Software Architecture Design by Stepwise Model Transformations
A Comparative Case Study

Fabian Gilson and Vincent Englebert
PReCISE Research Center, Faculty of Computer Science, University of Namur, Namur, Belgium

Keywords: Software Architecture, Model Transformation, Design Rationale, Design Decision, Case Study.

Abstract: Software architecture design is a critical task as lots of requirements can be taken into account on which many decisions can be made. The maintenance and evolution of resulting models often become tricky, even impracticable when their rationale is lost. In a previous work, we introduced a set of languages used in a transformation-centric design method meant to tackle this scattering of requirements and to facilitate further model evolutions. But, we did not provided a formal validation of our proposal yet. The present work depicts a comparative case study we conducted on a group of students. The participants were asked to develop an online book store in two phases, the second one simulating an evolution of the system. We evaluated the functional completeness of the created software as well as the traceability of design decisions and rationale. The participants were also asked to criticize the design method and language they used in a textual report and through a questionnaire. Even if the size of the case study is rather limited, it clearly highlighs the advantages of our approach regarding, among others, its expressiveness and decisions traceability.

1 INTRODUCTION

Software engineering methods offer guidelines and tool-support to structure the creation process of software systems. As the complexity of such systems increases, the need for iterative methods has been widely expressed (Bosch and Molin, 1999). In drawing the architecture model of the *system-to-be*, many decisions are taken and a large part of the knowledge resides in the reasons that lead to a particular model (Perry and Wolf, 1992).

Beside, systems must evolve over time in terms of the functionalities and qualities they fulfill and regarding technological frameworks. In order to handle system maintenance and evolution, appropriate documentation and traceability mechanisms are needed (Parnas and Clements, 1986).

An increasing number of companies are moving to *Agile* design methods that offer benefits like time-overrun reduction and a higher developer productivity (Dybå and Dingsøyr, 2008). With faster release frequencies, the amount of design decisions logically increases. Without an appropriate tracing mechanism, the architectural knowledge is often lost, even by the practitioners in charge of the impacted models or code (Tang et al., 2006). However, one of the most crucial piece of information is the link between a requirement and its implementation in a model or in the code (Jansen and Bosch, 2005).

We integrated both aspects into an *architecture framework* (ISO/IEC/IEEE, 2011). On the one hand, we use structural models iteratively enriched through model transformations, and on the other hand, an ad-hoc requirement modeling language with explicit traceability mechanisms for design decisions and rationale. In this comparative case study, we confronted the Domain Specific Languages (DSL) we formalized in our framework to the OMG's SysML^{TM} modeling language (Object Management Group, 2012). We evaluated the feasibility and advantages of using such an integrated framework to build a web-based software from scratch. Twelve teams of two students used one or the other set of languages and we identified that the functional completeness and correctness were higher under our framework. The *decision documentation rate*, *i.e.* the amount of documented design decisions, was also evaluated. During the two phases of the evaluation, we counted an average of *0.96* and *0.94* documented decision under our framework where only *0.35* and *0.32* on the other side. Aside, we conducted a paper-based survey and analyzed feedback reports to capture the participants' feelings regarding the languages they used. This survey highlighted a significant improvement regarding modeling element expressiveness.

The present paper starts with an overview of some

134 Gilson F. and Englebert V..
Software Architecture Design by Stepwise Model Transformations - A Comparative Case Study.
DOI: 10.5220/0005266101340145
In *Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2015), pages 134-145
ISBN: 978-989-758-083-3
Copyright © 2015 SCITEPRESS (Science and Technology Publications, Lda.)

related work in Section 2. Afterwards, in Section 3, we give a quick description of the proposed method and languages. The protocol of the comparative case study is then presented in Section 4. Next, the teams' deliverables and feedbacks are detailed in Section 5. We discuss the outcomes of the case study in Section 6 where we evaluate the research method itself and identify the potential threats to validity. Finally, the research perspectives and conclusions are presented in Section 7.

2 RELATED WORK AND MOTIVATION

Numerous architecture description languages have been proposed like ACME (Garlan et al., 1997) or AADL (Society of Automotive Engineers, 2012), but most of them do not provide decision-making traceability, even if their long-term added-value is widely recognized (Perry and Wolf, 1992; Malavolta et al., 2013).

Many decision and rationale recording methods have been defined, from the precursor *Potts and Bruns* model (Potts and Bruns, 1988) to the viewpoint-based *documentation framework* (van Heesch et al., 2012). van Heesch *et al.* recently conducted a case study on documentation of architecture design decisions and highlighted that explicit recordings of these decisions empower a more systematic exploration of design alternatives (van Heesch et al., 2013). However, all these techniques require to maintain extra models for decision and rationale traceability.

Within model-driven approaches, model-to-model transformation languages play a key role. Hybrid imperative and declarative language, like ATL (Jouault and Kurtev, 2005), usually do not support incremental model transformations with change propagations. Triple Graph Grammars offers such a feature, but makes it difficult to define transformations where operational semantics is needed (Ehrig et al., 2005). Abstract syntax-based techniques require to learn another language to define transformations, such that some extra expertise is needed.

In our approach, we believe in a fully integrated framework where any change in an architecture will be fully traceable as documented model transformations, together with their design rationale. Explored alternatives may also be recorded, again as concrete transformations. Last, reusable solutions, *i.e.* architectural patterns, may be documented as dedicated transformations too, again using the same formalism.

