M. J. Villanueva, "An agile model-driven approach for simplifying the development of genetic analysis tools", *Research Challenges in Information Science (RCIS)*, 2012.

[8] M. J. Villanueva, A. R. Guzmán, F. Valverde and A. M. Levin, "Diagen: A model-based bioinformatic tool for genetic analysis", *Research Challenges in Information Science (RCIS)*, 2012.

[9] N. Rusk, "Focus on Next-Generation Sequencing Data Analysis", *Nature Methods,* vol. 6, no. 11s, p. S1, 2009.

[10] M. Mernik, J. Heering and A. M. Sloane, "When and How to Develop Domain-Specific Languages", *ACM Computing Surveys (CSUR),* vol. 37, no. 4, pp. 316-344, 2005.

[11] I. Ceh, M. Crepinsek, T. Kosar and M. Mernik, "Ontology Driven Development of Domain-Specific Languages", *Computer Science and Information Systems (ComSIS),* vol. 8, no. 2, pp. 317-342, 2011.

[12] M. Strembeck and U. Zdun, "An approach for the systematic development of domain-specific languages", *Software: Practice and Experience,* vol. 39, no. 15, pp. 1253-1292, 2009.

[13] D. Spinellis, "Notable design patterns for domain-specific languages", *Journal of Systems and Software (JSS),* vol. 56, no. 1, pp. 91-99, 2001.

[14] K. Czarnecki and U. W. Eisenecker, "Generative Programming", *Edited by G. Goos, J. Hartmanis, and J. van Leeuwen,* p. 15, 2000.

[15] C. Atkinson and T. Kuhne, "Model-Driven Development: A Metamodeling Foundation", *IEEE Software,* vol. 20, no. 5, pp. 36-41, 2003.

[16] O. Pastor and J. C. Molina, "Model-Driven Architecture in Practice. A Software Production Environment Based on Conceptual Modeling", *Springer Science & Business Media*, 2007.

[17] B. Selic, "The Pragmatics of Model-Driven Development", *IEEE Software,* vol. 20, no. 5, pp. 19-25, 2003.

[18] B. Hailpern and P. Tarr, "Model-driven Development: The Good, the Bad, and the Ugly", *IBM Systems Journal,* vol. 45, no. 3, pp. 451-461, 2006.

[19] A. Olive, "Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research", *Advanced Information Systems Engineering (CAiSE)*, 2005.

[20] Ó. Pastor and S. Espana, "Full Model-Driven Practice: From Requirements to Code Generation", *Advanced Information Systems Engineering (CAiSE)*, 2012.

[21] D. C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering", *IEEE Computer,* vol. 39, no. 2, pp. 25-31, 2006.

[22] M. Voelter, S. Benz, C. Dietrich, B. Engelmann, M. Helander and others, "DSL Engineering - Designing, Implementing and Using Domain-Specific Languages", *dslbook.org*, 2013, pp. 1-558.

[23] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn and W. Cunningham, "The Agile Manifesto", 2001.

[24] S. Ambler, "Agile modeling: Effective practices for eXtreme Programming and the Unified Process", *John Wiley and Sons Inc.*, 2002.

[25] R. Wieringa, "Empirical research methods for technology validation: Scaling up to practice", *Journal of Systems and Software (JSS),* vol. 95, pp. 19-31, 2014.

[26] H. A. Simon, "The sciences of the artificial", *MIT press*, vol. 136, 1996.

[27] S. T. March and G. F. Smith, "Design and natural science research on information technology", *Decision Support Systems (DSS),* vol. 15, no. 4, pp. 251-266, 1995.

[28] A. R. Hevner, S. T. March, J. Park and S. Ram, "Design Science in Information Systems Research", *MIS quarterly,* vol. 28, no. 1, pp. 75-105, 2004.

[29] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, R. J. Mural and others, "The Sequence of the Human Genome", *Science,* vol. 291, no. 5507, pp. 1304-1351, 2001.

[30] Gene Codes Corporation, *Sequencher,* 2010.

[31] Applied Biosystems, *SeqScape,* 2010.

[32] Codon Code Corporation, *Codon Code Aligner,* 2010.

[33] Softgenetics, *Mutation Surveyor,* 2010.

[34] M. Stephens, J. S. Sloan, P. Robertson, P. Scheet and D. A. Nickerson, "Automating sequence-based detection and genotyping of SNPs from diploid samples", *Nature Genetics,* vol. 38, no. 3, pp. 375-381, 2006.

[35] C. Manaster and others, "InSNP: a tool for automated detection and visualization of SNPs and InDels", *Human Mutation,* vol. 26, no. 1, pp. 11-19, 2005.

[36] P. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks and others, "The variant call format and VCFtools", *Bioinformatics,* vol. 27, no. 15, pp. 2156-2158, 2011.

[37] S. Haider, B. Ballester, D. Smedley, J. Zhang, P. Rice and others, "BioMart Central Portal—unified access to biological data", *Nucleic Acids Research,* vol. 37, no. suppl 2, pp. W23-W27, 2009.

[38] K. Wang, M. Li and H. Hakonarson, "ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data", *Nucleic Acids Research,* vol. 38, no. 16, pp. e164-e164, 2010.

[39] P. Cingolani, A. Platts, L. L. Wang, M. Coon, T. Nguyen and others, "A program for annotating and predicting the effects of single nucleotide

polymorphisms, SnpEff: SNPs in the genome of Drosophila melanogaster strain w1118; iso-2; iso-3", *Fly,* vol. 6, no. 2, pp. 80-92, 2012.

[40] W. McLaren, B. Pritchard, D. Rios, Y. Chen, P. Flicek and others, "Deriving the consequences of genomic variants with the Ensembl API and SNP Effect Predictor", *Bioinformatics,* vol. 26, no. 16, pp. 2069-2070, 2010.

[41] A. McKenna, M. Hanna, E. Banks and others, "The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data", *Genome Research,* vol. 20, no. 9, pp. 1297-1303, 2010.

[42] H. Li and others, "The Sequence Alignment/Map Format and SAM Tools", *Bioinformatics,* vol. 25, no. 16, pp. 2078-2079, 2009.

[43] M. J. Villanueva and F. Valverde, "Software for the genetic analysis domain", *Technical Report*, http://hdl.handle.net/10251/57428".

[44] J. C. Bartlett and E. G. Toms, "Developing a protocol for bioinformatics analysis: An integrated information behavior and task analysis approach", *Journal of the American Society for Information Science and Technology (JASIST),* vol. 56, no. 5, pp. 469-482, 2005.

[45] S. Pabinger, A. Dander, M. Fischer, R. Snajder and others, "A survey of tools for variant analysis of next-generation genome sequencing data", *Briefings in bioinformatics,* vol. 15, no. 2, pp. 256-278, 2014.

[46] L. D. Stein, "Towards a cyberinfrastructure for the biological sciences: progress, visions and challenges", *Nature Reviews Genetics,* vol. 9, no. 9, pp. 678-688, 2008.

[47] W. M. Pereira and G. H. Travassos, "Towards the conception of scientific workflows for in silico experiments in software engineering", in *Empirical Software Engineering and Measurement (ESEM)*, 2010.

[48] J. E. Stajich and others, "The Bioperl Toolkit: Perl Modules for the Life Sciences", *Genome Research,* vol. 12, no. 10, pp. 1611-1618, 2002.

[49] P. J. A. Cock and others, "Biopython: freely available Python tools for computational molecular biology and bioinformatics", *Bioinformatics,* vol. 25, no. 11, pp. 1422-1423, 2009.

[50] R. C. G. Holland, T. A. Down, M. Pocock, A. Prlic, D. Huen and others, "BioJava: an open-source framework for bioinformatics", *Bioinformatics,* vol. 24, no. 18, pp. 2096-2097, 2008.

[51] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers and others, "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud", *Nucleic Acids Research,* vol. 41, no. Web Server Issue, pp. 557-561, 2013.

[52] B. Giardine, C. Riemer, R. C. Hardison, R. Burhans, L. Elnitski and others, "Galaxy: A platform for interactive large-scale genome analysis", *Genome Research,* vol. 15, no. 10, pp. 1451-1455, 2005.

[53] I. Wassink, M. Ooms, P. Neerincx, G. van der Veer, H. Rauwerda and others, "e-BioFlow: Improving Practical Use of Workflow Systems in Bioinformatics", *Information Technology in Bio- and Medical Informatics (ITBAM)*, vol. 6266, S. Khuri, L. Lhotska and N. Pisanti, Eds., Springer-Verlag, Heidelberg, 2010, pp. 1-15.

[54] S. Ghosh, Y. Matsuoka, Y. Asai, K.-Y. Hsin and H. Kitano, "Software for systems biology: from tools to integrated platforms", *Nature Reviews Genetics,* vol. 12, no. 12, pp. 821-832, 2011.

[55] T. McPhillips, S. Bowers, D. Zinn and B. Ludascher, "Scientific workflow design for mere mortals", *Future Generation Computer Systems,* vol. 25, no. 5, pp. 541-551, 2009.

[56] V. Cuevas-Vicenttin, S. Dey, S. Kohler, S. Riddle and B. Ludascher, "Scientific Workflows and Provenance: Introduction and Research Opportunities", *Datenbank-Spektrum,* vol. 12, no. 3, pp. 193-203, 2012.

