Design framework

3TU. School for Technological Design

STAN ACKERMANS INSTITUTE

Design framework : redesign and new multi-user and testing support

Navaraj Neupane

2015

TU Delft
Delft University of Technology

TU/e Technische Universiteit Eindhoven University of Technology

UNIVERSITY OF TWENTE.

Design Framework

Redesign and new multi-user and testing support

Navaraj Neupane

Arash Shafiei

September **2015**

Design Framework

Redesign and new multi-user and testing support

Navaraj Neupane

Arash Shafiei

Eindhoven University of Technology

Stan Ackermans Institute / Software Technology

Partners

Embedded Systems Innovation by TNO

Eindhoven University of Technology

Steering Group Hristina Moneva
Rudolf Mak

Date September 2015

Document status Public

The design described in this report has been carried out in accordance with the TU/e Code of Scientific Conduct

Contact Address
Eindhoven University of Technology
Department of Mathematics and Computer Science
MF 7.090, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
+31402474334

Published by
Eindhoven University of Technology
Stan Ackermans Institute

Printed by
Eindhoven University of Technology
UniversiteitsDrukkerij

ISBN
ISBN

Abstract

The use of models to conceptualize systems is an important part of the process of building Cyber Physical Systems. While designing such systems, which are in general a multi-disciplinary activity, multiple designers are involved in the design decisions. Those decisions most likely are not captured and eventually forgotten after a period. The Design Framework is a visual modeling tool that aims to help architects and designers to documents the design rationales besides the design artifacts. It also helps them to collaborate to design a system together in a multidisciplinary environment. The Design Framework is at the level of a good prototype, but it is not ready for operational application by end-users in industry. One of the main issues with the Design Framework system is a sub-optimal code structure due to the lack of proper design and development approach. The assignment therefore is to reverse engineer the current design of the Design Framework and to come up with a new design. In order to maintain a system in use, presence of a test framework is necessary. Since the Design Framework is used in a multi-disciplinary environment, an improvement in the multi-user support of the system is also needed. In the first part of this report, the redesign of the Design Framework is discussed. To redesign the Design Framework, a number of refactoring techniques are applied. As a result, the code complexity is reduced, therefore the maintenance is increased. The second part of the assignment includes multi-user support and testability. The Design Framework manages the changes to design descriptions and maintains the history of the design artifacts. In this respect, it operates similar to version control systems. In the multi-user part of this report, the version controlling aspect of the Design Framework is described and synchronization of data for multi-user is elaborated. Finally some multi-user features are improved and developed. In the testability part of this report, the test support is described. A set of unit tests and end-to-end tests including the test for multi-user support is implemented. Provided test sets and the approaches used to setup test environment makes the Design Framework more stable and maintainable.

Keywords
systems engineering, visual modeling tool, design process, multidisciplinary design, rationale tracking, refactoring, version controlling, multi-user environment, testing

Preferred reference
Navaraj Neupane, Arash Shafiei, Design Framework: Redesign and new multi-user and testing support. , SAI Technical Report, September 2015. (ISBN)

Partnership
This project was supported by Eindhoven University of Technology and TNO-ESI.

Preface

This technical report presents the results of the project on redesigning and adding new multi-user and test support for the Design Framework. This project was carried out as a final assignment for the Professional Doctorate in engineering degree in Software Technology program at Eindhoven University of Technology as a part of the larger Octo+ project. Octo+ is a joint project between the Embedded Systems Institute (ESI) and Océ-Technologies B.V (Océ) to streamline the use of models in the design process.

In this report the redesign and development of new test and multi-user support for the Design Framework presented. Chapters 2 and 3 give an insight about the domain and problem analysis and introduce the Design Framework. For readers interested in the redesign of the Design Framework Chapters 4, 5, and 6 are the most relevant. Multi-user support is addressed in the chapter 7 and testing support in chapter 8. For those only interested in the goals and results of the project Chapters 1 and 9 provide a sufficient overview.

Navaraj Neupane

Arash Shafiei

September 2015

Acknowledgements

A project is never done in isolation. This project is not an exception, without the help and guidance of several people it could never have been done.

First, I would like to heartily thank my supervisors, Dr. Rudolf Mak and Ir. Hristina Moneva PDEng for their guidance, feedback, and discussions during the entire project. From their different perspectives they helped this project to be successful, I got chance to improve each time after their feedback.

This project would not have been possible without TNO-ESI. I would like to thank them for providing me this project to complete my final and most important OOTI phase. I am thankful to all the people who I met in this company during the nine month tenure. My special thanks to Peter Vink and Marc Willekens for their help regarding technical details of the system.

I would like to express my special gratitude to Ad Aerts for his valuable input and guidance during the two years of OOTI. I heartily thank Maggy de Wert and Lut van Kollenburg for their support during my time in the Netherlands.

I would also like to express a very special gratitude to all my OOTI colleagues for making the past two years an incredible experience. My special thanks to my friend Arash Shafiei for a very fruitful collaboration during the final project.

Last but not least, I would like to thank my parents and siblings for their unconditional support and love. I want to dedicate this success to my entire family.