3 DESIGN METHOD IN SHORT

The design method is supported by three DSL. Those languages have been presented in dedicated publications, but we summarize their main principles in the following sections.

3.1 Requirement Listing

A simple language has been defined to list requirements with their design decisions (Gilson and Englebert, 2011b; Gilson and Englebert, 2011a). We focus only on *architecturally-significant requirements* (ASR), i.e, requirements that have *"a measurable impact on the software system's architecture"* (Chen et al., 2013).

A requirement model gathers a list of ASR for a specific architecture model. Every requirement must be assigned to an architectural construct in charge of satisfying it. Listing 1 depicts a simplified ASR model with one functional requirement that will be implemented as a transformation.

```
asrmodel clientserver {
  func SayHello assigned Server {
    description "The Server shall print 'Hello
        World!' to the console.";
    realisation implementSayHello;
    rationale {
      assessment "Functionality is trivial, a
          unique service should make the trick.";
      strength "Very simple implementation with
          unique service without parameters.";
    }
  }
}
```

Listing 1: Simplified ASR model.

Decisions can be taken on requirements, such as *refinements*, *implications*, alternative *selections* or *realization* through model transformations. Any decision must be at least justified by an *assessment*, but can be further refined by other design rationale like *strengths* or *constraints* for example.

3.2 Architecture Description and Transformations

A three-stage architecture description language has been defined to represent the structure of software systems at different levels of abstraction (Gilson and Englebert, 2011b). Listing 2 depicts an empty Client-Server *Definition Assemblage Deployment* (DAD) models.

```
dadmodel clientserver {
  definition {
    componenttype Client { }
    componenttype Server { }
  }
}
```

Listing 2: Simplified DAD model.

The set of transformation rules referred as the *realization* documented in Listing 1, is illustrated in Listing 3.

```
transformationset implementSayHello concerns
     SayHello {
  create interface Hello { sync void hello(); }
  alter componenttype Client{ uses Hello as hello;
       }
  alter componenttype Server { implements Hello as
       hello; }
  create connectortype One2One { mode one2one; }
  create linkagetype from Client.hello to Server.
       hello with One2One;
}
```

Listing 3: Simplified transformations set.

Concretely, a binding is created between the `Client` and the `Server` through their *facets* typed by the `Hello` *interface*.

More modeling elements are available in the DAD language to express a software architecture from *(i)* an abstract *definition*, *(ii)* a *runnable assemblage* with, among other, particular communication protocol constraints, and *(iii)* a *deployment* specification with (user-defined) infrastructure properties.

Listing 4 shows a more complete, though simplified, overview of all stages of a DAD model.

```
dadmodel dadsample {
  definition { // building blocks
    interface Hello { sync void hello (); }
    componenttype Client { uses Hello as h; }
    componenttype Server { implements Hello as h; }
    protocol TCP { layer transport; }
    connectortype Con { mode one2one; }
    linkagetype from Client.h to Server.h with Con;
    nodetype Computer { Ethernet eth; }
    mediumtype RJ45 { supports TCP; }
  }
  assemblage { // concrete instances
    soi c : Client { Client.h as h on TCP; }
    soi s : Server { Server.h as h on TCP; }
    linkage from c.h to s.h with Con;
  }
  deployment { // possible deployment
    node comp[2] : Computer;
    deploy client on comp[0]; open c.h on comp[0]::
        eth;
    deploy server on comp[1]; open s.h on comp[1]::
        eth;
    plug RJ45 from comp[0]::eth to comp[1]::eth;
  }
}
```

Listing 4: All DAD model stages.

Note that all stages in a model are optional so that, for example, reusable building blocks can be specified separately in dedicated library-models or different *assemblage* or *deployment* stages for the same *definition* can be defined.

3.3 A Transformational Approach

The IODASS design approach is transformation-centric, i.e., changes made in the architecture model must be expressed with model transformations (Gilson and Englebert, 2014). As illustrated in Listing 3, it provides rules to create, delete and modify any type of construct. Transformations are always related to a particular requirement. A software architecture is then created in an iterative way where designers start from a model with a list of ASR (possibly incomplete), a set of basic constructs and make successive decisions. Figure 1 depicts the typical design process, very close to Hofmeister's *general model* (Hofmeister et al., 2007).

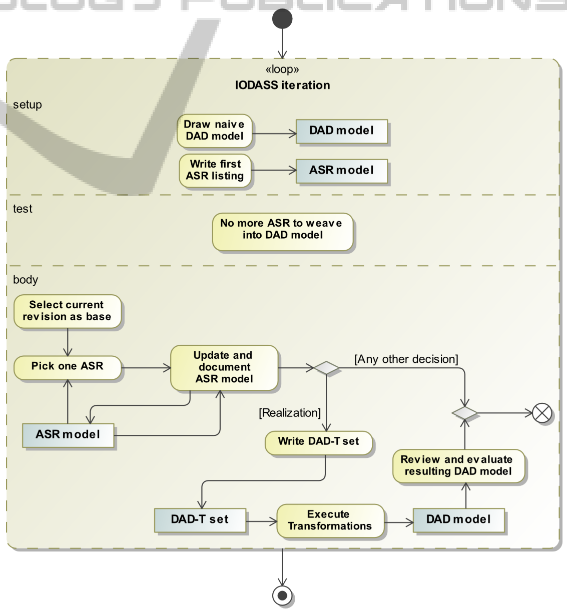

Figure 1: IODASS Design process as a UML activity.