[57] A. Barker and J. Van Hemert, "Scientific Workflow: A Survey and Research Directions", in *Parallel Processing and Applied Mathematics*, Springer-Verlag Berlin, Heidelberg, 2008, pp. 746-753.

[58] S. Cohen-Boulakia and U. Leser, "Search, Adapt and Reuse: The Future of Scientific Workflows", *ACM SIGMOD Record,* vol. 40, no. 2, pp. 6-16, 2011.

[59] J. Bhagat, F. Tanoh, E. Nzuobontane, T. Laurent, J. Orlowski y others, "BioCatalogue: A universal catalogue of web services for the life sciences", *Nucleic Acids Research,* vol. 38, no. suppl 2, pp. W689-W694, 2010.

[60] M. A. Swertz, M. Dijkstra, T. Adamusiak, J. K. van der Velde, A. Kanterakis and others, "The MOLGENIS toolkit: rapid prototyping of biosoftware at the push of a button", *BMC bioinformatics,* vol. 11, no. Suppl 12, p. S12, 2010.

[61] N. Sedlmajer, D. Buchs, S. Hostettler, A. Linard, E. Lopez and others, "GReg: a domain specific language for the modeling of genetic regulatory mechanisms", *International Workshop on Biological Processes and Petri Nets (BioPN)*, 2012.

[62] M. L. Wilson, S. Okumoto, L. Adam and J. Peccoud, "Development of a domain-specific genetic language to design Chlamydomonas reinhardtii expression vectors", *Bioinformatics,* vol. 30, no. 2 pp. 251-257, 2013.

[63] J. Elhai, A. Taton, J. Massar, J. K. Myers, M. Travers and others, "BioBIKE: a Web-based, programmable, integrated biological knowledge base", *Nucleic Acids Research,* vol. 37, no. suppl 2, pp. W28-W32, 2009.

[64] J. McCarthy, "LISP 1.5 programmer's manual", *MIT press*, 1962.

[65] S. Kelly and J.-P. Tolvanen, "Domain-Specific Modeling: Enabling Full Code Generation", *John Wiley & Sons*, 2008.

[66] F. Perez, P. Valderas and J. Fons, "Towards the Involvement of End-Users within. Model-Driven Development", *End-user Development (EUD)*, 2011.

[67] H. Nishino, "Misfits in abstractions: towards user-centered design in domain-specific languages for end-user programming", *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion (OOPSLA)*, 2011.

[68] H. Nishino, "How can a DSL for expert end-users be designed for better usability?: a case study in computer music", *Extended Abstracts on Human Factors in Computing Systems (CHI)*, 2012.

[69] D. A. Sadilek, M. Scheidgen, G. Wachsmuth and S. Weibleder, "Towards Agile Language Engineering", *Institut fur Informatik*, 2009.

[70] A. Barisic, V. Amaral, M. Goulao and A. Aguiar, "Introducing Usability Concerns Early in the DSL Development Cycle: FlowSL Experience Report", *Model-Driven Development Processes and Practices Workshop Proceedings (MD2P2)*, 2014.

[71] D. Wuest, N. Seyff and M. Glinz, "Semi-automatic generation of metamodels from model sketches", *Automated Software Engineering (ASE)*, 2013.

[72] H. Cho, J. Gray and E. Syriani, "Creating Visual Domain-Specific Modeling Languages from End-User Demonstration", *Modeling in Software Engineering (MISE)*, 2012.

[73] M. Kuhrmann, G. Kalus and A. Knapp, "Rapid Prototyping for Domain-specific Languages-From Stakeholder Analyses to Modelling Tools", *Enterprise Modelling and Information Systems Architectures,* vol. 8, no. 1, pp. 62-74, 2013.

[74] J. Sanchez Cuadrado, J. De Lara and E. Guerra, "Bottom-Up Meta-Modelling: An Interactive Approach", *Model Driven Engineering Languages and Systems (MODELS)*, Springer-Verlag, Heidelberg, 2012, pp. 3-19.

[75] J. L. Canovas Izquierdo and J. Cabot, "Enabling the Collaborative Definition of DSMLs", *Advanced Information Systems Engineering (CAiSE)*, 2013.

[76] A. Blackwell and T. Green, "Notational Systems – the Cognitive Dimensions of Notations Framework", *HCI Models, Theories, and Frameworks: Toward an Interdisciplinary Science,* 2003.

[77] C. Sadowski and S. Kurniawan, "Heuristic evaluation of programming language features: two parallel programming case studies", *SIGPLAN workshop on Evaluation and usability of programming languages and tools (PLATEAU)*, 2011.

[78] A. Barisic, V. Amaral, M. Goulao and B. Barroca, "How to reach a usable DSL? Moving toward a Systematic Evaluation", *Electronic Communications of the EASST,* vol. 50, 2012.

[79] H. Cho and J. Gray, "Design Patterns for Metamodels", *SPLASH '11 Workshops*, 2011.

[80] J. L. Canovas Izquierdo, J. Cabot, J. J. López-Fernández, J. S. Cuadrado, E. Guerra and others, "Engaging End-Users in the Collaborative Development of Domain-Specific Modelling Languages", *Cooperative Design, Visualization, and Engineering*, Springer-Verlag Berlin, Heidelberg, 2013, pp. 101-110.

[81] M. J. Villanueva, F. Valverde and O. Pastor, "Involving End-users in Domain-Specific Languages Development - Experiences from a Bioinformatics SME", *Evaluation of Novel Approaches to Software Engineering (ENASE)*, 2013.

[82] M. J. Villanueva, F. Valverde and O. Pastor, "Involving End-Users in the Design of a Domain-Specific Language for the Genetic Domain", in *Information System Development (ISD)*, 2013.

[83] M. J. Villanueva, F. Valverde and O. Pastor, "DSL Development with Geneticists", *Technical Report*, https://riunet.upv.es/handle/10251/57326".

[84] D. Embley and B. Thalheim, "Handbook of Conceptual Modeling", *Springer-Verlag, Heidelberg*, 2011, p. 608.

[85] B. Henderson-Sellers and J. Ralyté, "Situational Method Engineering: State-of-the-Art Review", *Journal of Universal Computer Science (JUCS),* vol. 16, no. 3, pp. 424-478, 2010.

[86] J. M. Rivero, J. Grigera, G. Rossi, E. R. Luna and F. Montero, "Mockup-Driven Development: Providing agile support for Model-Driven Web Engineering", *Information & Software Technology (IST),* vol. 56, no. 6, pp. 670-687, 2014.

[87] E. Visser, "WebDSL: A case study in domain-specific language engineering", *Generative and Transformational Techniques in Software Engineering II,* pp. 291-373, 2008.

[88] K. Beck and C. Andres, "Extreme Programming Explained: Embrace Change", *Addison-Wesley Professional*, 2004.

[89] K. Schwaber and M. Beedle, "Agile Software Development with SCRUM", *Prentice Hall PTR Upper Saddle River,* vol. 18, 2002.

[90] M. Fowler, "Language Workbenches: The Killer-App for Domain Specific Languages" *Accessed online from: http://www. martinfowler. com/articles/languageWorkbench. html,* pp. 1-27, 2005.

[91] S. Efftinge, "Xtext Reference Documentation", *openArchitectureWare.org,* https://eclipse.org/Xtext/documentation, 2006.

[92] E. Merks, R. Eliersick, T. Grose, F. Budinsky and D. Steinberg, "The Eclipse Modeling Framework", *Adison Wesley*, 2003, p. 37.

[93] R. Likert, "A Technique for the Measurement of Attitudes", *Archives of Psychology,* vol. 22, no. 140, p. 55, 1932.

[94] F. Valverde, "OOWS 2.0: Un Método de Ingeniería Web Dirigido por Modelos para la Producción de Aplicaciones Web 2.0", *PhD Thesis*, Universitat Politècnica de València, 2010.

[95] J. Sanchez Cuadrado, J. Canovas and J. Garcia Molina, "Comparison Between Internal and External DSLs via RubyTL and Gra2MoL", *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments. IGI Global,* 2012.

[96] S. Jamieson and others, "Likert scales: how to (ab) use them", *Medical education,* vol. 38, no. 12, pp. 1217-1218, 2004.

[97] N. Wirth, "Extended Backus-Naur Form (EBNF)", *ISO/IEC,* vol. 14977, p. 2996, 1996.

[98] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell and others, "Experimentation in Software Engineering", *Springer Science & Business Media*, 2012.

[99] N. Juristo and A. M. Moreno, "Basics of Software Engineering Experimentation", *1st ed., Springer Publishing Company, Incorporated*, 2010.

[100] C. Robson, Real World Research, Wiley, 2002.

[101] ISO/IEC/IEE, "Systems and Software Engineering--Vocabulary," 2010.

[102] D. L. Moody, "The Method Evaluation Model: A Theoretical Model for Validating Information Systems Design Methods," in *European Conference on Information Systems (ECIS)*, Utrecht, 2003.

[103] L. J. Cronbach, "Coefficient alpha and the internal structure of tests," *Psychometrika,* vol. 16, no. 3, pp. 297-334, 1951.

[104] A. Barisic, V. Amaral and M. Goulao, "Usability Evaluation of Domain-Specific Languages," in *8th International Conference on the Quality of Information and Communications Technology (QUATIC)*, 2012.

[105] M. Cervera, M. Albert, V. Torres and V. Pelechano, "MOSKitt4ME Approach: Providing Process Support in a Method Engineering Context," in *Conceptual Modeling. Lecture Notes in Computer Science*, Springer-Berlin, Heidelberg, 2012, pp. 228-241.

Annex A

This annex gathers the written guidelines that have been proposed in the method in order to explain to developers and end-users how apply each of the mechanisms (M1-M5) of the method:

A.1 Mechanism M1: Review DSL requirements (user stories, acceptance tests, and usage scenarios)

A.1.1. Guidelines for developers

These guidelines offer an alternative for developers to address end-users and facilitate the review of user stories acceptance tests and usage scenarios. To do that, these guidelines propose a set of questions that can be asked to end-users. The steps to review DSL requirements are:

1) Set the time of start and the time stamps along the document.
2) Ask end-users to read all the scenarios, user stories and acceptance tests
3) Ask the following questions to be able to detect errors, find out desirable changes and new essential elements only regarding the current iteration.

Q1) Did you understood both usage scenarios?

Q2) Are both scenarios fairly reflecting your task?
Q3) ¿Did you find any error? (Indicate Usage Scenario, number and letter)
- Wrong activity (explain why)
- Wrong parameters (point mistakes and correct them)
- Wrong order between activities (indicate activities involved, explain why and propose the correct order)
- Missing activity that is a prerequisite (explain why)

Q4) Did you find any error or would you change something in User stories? (Indicate number of User Story)
- Wrong description (write the words you want to use)
- Wrong action (explain the action you want to perform)
- Wrong goal (explain the goal you want to achieve)

Q5) Did you find any error or would you change something in Acceptance Test? (Indicate number of Acceptance Test)
- Wrong description (write the words you want to use)
- Wrong action (explain the action you want to perform)
- Wrong result (explain the result you expected)
- An additional acceptance test is missing (write the acceptance test using the pre-defined structure: As a role, When context, I will perform action, and I will get the result)

A.1.2. Guidelines for end-users

These guidelines offer a background about the step of revision. This way, end-users know exactly what developers expect from them, but also, they have a reference to consult the rationale of the artefacts of the method "user stories", "acceptance tests" and "usage scenarios". Additionally, this guideline provides a toy example for end-users to understand them. Concretely, an example related with a hotel booking service. The steps to review DSL requirements are:

1) Read all the scenarios, user stories and acceptance tests
2) Discuss with DSL developers the errors you detected or the changes and new essential elements only related with usage scenarios, user stories and acceptance tests included in the draft.

Example of mechanism M1 applied:

Scenario: A complete example of a domain

- Domain: "Tourism Reservation Management"

Scenario: "A client makes a reservation of a hotel for their vacation"

The client first searches the hotels of the city of vacation and chooses one of them. Then the dates of the staying are provided. Then, the information about all the guests to stay in the hotel are provided. Then the payment option is chosen. And finally, the client confirms their final intention to make the reservation.

- Domain: "ATM Operations"

Scenario: "A bank client pays a receipt on the ATM"

The client authenticates in the systems with its personal password. Then, the option to pay a receipt is selected. Then, the client bank account to charge the payment is selected. Then the receipt identifier is provided. And finally, the client confirms the operation.

User Stories: Feature/Step from end-user perspective

As a <role>

I want <action>

So that <goal>

As a client, I **want** to search for the hotels of a city, **so that** I can choose the one I like most.

As a bank client, I **want** to choose the option to pay a receipt, **so that** I can choose the one I like most.

Acceptance Test: Concrete example of a user story

Given <context>

When <action>

Then <result>

Given set of hotels from the hotel chain "Holiday Inn" and hotels in Spain **when** search hotels in Barcelona, **then** I am showed the "Holiday Inn Barcelona", "Holiday Inn Ramblas" and the "Holiday Inn Gracia".

Given set of hotels from the hotel chain "Holiday Inn" and hotels in Spain w**hen** search hotels in Helsinki, **then** I am showed the error message "There is no Hotel from the system in Helsinki".

A.2 Mechanism M2: Syntax questionnaire

A.2.1. Guidelines for developers

These guidelines offer developers the steps to deliver the questionnaire to end-users. The steps are the following:

1) Explain that the goal of the questionnaire: Choose syntax, propose changes to the syntax and identify errors in the description of the usage scenario.
2) Deliver the questionnaire to end-users. It can be handed to end-users or create an online questionnaire and deliver the link, for instance, by email.
3) Process the responses:
    a. Choose the most preferred syntax. To do this, you may pay choose the one that has been considered as preferred by the majority of end-users. If there is not one syntax that highlights over the rest, the preferred syntax is selected by observing the values that have been given by the end-users (Likert questions with scale 1-5). To do so, you may obtain the median of these values for each syntax.

A.2.2. Guidelines for end-users

These guidelines explain the end-users how to proceed to answer the syntax questionnaire. The steps are the following:

1) You will be handed a questionnaire to provide your opinion about the DSL syntax proposed by the developers.
2) Read each syntax proposed and indicate whether you like it or not, being the lowest rate 1 and the highest rate 5.
3) Choose the one you liked most, or make suggestions to propose a new one.
4) If you have chosen one of the syntaxes proposed by the developers, observe the usage scenario and indicate whether you would change or add any word, or step.

A.3 Mechanism M3: Behavioral semantic templates

A.3.1. Guidelines for developers

These guidelines offer developers how to explain end-users the different fields of the template. The steps to fulfil the templates are:

1) Prepare a copy of the user stories together with templates both to end-users and developers.
2) Prepare several copies of the templates in order to hand them to end-users if they need it (document "SemanticTemplates.docx").
3) Address each user story at a time
   - Ask end-users to read the user story.
   - Ask end-users to fulfil the template.

Example of mechanism M3 applied:

Semantic Template: An example of behavior mapping between a user story and a technological artefact

- **User Story**: Name of the user story whose semantics is being defined.
- **Service Identifier:** Name of the software service used to implement the functionality of the user story.
- **Source description**: Additional Information of the origin of the software service. Describing if it is a tool or web service, the operative system compatibility, the provider of the service, etc.
- **Inputs/Outputs**: Details about the information that flows in/out the software service.
  - Name of the parameter
  - Description of the parameter name in natural language
  - Type (it may correspond to an entity of the conceptual model)
  - For input parameters:
    - if it is constant or it may change.
    - Predefined value (usually if the value is constant).
  - For output parameters:
    - If it is an output relevant (visible) for the user story

User Story	**"Search the Hotels from a city"**			
Service Identifier	GetAccommodation			
Source description	Rest Service provided by the provider Booking.com			
Inputs	Description	Type	Constant	Value
City	City of Interest	String	No	-
Search_criteria	Type of accommodation to be searched	Enumeration	Yes	Hotels
Outputs	Description	Type	Visibility	
AccommodationList	List of accommodations that fulfil the search criteria	List of Accommodation	Yes	

A.3.2. Guidelines for end-users

These guidelines describe the different fields of the template and one example. In order to facilitate the explanation, it continues with the example of the hotel booking service. This way, end-users can also realize of the relationship between semantic templates and requirements. The steps to fulfil these templates are:

1) Read the User Story
2) Fulfil the semantics template together with DSL developers

Semantic Template: An example of behavior mapping between a user story and a technological artefact

- **User Story**: Name of the user story whose semantics is being defined.
- **Service Identifier:** Name of the software service used to implement the functionality of the user story.
- **Source description**: Additional Information of the origin of the software service. Describing if it is a tool or web service, the operative system compatibility, the provider of the service, etc.
- **Inputs/Outputs**: Details about the information that flows in/out the software service.
    - Name of the parameter
    - Description of the parameter name in natural language
    - Type (it may correspond to an entity of the conceptual model)
    - For input parameters:
        - if it is constant or it may change.
        - Predefined value (usually if the value is constant).
    - For output parameters:
        - If it is an output relevant (visible) for the user story

A.4 Mechanism M4: Demonstration

A.4.1. Guidelines for developers

These guidelines propose to the developers a script template to present the DSL infrastructure to end-users and the different options they may chose. The steps proposed in this script are:

1. Context description: Describe the scenario to be presented to end-users
2. Syntax presentation: Use the DSL editor to write with the chosen syntax one scenario described in the analysis stage.
    a. Characterize in advance a set of potential mistakes that end-users can introduce in the DSL editor and show the error/warning messages provided.
3. Semantics presentation: Compile the DSL script written in the previous step and show the effects.
4. Execution: If applicable, run the executable produced by the DSL editor and explain and compare the results with the scenario described in the beginning of the demonstration.

The demonstration can be performed online or composing a video in advance.

A.5 Mechanism M5: DSL testing

A.5.1. Guidelines for developers

These guidelines propose a set of activities and a questionnaire to test the following DSL aspects: functional requirements, syntax correctness, semantic correctness and implementation correctness. Although they will be asked about the DSL editor features, they must focus on the language not the editor. The steps are the following:

1) Encourage end-users to play first with the editor and try to reproduce the example of the demonstration and the other usage scenarios.
2) Recall end-users that the DSL is still a prototype that only supports the requirements of the iteration.

3) Suggest end-users to answer the questionnaire once they are familiar with the DSL infrastructure.