An architecture model is iteratively enriched and documented through model transformations such that all structural changes are kept for later references. Furthermore, any decision is directly encoded into the ASR model and refer, when needed, to a transformations set, such that structural elements are always bound to their related requirements.

Editors for all languages have been implemented as Eclipse plugins with the Xtext[1] framework. Trans-

[1]http://www.eclipse.org/Xtext

formations are run within Eclipse on the models' abstract syntax tree and produce a new model each time. The history of all created models is kept in the workspace to backtrack to previous versions and explore implementation alternatives, if necessary.

4 CASE STUDY PROTOCOL

After a couple of tests on toy examples, we decided to evaluate our approach in a comparative case study. We followed a rigorous process to design the study and conducted it on a group of master students (Pfleeger, 1995; Runeson and Höst, 2009; Wohlin et al., 2012). The study was planned to evaluate *(i) the feasibility and benefits of a transformation-centric approach to build a software system* and *(ii) the expressiveness of our modeling languages and their impacts on model documentation.*

In order to evaluate the feasibility of our design method, we had to compare it to an iterative design process where engineers may pick one requirement at a time, refine or implement it and evaluate the resulting model. As identified by Hofmeister *et al.* in their general model, this iterative process is very common as architecture design method (Hofmeister et al., 2007). Furthermore, it is a somewhat intuitive way of designing a piece of software. Then for the study itself, we required the students from the control group to follow an iterative design process too.

Regarding the languages themselves, we decided to compare them to the OMG's SysML. On top of structural modeling facilities at variable levels of abstraction, requirements, traceability and rationale-related information can be added into SysML models. It allows to define building blocks in a similar fashion as in our architectural language, with infrastructure-related objects too. Also, the participants had no knowledge of both modeling languages, even if they all are familiar with UML modeling, such that previous knowledge in one or the other language would not have biased the study.

Finally, we also had to take care of the development environment, since our languages are implemented inside the Eclipse platform. A couple of SysML plugins also exist in that platform. Thus the participants could work in the same environment as for our framework, which reduces the possible bias regarding the tool's takeover.

4.1 Participants' Properties

The participants were master students of the Faculty of Computer Science at the University of Namur and chosen by convenience. They all have a Bachelor degree that includes all the prerequisite competencies for the experiment concerning (UML) software modeling or iterative design methods. 24 students were involved in the study that was conducted from mid-March until end of May 2013 as part of a course in the Software Engineering option of the Master's curriculum. They all had a previous experience with a mid-scale academic project where they developed a web-based *three-tier* information system from scratch. They all were males between 21 and 25 year old. However, instead of creating random teams, we first conducted a preliminary phase to build comparable groups of students for the control and under-study teams.

4.2 Initial Phase

During this preliminary phase, students had to draw a UML class diagram from a requirement document describing a vehicle inspection system. This document contained an informal description of the system-to-be, the use case diagrams and corresponding scenarios. It also contained clear instructions regarding the expected class diagram to produce in terms of level of details and completeness. These guidances were illustrated by a sample diagram to make everything as clear as possible. During this preliminary round, all students received exactly the same remarks in classroom at the beginning of the lecture and were spread into a room wide enough to minimize cheating possibilities.

4.3 Build Comparable Groups

At the end of the lecture, we gathered the produced models and classified them using a *judging* method inspired from the work of Jones (Jones, 1983): two in-house researchers and one external senior researcher, all familiar with UML modeling, were asked to categorize the diagrams.

First, they individually drew their own quick-draft of the class diagram based on the requirement document. Second, they individually classified the students' models regarding their own criteria. The judges were free to define their own categories as well as the number of categories. After this classification round, it appeared that all judges used comparable criteria and built four categories of diagrams that we can summarize by the following characteristics:

Cat. 1. syntacticly and semantically incorrect

Cat. 2. syntacticly correct, semantically incorrect

Cat. 3. incomplete, correct syntax and semantics

Cat. 4. complete, correct syntax and semantics

We calculated the 2-digits truncated arithmetic mean (denoted *tam*) of the assigned values by the judges for each diagram, i.e, Cat.1 diagrams received a value of 1, Cat.4 a value of 4. Upon this calculation, we had three categories: low ($tam < 2$), mid-range ($2 \leq tam < 3$) and high results ($tam \geq 3$). The mid-range category contained too many elements so, we used external criteria, like their previous results in modeling lectures, to split the category in two.

In order to let a bit of freedom to the students to create pair-teams for the remaining of the case study, we created four *hats* based on the aforementioned categories to equalize as much as possible the modeling competences of the future teams. We fairly distributed the students between the hats assigned to SysML (*S-Teams*) and to our framework (*I-Teams*).

4.4 Case Study Startup

Prior to give the description of the system to develop to the students, they followed separately a 2 hours lecture where SysML was presented to the *S-Teams*, and our framework to the *I-Teams*. Both lectures were organized the same way and given by the same person. Two *screencasts* were realized to explain how to install the plugins in Eclipse and how to create a *helloworld model*. Both screencasts followed the same template and illustrated the same concepts.

The remaining of the study was realized by the participants outside classroom, per teams, except for one session dedicated to general questions and answers and for the final demonstration of their prototypes. The study was organized in two phases, the first one dedicated to the realization of a prototype from scratch, the second one to modify part of the system, with a notable impact at the architectural level. For each phase, the students received a separate document explaining the system requirements only for the concerned phase, so that they had no idea of the nature of the future evolutions. After the first phase, we had a 15-minutes informal discussion with all teams individually to answer their questions and debrief about their deliverables regarding documentation quality and functional correctness.