A.5.2. Guidelines for end-users

These guidelines suggest to the end-users a set of activities they may accomplish to facilitate the usage of the DSL and the further testing.

1) Get familiar with the DSL infrastructure. You may accomplish the following activities:
    a. Write the same usage scenario of the demonstration.

    …<*written here*>……

    b. Write another usage scenario: Choose one of the following usage scenarios.

    US1) ……..

    US2) ……

    4. With this purpose, you may follow the DSL syntax:

Syntax description

    c. Test the code generated
2) Answer the testing questionnaire.

Annex B

This annex gathers the complete set of artefacts that have been created for the last version of the DSL. Due to lack of space, we provide a summary of the usage scenarios, user stories, and acceptance tests that have been addressed in the three iterations, the final versions of the analysis and design models, and some examples that illustrate the implementation.

B.1 Usage scenarios, user stories and acceptance tests

Table B. 1 Overview of the number of Usage Scenarios, User Stories and Acceptance Tests

Iteration	Usage Scenarios	User stories	Acceptance Tests
1	4	7	14
2	2	5	9
3	3	13	25

B.1.1. Iteration 1

Usage Scenario 1.1. In order to diagnose the Lactase Persistence disease, I want to read the patient variations from a VCF file, annotate the variations with their Hgvs Notation, search the variations in HgvsDna NC_000002.11:g.136608646G>A and NC_000002.11:g.136616754C>A,

and create a report with the variations found with the variations main properties and their hgvs notations.

Usage Scenario 1.2. In order to diagnose the Alkaptonuria disease, I want to read the patient variations from a VCF file, annotate the variations with their Hgvs Notation, search the variations in HgvsCoding NM_000187.3:c.688C>T, NM_000187.3:c.899T>G, NM_000187.3:c.174delA, NM_000187.3:c.16-1G>A, NM_000187.3:c.342+1G>A, NM_000187.3:c.140C>T, and create a report with the variations found with the variations main properties and their hgvs notations.

Usage Scenario 1.3. In order to diagnose the Achondroplasia disease, I want to read the patient variations from a VCF file, annotate the variations with their Hgvs Notation, search the variations in HgvsCoding NM_000142.4:c.1123G>T NM_000142.4:c.1138G>A NM_000142.4:c.1138G>C, and create a report with the variations found with the variations main properties and their hgvs notations.

Usage Scenario 1.4. In order to diagnose the Achondroplasia disease, I want to read the patient variations from a VCF file, annotate the variations with their Hgvs Notation, search the variations in HgvsProtein NP_000133.1:p.Gly375Cys NP_000133.1:p.Gly380Arg, and create a report with the variations found with the variations main properties and their hgvs notations.

Table B. 2 User Stories of Iteration 1

US1.1	I want to read a patient's variations from a VCF file, so that I can analyse potential genetic diseases
US1.2	I want to annotate the patients' variations with the HGVS notation, so that I can see the change at the DNA, Coding and Protein level of each patient's variation expressed using a standard notation.
US1.3	I want to search a set of variations in HGVSDna in the patient's variations, so that I can focus on the suitable variations for the diagnosis
US1.4	I want to search a set of variations in HGVSCoding in the patient's variations, so that I can focus on the suitable variations only
US1.5	I want to search a set of variations in HGVSProtein in the patient's variations, so that I can focus on the suitable variations only.
US1.6	I want to create a report with a list of the variations and their main properties (chromosome, position, reference, alternative), so that I can see the main properties of the patient's variations
US1.7	I want to add to a report with the variations their HGVS (Dna, Coding and Protein), so that I can see the patient variation's expressed in a standard notation

B.1.2. Iteration 2

Usage Scenario 2.1. In order to diagnose the Mammalian Cancer disease (Analysis 1), I want to read the patient variations from a VCF file, annotate the variations with their Hgvs Notation, annotate the variations with their genes, annotate their variations with their rsId from DbSNP,

filter the variations by the genes BRCA1 and BRCA2, and create a report with the variations found with their main properties, their hgvs notations, their genes and their rsIds.

Usage Scenario 2.2. In order to diagnose the Mammalian Cancer disease (Analysis 2), I want to read the patient variations from a VCF file, annotate the variations with their Hgvs Notation, annotate the variations with their genes, annotate their variations with their rsId from DbSNP, filter the variations by the gene RAD51C, and create a report with the variations found with their main properties, their hgvs notations, their genes and their rsIds.

Table B. 3 User Stories of Iteration 2

US2.1	I want to annotate the patient variations with the gene names provided by HGNC, so that I can see all the genes involved each variation
US2.2	I want to annotate the patient variations with the rsId from dbSNP, so that I can see if a variation has been identified with an rsId from dbSNP and get additional information about it afterwards
US2.3	I want filter the patient's variations by a set of genes (by HGNC Gene Name), so that I can focus on the suitable variations only
US2.4	I want to add to a report with the variations their gene, so that I can locate easily each variation of the report
US2.5	I want to add to a report with the variations their rsId, so that I can easily see which variations of the report are known SNPs

B.1.3. Iteration 3

Usage Scenario 3.1. In order to research the diabetes mellitus type II disease, I want to read the genotypes of several samples from a VCF file. I want to annotate the variations with their genes, with all the names of the transcripts that they hit, and with the score and effect of SIFT and POLYPHEN. I want to filter the variations by the diabetes genes "ABCC8, CAPN10,KCNJ11, GCGR, SLC2A2, HNF4A, INS, INSR, PPARG, TCFl2, ADIPOQ, AKT2, PAX4, MAPK81p1, GPD2, MNTR1B", by the "deleterious" variations according to SIFT and "possibly damaging" or "probably damaging" variations according to POLYPHEN. Finally, I want to create a report with the variations main properties, their genes, their transcript names and their Sift and Polyphen predictions.

Usage Scenario 3.2. In order to research the diabetes mellitus type II disease, I want to read the genotypes of several samples from a VCF file. I want to annotate the variations with all the names of the transcripts that they hit and the sample MAF. I want to filter the variations by the diabetes genes "ABCC8, CAPN10,KCNJ11, GCGR, SLC2A2, HNF4A, INS, INSR, PPARG, TCFl2, ADIPOQ, AKT2, PAX4, MAPK81p1, GPD2, MNTR1B", I want to prioritize by the sample Sift [0,0.5] and order it from minimum to maximum. Finally, I want to create a report with the variations main properties, the genes and the Sift prediction.

Usage Scenario 3.3. In order to research the diabetes mellitus type II disease, I want to read the genotypes of several samples from a VCF file. I want to annotate the variations with all the names of the transcripts that they hit and the sample MAF. I want to filter the variations by the diabetes genes "ABCC8, CAPN10,KCNJ11, GCGR, SLC2A2, HNF4A, INS, INSR, PPARG, TCFl2, ADIPOQ, AKT2, PAX4, MAPK81p1, GPD2, MNTR1B", I want to prioritize by the sample MAF [0.1,0.5] and order it from maximum to minimum. Finally, I want to create a report with the variations main properties and the sample MAF.

Table B. 4 User Stories of Iteration 3

US3.1	I want to read several samples' genotypes from a VCF file, so that I can perform several analysis over those samples
US3.2	I want to annotate the patients' variations with the transcripts names (provided by RefSeq) that each variation hits (exons), so that I can see the different transcription patterns that the variation hits
US3.3	I want to annotate the patients' variations with the prediction of the SIFT algorithm score and effect for each variation transcript, so that I can preliminary assess the predicted effect of each variation taking into account to this algorithm
US3.4	I want to annotate the patient's variations with the prediction of the POLYPHEN algorithm score and the effect, for each transcript, so that I can preliminary assess the predicted effect of each variation taking into account to this algorithm
US3.5	I want to annotate the patients' variations with the sample Minor Allele Frequency, so that I can see the frequency of the allele that has minor occurrence in the analyzed samples
US3.6	I want to filter the patient variations by the effect predicted by SIFT (tolerated/deleterious), so that I can see only the variations that pass the filter
US3.7	I want to filter the patient variations by a set of effects predicted by POLYPHEN (benign, probably damaging, possibly damaging, unknown), so that I can see only the variations that pass the filter
US3.8	I want to prioritize the patient's variations by an interval (between 0 and 1) of the sample minor allele frequency and an order,Min2Max or Max2Min, so that I can focus and on the most important variations according to this frequency.
US3.9	I want to prioritize (filter and order) the patient's variations by a range (between 0 and 1) and an order of the SIFT prediction (min2Max, Max2Min), so that I can focus and on the most important variations for the analysis based on this algorithm
US3.10	I want to prioritize (filter and order) the patient's variations by a range(between 0 and 1) and order of the POLYPHEN prediction, so that I can focus and on the most important variations for the analysis based on to this algorithm
US3.11	I want to add the MAF to the variations' report, so that I can see which allele has the minimum frequency and the value of this frequency
US3.12	I want to add to a variation report their SIFT predictions, so that I can see which variations have an effect in codification
US3.13	I want to add to a variation report their POLYPHEN predictions (score and effect), so that I can see which variations have an effect in codification

B.2 Analysis models:

B.2.1. Feature model

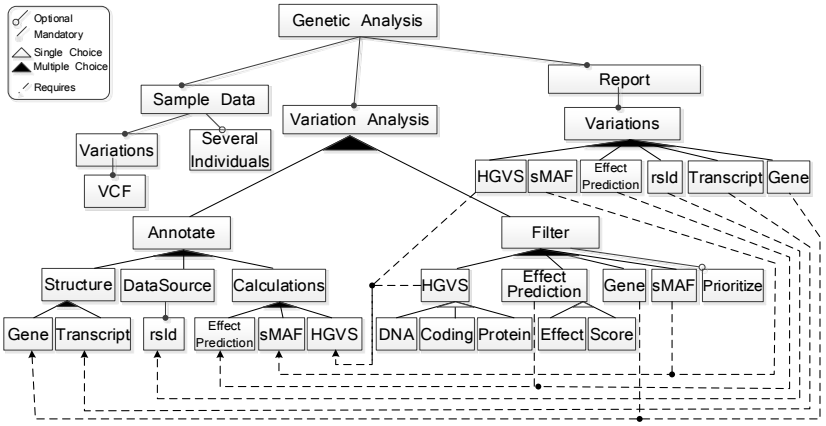


Figure B. 1 Feature Model of the DSL

B.2.2. Concepts model

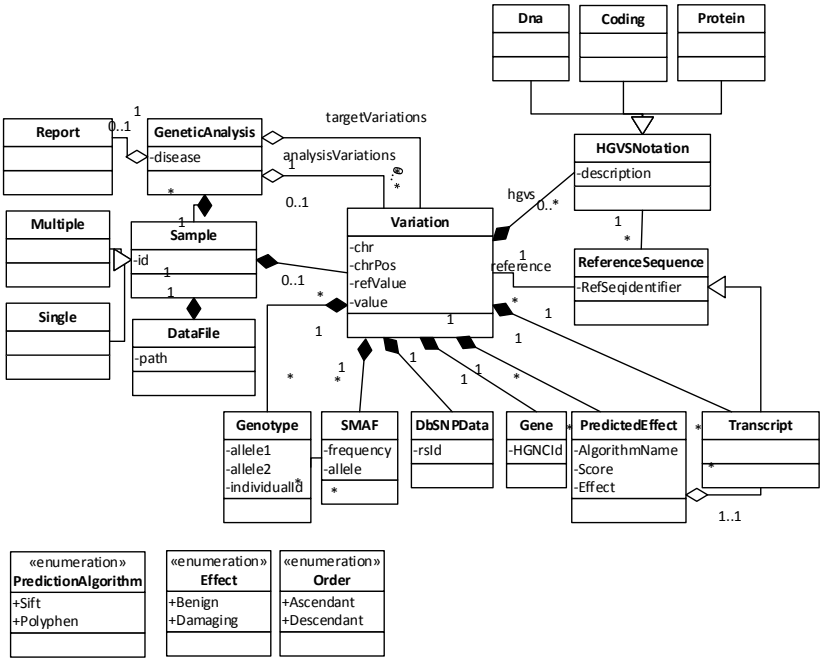


Figure B. 2 Conceptual model of the DSL

B.2.3. Glossary of terms

- **Genetic Analysis**: Analysis that is performed to a sample observing genetic data.
- **Report**: Relevant information gathered as a result of a genetic analysis.
- **Sample**: Object of study to perform a genetic analysis (one or several individuals).
- **Single (Sample)**: When the object of study is a single individual
- **Multiple (Sample)**: When the object of study are several individuals.
- **Datafile**: Genetic data of the sample saved in a textual file.
- **Variation**: Each of the nucleotides that each individual of the sample has different in regards to a reference sequence.
- **Reference Sequence**: A representative sequence of nucleotides that theoretically represents the sequence of a "disease free" human.
- **HGVS Notation**: Standard nomenclature the describe variations
- **(HGVS Notation) DNA**: HGVS Nomenclature that represents the value of the variation at nucleotide level.
- **(HGVS Notation) Coding**: HGVS Nomenclature that represents the value of the variation at the coding level.
- **(HGVS Notation) Protein**: HGVS Nomenclature that represents the value of the variation at the amino acid level.
- **Gene**: Functional unit that delimiters a subset of nucletides from the DNA sequence that is responsible to regulate a function of the body.
- **DbSNPData**: Information from the database of SNPs dbSNP, a reference database in the field.
- **Transcript**: Functional structure of the gene that represents the parts that play a role in the transcription of the nucleotides of the genes to proteins.
- **Predicted Effect**: Result of the execution of a prediction algorithm that assesses the effect of the variation in an individual.
- **Genotype**: Two alleles of an individual in a position in the chromosome.
- **sMAF**: Abbreviation of sample Minimum Allele Frequency. Calculation that represents the allele has the minimum frequency among the individuals of the sample.

B.2.4. Relationships between the concepts model and the feature model

- Feature *VCF*-> Entity *DataFile*

- Feature *Annotate.Calculations.EffectPrediction*-> Entity *PredictedEffect*
- Feature *Filter.Gene*->Entity *Gene*
- Feature *Filter.EffectPrediction*->Entity *PredictedEffect*
- Feature *Filter.sMAF*->Entity *sMAF*
- Feature *Priotitize*->Entity *Interval*
- Feature *Priotitize*->Entity *Order*
- Feature *Hgvs.HgvsDna*->Entity *DNA*
- Feature *Hgvs.HgvsCoding*->Entity *Coding*
- Feature *Hgvs.HgvsProtein*->Entity *Protein*

B.3 Design models:

B.3.1. Concrete syntax grammar

```
   grammar diagnosis.it3.mydsl.MyDiag with
org.eclipse.xtext.common.Terminals
   import "diagnosis"
   import "http://www.eclipse.org/emf/2002/Ecore" as ecore

   geneticAnalysis returns geneticAnalysis: 'Analyze' disease=disease
patientData=patientData analyses+=analysis+ report=report;
   /*PATIENT DATA */
   patientData returns PatientData: 'Read' variations=variations;
variations returns Variations: 'variations' several=severalPatients
 format=vcf;
   severalPatients returns
SeveralPatients:'genotypes'{SeveralPatients};
   vcf returns Vcf:'from' 'a VCF file' datafile=dataFile;

   /*ANALYSES */
   analysis returns Analysis: annotation | search;
   //Variation Annotation
annotation returns Annotation: 'Annotate variations with'{Annotation}
 (hgvs=hgvsA)? (gene=geneA)?(transcript=transcriptA)?
 prediction+=predictionA* (rsId=rsIdA)?;
   //Annotation Fields
   hgvsA returns HgvsA: 'hgvs'{HgvsA};
   geneA returns GeneA: 'gene'{GeneA};
   transcriptA returns TranscriptA: 'transcript'{TranscriptA};
   predictionA returns PredictionA: algorithm=predictionAlgorithm;
   rsIdA returns RsIdA:'rsId'{RsIdA};
   //Variations Filter
```

```
search returns Filter: hgvsS |('Filter variations by' (geneF |
  predictionF))|('Prioritize variations by' (geneF | predictionF)
  order=order);
    hgvsS returns HgvsS: 'Search variations'(dnaS|codingS|proteinS);
    dnaS returns DnaS: hgvsdna+=hgvsdna+;
    codingS returns CodingS: hgvscoding+=hgvscoding+;
    proteinS returns ProteinS: hgvsprotein+=hgvsprotein+;
    geneF returns GeneF: 'gene' gene+=gene+;
    predictionF returns PredictionF: effectF|scoreF;
effectF returns EffectF: algorithm=predictionAlgorithm 'effect'
  effect+=effectEnum+;
scoreF returns ScoreF: algorithm=predictionAlgorithm 'score'
  '['minScore=EDouble','maxScore=EDouble']';
    order returns Order: criteria=orderCriteria;

    /*REPORT */
    report returns Report: 'Report'reportVariations=reportVariations;
    reportVariations returns ReportVariations: 'variations'
{ReportVariations} ('with' (hgvs=hgvsR)? (gene=geneR)? (rsId=rsIdR)?)?;
    hgvsR returns HgvsR: 'hgvs' {HgvsR};
    geneR returns GeneR: 'gene' {GeneR};
    rsIdR returns RsIdR: 'rsId' {RsIdR};
    transcriptR returns TranscriptR: 'transcript'{TranscriptR};
    predictionR returns PredictionR: algorithm=predictionAlgorithm;

    /*DataModel Types */
    disease returns Disease: name=EString;
dataFile returns DataFile: 'from'{DataFile} (dynamic?=INPUT
  |path=EString);
hgvsdna returns HgvsDna: reference=refSeqReference
  ':''g.'description=HGVSEXPR;
hgvscoding returns HgvsCoding: reference=refSeqReference
  ':''c.'description=HGVSEXPR;
    hgvsprotein returns HgvsProtein: reference=refSeqReference
':''p.'description=HGVSEXPR;
    gene returns Gene: hgncId=(EString|HGNCGENE);
    refSeqReference returns RefSeqReference:
identifier=(REFSEQ|ASSEMBLY);

    /* Data Types ecore */
    EBoolean returns ecore::EBoolean: 'true' | 'false';
    EString returns ecore::EString: STRING | ID;
    EInt returns ecore::EInt: '-'? INT;
    EDouble returns ecore::EDouble: '-'? INT '.' INT;

    /*Terminals and Enumerations */
    terminal
HGNCGENE:(('A'..'Z')+((('0'..'9')+('A'..'Z')+)*|('0'..'9')+));
```

```
    terminal INPUT: 'input';
    terminal REFSEQ:'N'('C'|'G'|'M'|'P')'_' INT'.'INT;
    terminal ASSEMBLY: ('Hg'INT) | ('NCBI'INT);
    terminal HGVSEXPR:
   (INT(('+'|'-')INT)?('ins'|'del')('A'|'T'|'G'|'C')+)|//ins/del
   (INT(('+'|'-')INT)?('A'|'T'|'G'|'C')+'>'('A'|'T'|'G'|'C')+)|//indel
   (('A'..'Z'|'a'..'z')+INT('A'..'Z'|'a'..'z')+);//Protein
    enum predictionAlgorithm returns PredictionAlgorithm: Sift='Sift' |
Polyphen='Polyphen';
    enum orderCriteria returns OrderCriteria: AlphAsc='AlphAsc'|
AlphDes='AlphDes'|Max2Min='Max2Min'|Min2Max='Min2Max';
    enum effectEnum returns Effect: Benign=benign| Damaging= 'damaging'|
Tolerated='tolerated'|ProbablyD='probably damaging'|PossiblyD='possibly
damaging';
```

B.3.2. Abstract syntax metamodel

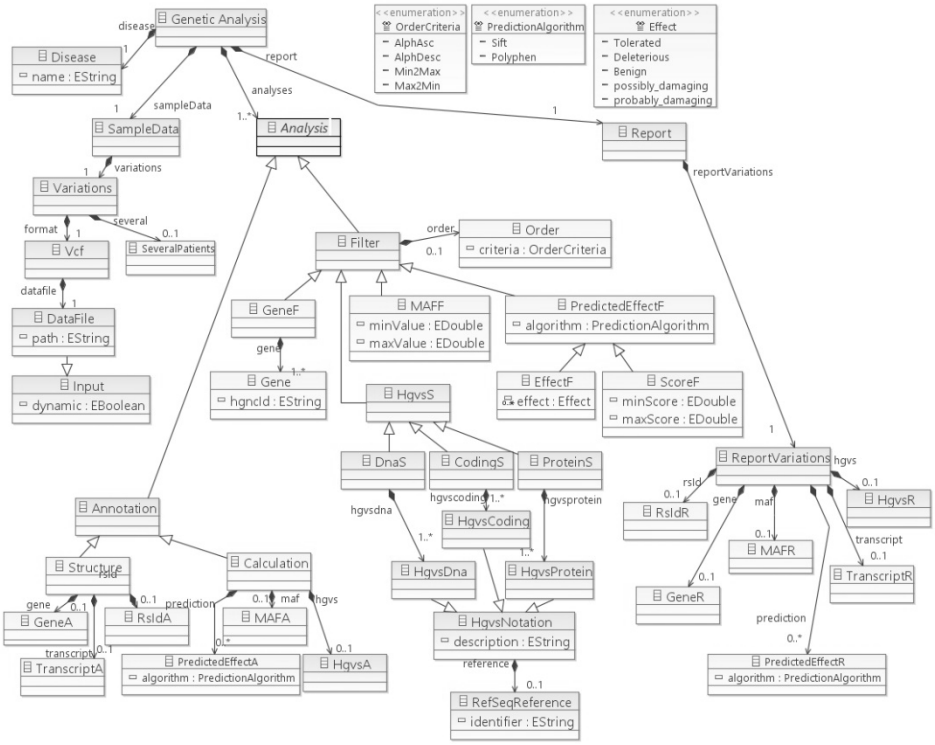


Figure B. 3 Abstract Syntax Metamodel of the DSL

B.3.3. Implementation example

Figure B. 4 and Figure B. 5 are fragments of the complete implementation. These figures show respectively the generator test used to guide the implementation, and the source code that makes this test succeed. The test checks that the generator generates the correct Galaxy workflow fragment associated with the user story "Annotate MAF". The generator contains the corresponding transformation rules in order to create a fragment of a Galaxy workflow that will execute the annotation of the MAF with the tool *allele frequencies*.

The complete implementation of the DSL is uploaded to GitHub in https://github.com/mvillanueva/GeneticAnalysisDSL

```
@Before
def void testSetupOnce() {
DiagnosisPackage.eINSTANCE.eClass();
diagnosis = parser.parse ('''Analyze Diabetes Mellitus Type 2 (Analysis 3)
Read Variations genotypes from VCF file Patient1.vcf
Annotate Variations with gene, transcripts, maf
Filter Variations by genes {ABCC8, CAPN10, KCNJ11, GCGR, SLC2A2,
HNF4A, INS, INSR, PPARG, TCFl2, ADIPOQ, AKT2, PAX4, MAPK81p1,
GPD2, MNTR1B}
Prioritize Variations by MAF [0.1, 0.5] max2min
Report Variations with gene, predicted_effect''')
fsa= new InMemoryFileSystemAccess()
generator.doGenerate(diagnosis.eResource, fsa)
filecontent=fsa.getTextFiles().values().iterator().next().toString()   }
@Test
def testAnnotateMAF(){
Assert.assertTrue("The workflow fragment of AnnotateMAF is different to
the generated one",GeneratorGalaxy.checkGeneratorGalaxy(filecontent,
path.concat("US5AnnotateMAF.ga"), "ensembl_id"))}
```

Figure B. 4 Test that checks the user story "Annotate with sample MAF"

```
 def steps(Resource resource)'''«/*PatientData */»
«var patient=new PatientDataGenerator()»
«patient.readPatientData(resource.allContents.toIterable.filter(PatientData).
get(0))»«
/*Analyses */»
«FOR Analysis a:resource.allContents.toIterable.filter(Analysis)
SEPARATOR ','»
«var analysis=new AnalysisGenerator()»
«analysis.runAnalysis(a)»«ENDFOR»,«
/*Report */»
«var report=new ReportGenerator()»
«report.generateReport(resource.allContents.toIterable.filter(ReportVariation
s).get(0))»'''
def annotateVaritionsWithVCFTools(boolean maf)'''«
var step=galaxy.getLastStep+1»
    "«step»": {
          "annotation": "Annotate MAF",
          "id": «step»,
          "input_connections": {
            "input": {
                "id": «galaxy.getLastWorkflowStep»,
                "output_name": "output"}
           },
    "inputs": [],
         "name": "Allele Frequencies",
         "outputs": [{
                "name": "output1",
                "type": "tabular"
                }, {
                "name": "output",
                "type": "vcf"
                }],
          "tool_errors": null,
          "tool_id": "allele_frequencies",
          "tool_state": "{«
            »\"__page__\": 0, \"input\": \"null\", \"__rerun_remap_job_id__\":
null,«
            »\"mafOption\": \"{«mafTranslator(maf)»}\"«
            »}",
          "tool_version": "latest",
          "type": "tool",
          "user_outputs": []
      }«
def mafTranslator(Boolean maf)'''«
»\\\"mafFieldname\\\": \\\"«maf»\\\",«
»\\\"mafCheckbox\\\": \\\"«IF maf»True«ELSE»False«ENDIF»\\\", «
» \\\"__current_case__\\\": «IF maf»0«ELSE»1«ENDIF»'''
```

Figure B. 5 Source Code of the Generator (Xtend)

Annex C

This annex gathers the extra material of the experiment. Specifically, it gathers the questionnaires that measure the end-user satisfaction and the raw data that has been collected from subjects.

C.1 Questionnaires for measuring end-user satisfaction about mechanisms:

Next, we provide the five questionnaires that were used in the experiment to assess the perceived ease of use and usefulness. Each question had 5 response options: strongly disagree, disagree, neutral, agree and strongly agree. The questions were

C.1.1. Demographic assessment

Q1. My experience in genetic diagnosis is: “None”, “1 year or less”, “Between 1 year and 5 years”, “More than 5 years”.

Q1. My experience in designing languages is: “None”, “1 year or less”, “Between 1 year and 5 years”, “More than 5 years”.

C.1.2. Assessment of the review step (T2)

For each activity described below, indicate the extent to which you agree with the statement:

Q1. "I found difficult to fulfill the activity":

- Reviewing Usage Scenarios that describe diagnosis that the DSL must represent
- Reviewing user Stories about diagnosis steps
- Reviewing acceptance tests that describe examples that validate diagnosis steps that the DSL must support.

Q2. Overall, I found the usage scenarios, user stories and acceptance tests easy to understand.

Q3. "The activity took me the expected time"

- Reviewing Usage Scenarios that describe diagnosis that the DSL must represent
- Reviewing user Stories about diagnosis steps
- Reviewing acceptance tests that describe examples that validate diagnosis steps that the DSL must support.

Q4. "I found the activity useless to provide my knowledge"

- Reviewing Usage Scenarios that describe diagnosis that the DSL must represent
- Reviewing user Stories about diagnosis steps
- Reviewing acceptance tests that describe examples that validate diagnosis steps that the DSL must support.

Q5. Overall, I found that the activities proposed engaged my participation in the DSL development.

C.1.3. Assessment of the syntax questionnaire (T3)

For each activity described below, indicate the extent to which you agree with the statement:

Q1. "I found difficult to fulfill the activity":

- Fulfilling the questionnaire for selecting the most suitable syntax

Q2. Overall, I found the questionnaire about syntax examples easy to understand.