4.5 Case Description

We specified a fictitious online library system where customers can order books from different book stores. Each time a book is sold, an auction is organized between all stores to determine the actual selling price. When the auction is done, the library contacts the parcel delivery system to pick up the book at the store and deliver it to the customer. For this first phase, the library was leading the overall process. The students were given a document with an informal description of the case study, as well as the list of the expected functionalities and qualities for all three sub-systems, specified in ten very detailed requirements. Some simplifying hypotheses and methodological aspects regarding the iterative design method to follow and the expected deliverables were also specified in that document.

The first document was very precise and the required implementation was almost straightforward in order to let them concentrate on their modeling, documentation and coding activities. For the second phase, the participants received another document with new requirements. They were expressed at a higher abstraction level, though still complete and unambiguous, in order to compare the decision traceability mechanisms of both approaches. They also received detailed guidelines on how to write their own evaluations of the language(s) they used. The new requirements were *(i)* the auction had to be taken in charge by the book stores, *(ii)* the customer could withdraw its purchase and then receives a credit note, and *(iii)* the catalog had to be exposed also as a web service.

At the end of both phases, the teams delivered a set of documented models representing the overall system with the design rationale and the description of the iterative process they actually followed, as required at the beginning of the study. They also had to provide a functional prototype. At the end of the second phase, the teams also submitted an evaluation report on *(i)* the language expressiveness, *(ii)* the documentation and traceability facilities, and *(iii)* the maintainability of the models.

4.6 Evaluation Method

We will now present the results of the case study. We used a *Goal-Question-Metric* approach to define our evaluation criteria and metrics (Basili, 1992) detailed hereafter with the *Purpose/Issue/Object/Viewpoint* template (PIOV).

GOAL 1. *Evaluate the feasibility of a transformational architecture design method to design a software system*

PIOV. Evaluate / the feasibility of iteratively transform a / software architecture model / from the project manager's viewpoint

Question 1. Is it effective to implement a software based on an architecture model created from stepwise formal model transformations?

Metric 1. Number of top-level functionalities correctly implemented

GOAL 2. *Evaluate the quality of architecture and requirement models using DAD-ASR languages*

PIOV. Evaluate / the functional completeness of a / software architecture model regarding the expected model elements / from the architect's viewpoint

Question 2. Does the produced architecture models contain all expected components and interfaces to fulfill the software's requirements?

Metric 2. Number of requirements without any responsible component/part.

Question 3. Are the newly introduced subrequirements correctly documented with their rationale?

Metric 3. Number of decisions regarding subrequirements with a meaningful explanation of their purposes (rationale).

GOAL 3. *Evaluate the traceability of a transformational method regarding the history of the development process (planning-evaluation)*

PIOV. Evaluate / the actual implementation order of the / architecturally significant requirements / from the architect's viewpoints

Question 4. Does all development iterations have been *backlogged* for evaluation and traceability purposes?

Metric 4. Number of iterations reported with corresponding implementation plans.

GOAL 4. *Evaluate the feasibility of a transformational method in maintenance and evolution activities of a software system*

PIOV. Evaluate / the feasibility of iteratively transform an / existing software architecture model / from the architect's viewpoints

Question 5. Is it effective to incorporate new functionalities in a software based on an architecture model modified by stepwise formal model transformations?

Metric 5. Number of impacted functionalities correctly implemented

5 CASE STUDY RESULTS

We now detail the results we gathered from the deliverables produced by all teams.

5.1 Functional Correctness and Quality of Deliverables

At the end of the first phase, we deployed their prototypes, following their `readme` files to test the four top-level functional requirements. To this end, we ordered a book and checked if the auction started and completed as expected, as well as if the delivery was correctly done (happy scenario). We also analyzed the requirements and structural models, as well as the project management documentation they produced to detail their development iterations. Table 1 summarizes the results for all teams. Some columns have been directly linked to the metrics identified in the previous section. We also provide the arithmetic mean values for each groups.

Regarding the test of the functionalities of the happy scenario, our framework produced slightly better results. One more *I-Team* provided a fully functional prototype than the *S-Teams*. Two *S-Teams'* prototypes did not implement correctly any requirement, for one *I-Team*. Moreover, an average of *3* requirements were correctly implemented by the *I-Teams*, where *2.17* for the *S-Teams*. Even if the difference is not significant, we can notice than, except for the *I6-Team*, all other five *I-Teams* implemented at least *3* requirements where three *S-Teams* completed similar functional results.

Concerning the number of produced requirements and their traceability as structural elements, the results show a better result for the *S-Teams*, where only *0.66* requirement remained untraced against *3.33* for *I-Teams*. However, the amount of sub-requirements identified within our framework is significantly higher with an average of *24.16* requirements against *12.5* on the other side. Half of the *S-Teams* listed only the ten requirements found in the case study document and did not refine any of them. This was also the case for the *I3-Team* that did not provide any traceability information or even design rationale.

Similarly to the amount of produced requirements, the amount of design rationale is significantly higher for *I-Teams* than for *S-Teams*. To identify the decisions and their rationale, we analyzed the design reports given by the students, where they were asked to justify the decisions (and alternatives) they made to build the online library system. An average of *12.83* decisions could be found in the models and documents produced by the *S-Teams*, from which *4.5* (*0.35*) were documented. On the other side, *I-Teams* produced an average of *17.5* decisions from which *16.83* were justified (*0.96*). This result is not really surprising since ASR models are meant to receive such pieces of information. But we analyzed the content of the rationale-dedicated constructs to remove *useless* justifications, *i.e.* empty fields or even incomplete or fuzzy justifications that gave no clue on the reasons why the requirement was actually produced.

The last metric we are interested in concerns the

Table 1: Evaluation of the deliverables of phase 1.

Team	Happy scenario	Impl.(M1)	Req.	Untr.(M2)	Decis.	Rat.(M3)	Iter.(M4)
S1	Stopped during auction	2	19	1	17	10	4
S2	No possibility to order	0	15	2	11	2	0
S3	Fully functional	4	10	0	10	1	0
S4	Fully functional	4	10	0	11	8	3
S5	Stopped after auction	3	11	1	16	4	0
S6	No possibility to order	0	10	0	12	2	3
MeanS	n/a	2.17	12.50	0.66	12.83	4.50	1.66
I1	Fully functional	4	23	1	15	15	1
I2	No book delivery	3	30	1	28	28	5
I3	Fully functional	4	10	10	0	0	0
I4	Stopped after auction	3	14	2	9	8	8
I5	Fully functional	4	41	3	29	29	4
I6	Compilation failure	0	27	3	24	21	5
MeanI	n/a	3	24.16	3.33	17.50	16.83	3.83

Table 2: Evaluation of the final prototypes and deliverables.

Team	Feedback	Auction	Credit	Impl.(M5)	Req.	Untr.(M2)	Decis.	Rat.(M3)	Iter.(M4)
S1	Medium	Fully	Incomplete	5	23	2	19	12	0
S2	None	Partially	Complete	3	17	1	13	2	0
S3	Medium	Fully	Incomplete	5	15	0	13	4	0
S4	Basic	Fully	Complete	5	14	0	24	7	0
S5	Basic	Fully	None	3	15	1	18	5	0
S6	Basic	Partially	Complete	4	18	1	16	4	2
MeanS	n/a	n/a	n/a	4.16	17	0.83	17.33	5.66	0.33
I1	Medium	Fully	Complete	6	27	3	24	24	1
I2	Basic	Fully	Complete	5	35	1	31	31	3
I3	Medium	Fully	Complete	6	17	14	3	3	1
I4	Basic	Fully	None	3	17	3	11	11	2
I5	Basic	Fully	Complete	5	56	5	51	48	3
I6	Complete	Fully	Complete	7	38	4	34	28	7
MeanI	n/a	n/a	n/a	5.33	31.66	5	25.66	24.16	2.83

number of iterations needed by the participants to implement the system. As already mentioned, the initial document also contained explicit methodological guidances about the iterative process the participants had to follow. Since all of the students had a previous experience with a SCRUM-based development project (Schwaber and Beedle, 2001), they were already familiar with small iteration steps for software development. However, we could not retrieve any information regarding the implementation order of the functionalities for three *S-Teams*. On the other hand, two *I-Teams* showed similar results (*I1* recorded one iteration and *I3* that provided again not a single piece of information). We can note that the number of iterations are quite similar for all teams that actually reported on that aspect, but one more *I-Team* documented their development life cycle.

At the end of the second phase, we evaluated the functional correctness of their prototypes in a different way. Each team had a 10 minutes slot to demonstrate the full scenario, from the book purchase to the withdrawal at the delivery. We especially evaluated *(i)* the user feedback, *(ii)* the new auction mechanism, *(iii)* the withdrawal with credit note and *(iv)* the catalog as a web service. Table 2 summarizes the results of this second phase[2]. Each new requirement received a value corresponding to the amount of sub-requirements that should have been produced at least.

The completeness of the user feedback was evaluated in order to check whether the structured way of writing requirements had an impact on how they specified end-user functionalities. No clear requirement was expressed to this end, but we were curious to evaluate if our stringent process produced more exhaustive lower-order requirements. The ratings are *None=0* (no user feedback at all), *Basic=1* (very few details), *Medium=2* (most of the expected details are shown) and *Complete=3* (the summary page contains all book and customer details).

The new auction mechanism was obviously tested as it was a major rework at architecture and implementation levels. This rework was playing the role

[2]Note that all teams implemented correctly the web service, so we do not show it on Table 2. Also, note that the M4 metrics is reused, but concerns only the iterations of the second phase, that could have been isolated easily.

of an evolution activity needed to fulfill a new non-functional requirement that the participants had to translate into multiple lower-order functional requirements. Above the evaluation of modeling and evolution facilities, this aspect was a major indicator of the feasibility and effectiveness of our framework. The auction process is either *Partially=1* (responsibility transfered, but still done synchronously), or *Fully=2* implemented (asynchronously with a callback from the store).

Unlike the change in the auction mechanism, the withdrawal requirement was almost isolated from the existing functionalities, i.e., less intrusive at the architecture level. The objective was to check whether adding functionalities to an existing product was straightforward or not. The credit note values are *None=0*, *Incomplete=1* (some details are missing) and *Complete=2* (every customer details are present in the note).

The overall amount of requirements implemented successfully is very close for both groups, with an average of one more requirement correctly implemented by the *I-Teams*. Qualitatively, we may note that all six *I-Teams* implemented correctly the modification of the *Auction* mechanism, which was more intrusive and used as an evaluation criterion for GOAL 4, where four *S-Teams* provided the same functional completeness for that requirement. However, even if this result is promising, the difference is not significant enough to claim our framework lead to a more efficient system design.