Q3. "The activity took me the expected time"

- Fulfilling the questionnaire for selecting the most suitable syntax

Q4. "I found the activity useless to provide my knowledge"

- Fulfilling the questionnaire for selecting the most suitable syntax

Q5. Overall, I found engaging the questionnaire for selecting my favorite syntax.

C.1.4. Assessment of the semantic templates (T3b)

For each activity described below, indicate the extent to which you agree with the statement:

Q1. "I found difficult to fulfill the activity":

- Fulfilling the service templates to detail the semantics of a user story.

Q2. Overall, I found the semantic templates easy to understand.

Q3. "The activity took me the expected time"

- Fulfilling the service templates to detail the semantics of a user story

Q4. "I found the activity useless to provide my knowledge"

- Fulfilling the service templates to detail the semantics of a user story

Q5. Overall, I found engaging the service templates for describing DSL semantics.

C.1.5. Assessment of the demonstration (T4)

For each activity described below, indicate the extent to which you agree with the statement:

Q1. "I found difficult to fulfill the activity":

- Watching a demonstration to understand the current state of the DSL and the usage of the DSL editor.

Q2. Overall, I found the demonstration easy to understand.

Q3. "The activity took me the expected time"

- Watching a demonstration to understand the current state of the DSL and the usage of the DSL editor.

Q4. "I found the activity useless to provide my knowledge"

- Watching a demonstration to understand the current state of the DSL and the usage of the DSL editor.

Q5. Overall, I found engaging the demonstration for understanding the current state of the DSL and usage.

C.1.6. Assessment of the testing guidelines (T5)

For each activity described below, indicate the extent to which you agree with the statement:

Q1. "I found difficult to fulfill the activity":

- Using the DSL editor to assess the language correctness
- Following the guidelines to assess the language correctness.

Q2. Overall, I found the testing guidelines easy to understand.

Q3. "The activity took me the expected time"

- Using the DSL editor to assess the language correctness
- Following the guidelines to assess the language correctness.

Q4. "I found the activity useless to provide my knowledge"

- Using the DSL editor to assess the language correctness
- Following the guidelines to assess the language correctness.

Q5. Overall, I found engaging the testing guidelines for assessing the DSL correctness.

C.2 Data gathered from questionnaires (end-user satisfaction)

Acronyms: SA=Strongly Agree, A=Agree, N=Neutral, D=Disagree, and SD=Strongly Disagree

C.2.1. Raw data from Google Forms

Table C. 1 Responses about Usage Scenarios, User Stories and Acceptant Tests

	Difficult			**Adequate time**			**Overall (easy)**	**Useless**			**Overall (engaging)**
	USC	US	AT	USC	US	AT		USC	US	AT	
P1	SA	SA	A	SA	SA	SA	SA	D	D	D	SA
P2	D	D	D	A	A	A	A	SD	SD	SD	A
P3	SD	SD	D	A	A	A	SA	N	N	N	A

Table C. 2 Responses about the Syntax Questionnaire

	Difficult	**Adequate time**	**Overall (easy)**	**Useless**	**Overall (engaging)**
P1	D	SA	A	D	A
P2	SD	A	SA	D	A
P3	N	N	A	D	N

Table C. 3 Responses about Semantic Templates

	Difficult	Adequate time	Overall (easy)	Useless	Overall (engaging)
P1	D	A	SA	D	SA
P2	D	A	A	A	A
P3	N	D	SA	SD	SA

Table C. 4 Responses about the demonstration

	Difficult	Adequate time	Overall (easy)	Useless	Overall (engaging)
P1	SD	SA	SA	SD	SA
P2	SD	SA	SA	SD	SA
P3	SD	SA	SA	SD	SA

Table C. 5 Responses about the DSL editor and the assessment guidelines

	Difficult		Adequate time		Overall (easy)	Useless		Overall (engaging)
	Editor	Guidelines	Editor	Guidelines		Editor	Guidelines	
P1	SD	D	SA	A	SA	SD	SD	SA
P2	D	D	SA	SA	SA	SD	SD	SA
P3	SD	D	N	SD	SA	SD	SD	SA

C.2.2. Standardization of responses

Table C. 6 Standardization of Responses about Usage Scenarios, User Stories and Acceptance Tests

	Easy (Opp. Difficult)			Adequate time			Overall (easy)	Useful (Opp. Useless)			Overall engaging
	USC	US	AT	USC	US	AT		USC	US	AT	
P1	SD	SD	D	SA	SA	SA	SA	A	A	A	SA
P2	A	A	A	A	A	A	A	SA	SA	SA	A
P3	SA	SA	A	A	A	A	SA	N	N	N	A

Table C. 7 Standardization of Responses about the Syntax Questionnaire

	Easy (Opp. Difficult)	Adequate time	Overall (easy)	Useful (Opp. Useless)	Overall (engaging)
P1	A	SA	A	A	A
P2	SA	A	SA	A	A
P3	N	N	A	A	N

Table C. 8 Standardization of Responses about Semantic Templates

	Easy (Opp. Difficult)	**Adequate time**	**Overall (easy)**	**Useful (Opp. Useless)**	**Overall (engaging)**
P1	A	A	SA	A	SA
P2	A	A	A	D	A
P3	N	D	SA	SA	SA

Table C. 9 Standardization of Responses about the demonstration

	Easy (Opp. Difficult)	**Adequate time**	**Overall (easy)**	**Useful (Opp. Useless)**	**Overall (engaging)**
P1	SA	SA	SA	SA	SA
P2	SA	SA	SA	SA	SA
P3	SA	SA	SA	SA	SA

Table C. 10 Standardization of Responses about the DSL editor and the assessment guidelines

	Easy (Opp. Difficult)		**Adequate time**		**Overall (easy)**	**Useful (Opp. Useless)**		**Overall (engaging)**
	Editor	Guidelines	Editor	Guidelines		Editor	Guidelines	
P1	SA	A	SA	A	SA	SA	SA	SA
P2	A	A	SA	SA	SA	SA	SA	SA
P3	SA	A	N	SD	SA	SA	SA	SA

C.2.3. Separation of responses per variable

C.2.3.1 Ease of use

Table C. 11 Ease of Use Responses about Usage Scenarios, User Stories and Acceptance Tests

	Easy (Opp. Difficult)			**Adequate time**			**Overall (easy)**
	USC	US	AT	USC	US	AT	
P1	SD	SD	D	SA	SA	SA	SA
P2	A	A	A	A	A	A	A
P3	SA	SA	A	A	A	A	SA

Table C. 12 Ease of Use Responses about the Syntax Questionnaire

	Easy (Opp. Difficult)	**Adequate time**	**Overall (easy)**
P1	A	SA	A
P2	SA	A	SA
P3	N	N	A

Table C. 13 Ease of Use Responses about Semantic Templates

	Easy (Opp. Difficult)	**Adequate time**	**Overall (easy)**
P1	A	A	SA
P2	A	A	A
P3	N	D	SA

Table C. 14 Ease of Use of Responses about the demonstration

	Easy (Opp. Difficult)	**Adequate time**	**Overall (easy)**
P1	SA	SA	SA
P2	SA	SA	SA
P3	SA	SA	SA

Table C. 15 Ease of Use Responses about the DSL editor and the assessment guidelines

	Easy (Opp. Difficult)		**Adequate time**		**Overall (easy)**
	Editor	Guidelines	Editor	Guidelines	
P1	SA	A	SA	A	SA
P2	A	A	SA	SA	SA
P3	SA	A	N	SD	SA

C.2.3.2 Usefulness

Table C. 16 Usefulness Responses about Usage Scenarios, User Stories and Acceptance Tests

	Useful (Opp. Useless)			**Overall (engaging)**
	USC	US	AT	
P1	A	A	A	SA
P2	SA	SA	SA	A
P3	N	N	N	A

Table C. 17 Usefulness Responses about the Syntax Questionnaire

	Useful (Opp. Useless)	**Overall (engaging)**
P1	A	A
P2	A	A
P3	A	N

Table C. 18 Usefulness of Responses about Semantic Templates

	Useful (Opp. Useless)	**Overall (engaging)**
P1	A	SA
P2	D	A
P3	SA	SA

Table C. 19 Usefulness of Responses about the demonstration

	Useful (Opp. Useless)	**Overall (engaging)**
P1	SA	SA
P2	SA	SA
P3	SA	SA

Table C. 20 Usefulness of Responses about the DSL editor and the assessment guidelines

	Useful (Opp. Useless)		**Overall (engaging)**
	Editor	Guidelines	
P1	SA	SA	SA
P2	SA	SA	SA
P3	SA	SA	SA

C.2.4. Calculation of means and ranges

Responses from a participant for the questions that measure the same variable must not oscillate from one side to the other, since they are designed to measure the same concern. For this reason, when we observed that some range values of the responses to be very high, our hypothesis for this situation was a lack of comprehension of some questions. We blame the likert scale labels because during the experiment participants complained about the extra effort they had to do to respond correctly.

As a solution, for each of the concerns with a high range, we identified the values that were outside the central tendency, and we asked end-users to review them. As we can check in Table C. 21, we observed a high range (with value 4), so we asked the corresponding participant (P1) to review the first question. At the end, the participant confirmed that the answers were the opposite of their opinion. As a consequence, we change the values and the range was back to normal (Table C. 22).