The same tendencies as during the first phase are observed concerning the requirement traceability, design decisions and rationale. The amount of identified requirements increased in a comparable manner in both groups, confirming a significantly more systematic decomposition of requirements under our framework. The proportion of documented decisions by meaningful rationale stays at a very high level too (*0.94*).

Last, the explanation regarding the iterative process dropped significantly for the *S-Teams* where only one team actually reported the design steps they followed.

5.2 Paper-based Survey

In order to complement the above metrics, we wanted to gather the participants' *feelings* about the case study and the modeling languages they used with a paper-based survey. Participants had to answer individually and anonymously to encourage them to freely express their opinions. The survey evaluated *(i)* the language expressiveness, *(ii)* the added value of the language (implicitly regarding UML, as it is the most common general purpose modeling language they learn here at the university), *(iii)* the evolution capabilities and *(iv)* the documentation support for model evolution. Table 3 lists the questions we asked to the participants.

Table 3: Questionnaire.

1.	The languages constructs allow to represent:
1.a.	the expected functionalities of the system.
1.b.	the technological and communication constraints.
1.c.	the physical constraints related to the deployment.
1.d.	the non-functional requirements.
2.	The modeling language coupled to an agile development method as the one used during this laboratory offers an added value :
2.a.	to manage the complexity of the system-to-be.
2.b.	for the traceability of the requirements in terms of functionalities to implement.
2.c.	for the correctness and completeness of the implementation (code) of the system.
3.	The structural constructs impacted by a modification of a requirement can be identified quickly.
4.	The structural constructs impacted by a modification of a requirement can be identified at a glance.
5.	The language offers the necessary constructs and mechanisms to write an accurate documentation.
6.	The written documentation allows to efficiently comprehend the system within the framework of a modification of the system.
7.	During the second phase:
7.a.	a major work was necessary to re-understand the architectural concepts of the system.
7.b.	the modeling languages eased the structural changes linked to the new functionalities to implement.

We used a non-graduated scale only indexed by *fully disagree* (0) and *fully agree* (5) marks, inspired from (Krosnick and Presser, 2010). The participants could draw a line wherever they estimated it appropriate. This technique suits particularly when comparing different approaches because answers are expressed on a continuous interval that can be measured with a ruler. It also lets more freedom to the respondents and usually avoid them to backtrack to previous answers because they want to order them within related questions. Besides, we dissimulated two very close questions (Q3 and Q7b.) that concern the evolution phase to evaluate more precisely the impact of our framework on model maintenance and evolution tasks. Figure 2 shows the average rankings given by the respondents.

The collected answers show notably better results with our framework regarding constructs expressiveness (Q1), especially for communication facilities. The ratings for the added value and evolution capabilities are very close, so we cannot say anything on that. Model evolution seems a bit more complicated in our approach since the impacted elements can be identified almost as rapidly than for SysML diagrams

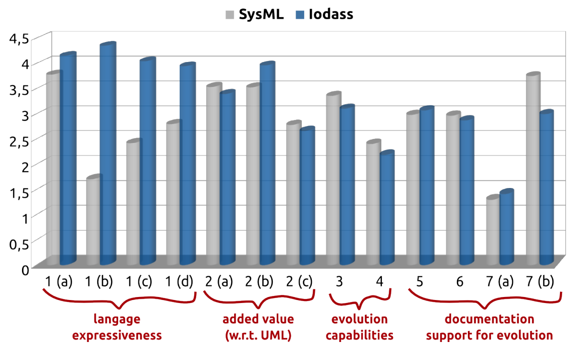

Figure 2: Results of the questionnaire-based survey.

(Q3), but requires more effort to be identified (Q4, Q7).

But, for all questions except Q7a. that was formulated in a dissimulated *negative* manner and Q4, the IODASS ratings are upper to *2.5* which is a rather satisfactory result.

5.3 User Remarks

In their own evaluation reports, the participants expressed interesting remarks and part of them help to interpret more accurately the values we observed in the survey.

All *S-Teams* pointed the excessive freedom offered by the SysML *block* construct, even if this flexibility is interesting in some cases. They were often puzzled on how to concretely refine the semantics of a block to enhance the model comprehension and when to stop the BDD-IBD refinement process. This is particularly visible in the differences expressed in the related questions in the survey (Q1a. to Q1c.). We can reasonably think that the participants are more comfortable with semantically rich modeling constructs.

The *S-Teams* were disappointed regarding the design decisions and rationale traceability mechanism, partially because of a bug in some Eclipse builds used by some students. For a part of them, their version of Obeo Designer[3], yet a commercial plugin, was not displaying requirement *satisfy tables* correctly[4]. However, the rationale construct was working correctly, but none of the *S-Teams* used it. We objectively could not include in our survey specific questions to evaluate the decision tracing mechanism since it would have been biased in favor of our framework.

Our prototype was not preserving the comments in the models after a transformation. All *I-Teams* noted that it was particularly annoying for comprehension and maintenance purposes since they consider comments as an important piece of information and a first-class entity for model documentation. Also, many *I-Teams* suggested to develop a visual layer for architectural models. They reported to have had some difficulties during the first phase with textual models because they were not used to such a representation. They all were experienced with graphical syntaxes, mainly UML models. These remarks partially explain the lower score of our framework in Q7b., and to a wider extend, their difficulties in textual-based model comprehension.

[3]http://marketplace.obeonetwork.com

[4]Even if the version we tested prior to the study was working correctly, the problem was randomly present across operating systems and Eclipse builds.

6 DISCUSSION

We will now summarize the observations we made and discuss the results regarding our four goals introduced at the beginning of Section 5. We will afterwards evaluate the protocol itself.

We decided to discuss the results of the study and survey outside a statistical framework because we did not have a statistically significant sample since the amount of software engineers is very large (which requires then a large sample) and we were conducting the study over students (which limits the generalization possibilities). We then preferred to stick to objective metrics and discuss only over very large differences.

6.1 Evaluation of the Approach

As a first goal, we wanted to evaluate the feasibility of our approach. After both phases, the functional correctness (M1 and M5) was slightly higher for the *I-Teams*. The changes in the *Auction* mechanism was correctly implemented by the six *I-Teams*, against four *S-Teams*. Two more *I-Teams* also implemented the *Credit Note* with the expected level of details. Even if the small amount of involved teams does not allow us to claim the difference is statistically significant, we may claim our stringent design framework does not have a negative impact on the functional correctness and may probably have some impact on requirement elicitation.

The second goal concerned the quality of DAD and ASR models regarding their completeness (M2) and traceability of design decisions. (M3). The amount of unrelated requirements is higher in our framework than in the produced SysML models. These unrelated requirements were mostly the ones that were not related to other requirements or formally linked to a model transformation. However, in an ASR model, a requirement is always *assigned* to a structural element which is equivalent to the *satisfy* relation in SysML. Since this feature is mandatory in ASR models, we looked for other types of formal relations, where the SysML *satisfy* relationship was considered as an explicit link and counted accordingly. Strictly speaking, an ASR is always related to a structural element, but we wanted to evaluate more complex relationships. which explains these rather high values of unrelated requirements (*0.07*) for the first and *0.15* for the second phase).

About the traceability of design decisions (M3), the results are significantly higher. In average, *0.96* and *0.94* design decisions were documented, where the *S-Teams* documented an average of *0.35* and *0.32* of their decisions. Moreover, the extraction of the decisions and their rationale was a very tough analyzing task to recover them, usually from free text justifications. The number of recorded decisions is also a bit higher for the *I-Teams* which probably indicates a more structured way of thinking under our framework.

Last, we wanted to evaluate the reporting of the history of the design process with the number of iterations necessary to draw the architecture model (M4). Even if we explicitly asked for such details, the number of *S-Teams* that provided their development plan dropped at only one team for the second phase. We could not find any explicit or implicit explanations in the final reports from the *S-Teams* and we can reasonably think they did not implement the whole second phase at once, such that part of the design history is lost.

6.2 Threats to Validity

We now discuss relevant threats to validity (Cook and Campbell, 1979; Wohlin et al., 2012).

Internal Validity

First, the selection of the participants was not performed randomly. We used a control group with a comparable, well recognized, modeling language. However, the aforementioned bug in the Obeo plugin may have played a role in the motivation of the *S-Teams* which has lead to a lower documentation rate.

We especially took care of acting fairly between teams, without favoring one or the other. Some students may have been too gentle, or harsh regarding our method. The impact of this *good-looking* effect is hardly identifiable and must be kept in mind aside our conclusions.

The extraction of design decisions and their rationale from textual reports could have lead to lower values. Some of the decisions were explicitly stated, but not all of them. We carefully analyzed multiple times these reports, but we cannot ensure all decisions were actually counted. However, with the many report re-readings, it is fairly unconceivable we missed so many decisions or rationale that the difference would have been insignificant. Furthermore, we were particularly careful regarding that aspect in the conclusions we drew.

Construct and External Validity

Focusing on students represents a significant threat to external validity. They all were students from the university, but with different backgrounds and experiences. They were almost equally coming from the first and the second year of the Master's degree. During the preliminary phase, we particularly paid attention to build comparable groups regarding their former experiences and modeling skills.

The size of the case study is another threats, even if the case sounds realistic, and the final prototype was evaluated by all authors in a *client demo*-like setup. However, since it involved various technologies, analysis and programming skills, we may reasonably consider a valid extrapolation of our results to junior analysts.

Conclusion Validity

Because the amount of participants is rather small, we intentionally did not use a statistical test, but focused on metrics where the differences in the results were very high. Our main goal was to evaluate the feasibility of the approach, *i.e.* pieces of software produced with models written within our framework were *as functional as* the ones produced within a comparable iterative method with a recognized modeling language.

The other aspect we wanted to evaluate was the frequency of documented design decisions and the SysML modeling language offers enough constructs to record this kind of details, even if the analysis was mainly a manual extraction from their reports.

7 CONCLUSIONS

The present work detailed a comparative case study that evaluated the feasibility of a transformation-centric architecture design method. The study was conducted on a group of students from the University of Namur as part of a Software Engineering course. The participants were separated in two groups and composed pair-teams to develop a library system in two phases. Half of the teams used SysML models, the other half used a set of languages we defined in previous publications. Both teams were required to follow an Agile method and to document the design rationale and decisions during the project. We tested their prototypes after each phase and evaluated the quality of the produced models. We also conducted a paper-based survey in order to collect their feedbacks in a structured way.

At the sight of this study, it appears that our transformation-centric approach produces software with a slightly higher rate of functional correctness and completeness (one more group delivered a fully-functional prototype). Also, the amount of justified design decisions was significantly higher (*0.94* against *0.32* documented design decisions for the second phase). Some participants even took the opportunity to add more details than the only mandatory information. However, the study was rather small, so the scalability of our approach should be definitely evaluated on a bigger project[5].