Table C. 21 Observation of unexpected values

	Easy (Opp. Difficult)			**Adequate time**			**Overall (easy)**	**Median**	**Range**
	USC	US	AT	USC	US	AT			
P1	1	1	2	5	5	5	5	5	4
P2	4	4	4	4	4	4	4	4	0
P3	5	5	4	4	4	4	4	4	1

Table C. 22 Correction of wrong values

	Easy (Opp. Difficult)			**Adequate time**			**Overall (easy)**	**Median**	**Range**
	USC	US	AT	USC	US	AT			
P1	5	5	4	5	5	5	5	5	1
P2	4	4	4	4	4	4	4	4	0
P3	5	5	4	4	4	4	4	4	1

Since our sample size is small, this observation of ranges and reasoning was our approach to assess the validity and reliability of the results. In future assessments, if the sample is big enough, construct validity and reliability can be assessed by conducting the Chronbach's alpha calculation: calculating the variances between responses and checking that the coefficient is at least 0.7.

C.2.4.1 Ease of Use

Table C. 23 Ease of Use numerical values about Usage Scenarios, User Stories and Acceptance Tests

	Easy (Opp. Difficult)			**Adequate time**			**Overall (easy)**	**Median**	**Range**
	USC	US	AT	USC	US	AT			
P1	1	1	2	5	5	5	5	5	4
P2	4	4	4	4	4	4	4	4	0
P3	5	5	4	4	4	4	4	4	1

Table C. 24 Ease of Use numerical values about the Syntax Questionnaire

	Easy (Opp. Difficult)	**Adequate time**	**Overall (easy)**	**Median**	**Range**
P1	4	5	4	4	1
P2	5	4	5	5	1
P3	3	3	4	3	1

Table C. 25 Ease of Use numerical values about Semantic Templates

	Easy (Opp. Difficult)	**Adequate time**	**Overall (easy)**	**Median**	**Range**
P1	4	4	5	4	1
P2	4	4	4	4	0
P3	3	2	5	3	3

Table C. 26 Ease of Use numerical values about the demonstration

	Easy (Opp. Difficult)	**Adequate time**	**Overall (easy)**	**Median**	**Range**
P1	5	5	5	5	0
P2	5	5	5	5	0
P3	5	5	5	5	0

Table C. 27 Ease of Use numerical values about the DSL editor and the assessment guidelines

	Easy (Opp. Difficult)		**Adequate time**		**Overall (easy)**	**Median**	**Range**
	Editor	Guidelines	Editor	Guidelines			
P1	5	4	5	4	5	5	1
P2	4	4	5	5	5	5	1
P3	5	4	3	5	5	5	2

C.2.4.2 Usefulness

Table C. 28 Usefulness numerical values about Usage Scenarios, User Stories and Acceptance Tests

	Useful (Opp. Useless)			**Overall (engaging)**	**Median**	**Range**
	USC	US	AT			
P1	4	4	4	5	4	1
P2	5	5	5	4	5	1
P3	3	3	3	4	3	1

Table C. 29 Usefulness numerical values about the Syntax Questionnaire

	Useful (Opp. Useless)	**Overall (engaging)**	**Median**	**Range**
P1	4	4	4	0
P2	4	4	4	0
P3	4	3	3,5	1

Table C. 30 Usefulness numerical values about Semantic Templates

	Useful (Opp. Useless)	**Overall (engaging)**	**Median**	**Range**
P1	4	5	4,5	1
P2	2	4	3	2
P3	5	5	5	0

Table C. 31 Usefulness numerical values about the demonstration

	Useful (Opp. Useless)	**Overall (engaging)**	**Median**	**Range**
P1	5	5	5	0
P2	5	5	5	0
P3	5	5	5	0

Table C. 32 Usefulness numerical values about the DSL editor and the assessment guidelines

	Useful (Opp. Useless)		**Overall (engaging)**	**Median**	**Range**
	Editor	Guidelines			
P1	5	5	5	5	0
P2	5	5	5	5	0
P3	5	5	5	5	0

C.2.5. Summary

Table C. 33 Perceived Ease of Use and Usefulness of the method by participants

	Ease of Use						**Usefulness**					
	M1	M2	M3	M4	M5	All	M1	M2	M3	M4	M5	All
P1	5	4	4	5	5	5	4	4	4,5	5	5	4,5
P2	4	5	4	5	5	5	5	4	3	5	5	5
P3	4	3	3	5	5	4	3	3,5	5	5	5	5
Median	4	4	4	5	5	5	4	4	4,5	5	5	5
Range	1	2	1	0	0	1	2	0,5	2	0	0	0,5

C.3 Data gathered by the developer (developers' satisfaction)

C.3.1. Mechanism M1

C.3.1.1 Comprehension:

- Only one question was asked about how to apply M1. Only one subject asked "What do US and AT from the template mean?"
    - Developers' conclusion: Geneticists understood well the mechanism 1.
- Geneticists reviewed acceptance tests with better enthusiasm and celerity, proposing additional tests for each user story.
    - Developers' conclusion: Acceptance tests were the most familiar artefact

C.3.1.2 Agreement degree

- All subjects agreed that one of the user stories was ambiguous. The three geneticists told the developer that the user story "calculate the frequency (MAF)" was not clear because it could be the samples' frequency of the population's
    - Developers' conclusion: The mechanism 1 was useful to detect big mistakes (about requirements).
- Some syntax errors and changes in acceptance tests were only detected by some geneticists.
    - The mechanism 1 was not powerful to detect small mistakes (about requirements descriptions)

C.3.1.3 Undetected errors

- After the DSL released, some geneticists complained that they brainstormed the addition of two new requirements that were not recorded.
    - The mechanism 1 needs to add the use of a product backlog to record new requirements that are important for end-users.

C.3.2. Mechanism M2

C.3.2.1 Comprehension

- None questions about the syntax was made about the four syntaxes provided. Geneticists understood the four syntaxes because they were

four different ways to describe the usage scenario "Analyse Diabetes Mellitus using MAF", an example they understand well

- Developers' conclusion: Mechanism 2 is well understood by end-users because it uses a domain example to reduce the DSL syntax complexity

C.3.2.2 Agreement degree

- Geneticists had to rate each syntax option according to their preferences and choose their favourite. However, there were lots of differences of opinion among geneticists. As a solution, in order to choose the most preferred syntax by geneticists, developers had to weight their ratings and choose the syntax supposedly best rated in general.
    - Developers' conclusion: Mechanism 2 should be improved to ensure a better agreement among end-users.

C.3.2.3 Undetected errors

- One geneticist suggested changing the word "Diagnose" for "Find putative variations" by using the questionnaire. However, other changes like: 1) adding an underscore between the terms "possibly" "damaging" to be "possibly_damaging", or 2) change the enumeration "AlphAsc, AlphDesc, Min2Max, Max2Min" for the enumeration "Ascendent and Descendent", were only proposed by geneticists after the DSL released instead of using the syntax questionnaire.
- Developers' conclusion: Mechanism 2 is not optimal to refine the syntax structure. It should provided a more usable mechanism to gather end-users' feedback about the concrete and abstract syntax.

C.3.3. Mechanism M3

C.3.3.1 Comprehension

- All the geneticists understood correctly all the fields of the template but the field "source description". The three geneticists did not know what information to provide in this field.
    - Developers' conclusion: Mechanism 3 is suitable to gather the details of how to use a technological artefact to fulfil geneticists goals, however, it should be improved to be able to gather information about the source (author, consortium, website, etc.)

from which the technological artefact was created and from which is available.

- One geneticist was not expert enough to provide all the required information of the template
    - Developers' conclusion: Mechanism 3 should be improve to ensure that the end-users who apply the mechanism have enough knowledge to contribute in them.

C.3.3.2 Agreement degree

- Geneticists specified in the semantic template the same information about the technological artefact that implemented the functionality.
    - Developers' conclusion: Mechanism 3 is suitable to gather information about technological artefacts and their specific details

C.3.3.3 Undetected errors

- None error was detected by geneticists in regards to semantics.
    - Developers' conclusion: Mechanism 3 is suitable to gather the correct information about DSL semantics.
- Developers had all the information required to implement the semantics of the DSL
    - Developers' conclusion: Mechanism 3 is suitable to gather all the information required by developers for the implementation of the DSL semantics.

C.3.4. Mechanism M4

C.3.4.1 Comprehension

- None questions were asked by geneticists during the demonstration. Geneticists understood how to use the DSL infrastructure because the demonstration exemplified how to do it with the usage scenario "Analyse Diabetes Mellitus using MAF", an example they understand well.
    - Developers' conclusion: The mechanism 4 is suitable to present the DSL to end-users.

C.3.5. Mechanism M5

C.3.5.1 Comprehension

- One geneticists left two questions unanswered, and another geneticist had doubts about the meaning of another two questions.
    - Developers' conclusion: The mechanism 5 needs to be improved to ensure the comprehension of the questions. The relationship between each testing question with the DSL element under test should be clearer.

C.3.5.2 Agreement degree

- Geneticists agreed in the majority of the testing questions. In general they agreed the suitability of the DSL syntax and semantics, although some of them provided some suggestions to improve the DSL, such as the addition of a new user story "filter by a list of genes from a file" or the correction of syntax elements such as "possibly_damaging", none of these feedback was contradictory.
- Developers' conclusion: The mechanism 5 is suitable to test the general satisfaction of end-users about the DSL release.