A critical feature requested by the participants concerned a graphical editor, combined to the textual syntax, in order to enhance model visualization. Combined textual and graphical representations of the same model, but focusing on different aspects, may be a very effective tool support and should be investigated.

ACKNOWLEDGEMENTS

The authors thank Nicolas Genon, Quentin Boucher (*PReCISE*), and Xavier Le Pallec (*LIFL, France*) for their help and advices. The authors are also grateful to the Master students of the University of Namur for their kind participation in this case study.

REFERENCES

Basili, V. R. (1992). Software modeling and measurement: The goal/question/metric paradigm. Technical report, University of Maryland at College Park, College Park, MD, USA.

Bosch, J. and Molin, P. (1999). Software architecture design: Evaluation and transformation. *IEEE Int. Conf. on the Engineering of Computer-Based Systems*, pages 4–10.

Chen, L., Ali Babar, M., and Nuseibeh, B. (2013). Characterizing architecturally significant requirements. *Software, IEEE*, 30(2):38–45.

Cook, T. D. and Campbell, D. T. (1979). *Quasi-Experimentation: Design & Analysis Issues for Field Settings*. Houghton Mifflin Company.

Dybå, T. and Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(910):833 – 859.

Ehrig, K., Guerra, E., Lara, J. D., Lengyel, L., Prange, U., Taentzer, G., Varro, D., and Varro-Gyapay, S. (2005). Model transformation by graph transformation: A comparative study. In *Proc. of Model Transformation in Practice Workshop*, MoDELS '05, pages 71–80.

Garlan, D., Monroe, R. T., and Wile, D. (1997). Acme: An architecture description interchange language. In *Conference of the Centre for Advanced Studies on Collaborative research (CASCON 97)*, pages 169–183, Toronto, Ontario.

Gilson, F. and Englebert, V. (2011a). Rationale, decisions and alternatives traceability for architecture design. In *Proc. of the 5th European Conf. on Software Architecture (Comp. Vol.)*. ACM.

Gilson, F. and Englebert, V. (2011b). Towards handling architecture design, variability and evolution with model transformations. In *Proc. of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, pages 39–48. ACM.

Gilson, F. and Englebert, V. (2014). A domain specific language for stepwise design of software architectures. In *Proc. of the 2nd Int'l Conf. on Model-Driven Engineering and Software Development*, pages 67–78. SciTePress.

Hofmeister, C., Kruchten, P., Nord, R. L., Obbink, H., Ran, A., and America, P. (2007). A general model of software architecture design derived from five industrial approaches. *J. Sys. Softw.*, 80(1):106–126.

ISO/IEC/IEEE (2011). Systems and software engineering – architecture description. ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000).

Jansen, A. and Bosch, J. (2005). Software architecture as a set of architectural design decisions. In *Proc. of the 5th Working Conf. on Software Architecture*, pages 109–120, Washington, DC, USA. IEEE Computer Society.

Jones, S. (1983). Stereotypy in pictograms of abstract concepts. *Ergonomics*, 26(6):605–611.

Jouault, F. and Kurtev, I. (2005). Transforming models with ATL. In *Model Transformations in Practice (MTIP) Workshop at ACM/IEEE 8th Int'l Conference on Model Driven Engineering Languages and Systems*.

Krosnick, J. A. and Presser, S. (2010). Question and questionnaire design. In Marsdenand, P. V. and Wright, J. D., editors, *Handbook of Survey Research, Second Edition*, pages 263–313. Emerald Group Publishing Limited.

Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., and Tang, A. (2013). What industry needs from architectural languages: A survey. *IEEE Trans. Softw. Eng.*, 39(6):869–891.

[5]The detailed protocol and analysis may be found on http://info.unamur.be/∼fgi

Object Management Group (2012). OMG Systems Modeling Language (OMG SysML^{TM}), version 1.3. OMG document formal/2012-06-01.

Parnas, D. L. and Clements, P. C. (1986). A rational design process: How and why to fake it. *IEEE Trans. Software Eng.*, 12:251–257.

Perry, D. E. and Wolf, A. L. (1992). Foundations for the study of software architecture. *SIGSOFT Software Engineering Notes*, 17(4):40–52.

Pfleeger, S. L. (1995). Experimental design and analysis in software engineering. *Ann. Softw. Eng.*, 1:219–253.

Potts, C. and Bruns, G. (1988). Recording the reasons for design decisions. In *Proc. of the 10th Int. Conf. on Software Engineering*, pages 418 –427.

Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empir. Softw. Eng.*, 14(2):131–164.

Schwaber, K. and Beedle, M. (2001). *Agile Software Development with Scrum.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

Society of Automotive Engineers (2012). Architecture Analysis & Design Language (AADL). Standard number AS5506 Revision: B.

Tang, A., Ali Babar, M., Gorton, I., and Han, J. (2006). A survey of architecture design rationale. *J. Syst. Softw.*, 79(12):1792 – 1804.

van Heesch, U., Avgeriou, P., and Hilliard, R. (2012). A documentation framework for architecture decisions. *J. Syst. Softw.*, 85(4):795 – 820.

van Heesch, U., Avgeriou, P., and Tang, A. (2013). Does decision documentation help junior designers rationalize their decisions? a comparative multiple-case study. *J. Syst. Softw.*, 86(6):1545 – 1565.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer-Verlag.