Thanks.

Navaraj Neupane
September, 2015

I would like to thank my two advisors Dr. Rudolf Mak and Ir. Hristina Moneva PDEng with whom I had the most contact during this project.

I had many meaningful conversations with Dr. Rudolf Mak. These conversations not only clarified the subject and connected the dots together but also gave me more insight on how to model a system using formal mathematical modeling so that it can be usable in broader contexts. He was also always giving ideas on how to see things in different perspectives.

Hristina Moneva was the person with whom I had many conversations to clarify the subject and to understand the requirements in a deeper level so that it can be of use. Having an eye on all aspects of the project, she was helping so that my view does not narrow to only my part of the project.

I would like to thank two other advisors who left the project earlier. Roelof Hamberg supported many ideas and Bas Huijbrechts gave good advices at the beginning of the project.

I would like to thank my friend Navaraj Neupane with whom we had a good collaboration. He was very helpful and he was helping us to organize the process of development.

I would like to thank Ad Aerts who was organizing the program and supporting us so that we have a smooth project.

I would like to thank all friends, and colleagues, especially Pelagia and Favio with whom I could talk without the fear of being judged and other colleagues for their fruitful feedbacks and opinions.

Last but not least I would like to express my special gratitude my family without whom nothing would be possible.

Arash Shafiei

September 2015

Executive Summary

The Design Framework is a system modeling tool. This tool has a number of functionality which helps the architects and designers to document architectural knowledge.

The Design Framework is not only used for documenting design artifacts, but also for capturing the design rational during the design process. It also allows the designers in a multidisciplinary environment to deal with heterogeneous models.

The Design Framework is at the level of a good prototype. However due to the lack of a proper design and development approach, the code structure is sub-optimal and needs to be redesigned.

The goal of this project is to reverse engineer the code of the Design Framework in order to document the current architecture. From the current architecture, a new architecture is to be proposed and implemented.

In order to increase the maintainability of the system, a test framework is also needed to be developed to validate the new architecture.

Since the Design Framework is used in a multi-disciplinary environment, designers collaborate in order to achieve a proper design, therefore the functionality of the Design Framework needs to be improved in order to better support the multi-user functionality.

The assignment is split in two parts. The first part includes redesigning the Design Framework which is done in a group of two persons. The second part of the project includes supporting test and multi-user. This part is done as individual assignment. Arash took the responsibility for multi-user support and Navaraj took the responsibility for the testing support.

In the first part of the project, the current architecture of the Design Framework is analyzed. A number of design redundancies and inconsistencies are found. A new data model is proposed and a number of refactoring techniques are applied.

The refactoring techniques includes techniques that allow for more abstraction, techniques that break the code apart into more logical pieces and techniques for improving names and location of code.

As a result of the refactoring part the code complexity is reduced and therefore the maintainability is increased. The measure for the reduction of the code complexity is mainly the number of lines of code which is reduced by 25%.

In the multi-user support part of the project, the Design Framework is compared to the current version control systems. Like all version control systems which are meant to manage the changes of documents, the Design Framework documents the architectural knowledge.

The requirements for the Design Framework are engineered and a number of flaws in the current multi-user support are detected. The model for the version control aspect of the Design Framework is proposed and improvements are made on the legacy code. As a result of this section, the designers can synchronize their works together and merge their changes.

In the testing support part of the project, the scenarios are analyzed and a number of unit tests are developed.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Project Overview

The Design Framework (DF) is an approach and a visual modeling tool that helps to link all design activities to concrete design artifacts and to track consistency of these artifacts in a multi-disciplinary environment. It is designed to help architects and designers to develop complex systems. In the current practice of designing complex systems, like embedded or Cyber-Physical Systems (CPS), designers face the challenge of balancing the abilities to create a variety of models for analyzing system options and to track their role in the decision process. The DF accomplishes this mission by capturing the design rationales in the design process and by providing a mechanism for using heterogeneous models.

The DF project is carried out as a part of the Octo+ project. It is a joint project between the Embedded Systems Innovation by TNO (TNO-ESI) and Océ-Technologies B.V (Océ) that is carried out as a final assignment of the Software Technology designer program at the Eindhoven University of Technology (TU/e). This project is based on the existing prototype built using state-of-the-art web technologies such as HTML5 and PHP5.

TNO-ESI collaborates in an open innovation structure with a wide range of industrial and academic partners, helping its partners to stay ahead of the innovation curve and lead innovations in embedded systems technology. TNO-ESI strives to improve the current state of the design and development of industrial practices by advocating the use of models in every aspect and variety – from analysis to synthesis, from communication and exploration to specification, from informal to formal, from mono- to multi-disciplinary.

There are two types of projects defined at TNO-ESI – fundamental research projects as well as applied research projects. The DF project started in the context of fundamental research projects in 2009. The basic model underlying the DF tool was defined in [1] and few prototypes based on that model were made. The initial prototype was developed using Eclipse modeling tools. Later on, usability research was done about the functionality as well as the graphical user interfaces and a web-based prototype was proposed. The initial prototype was developed by a group of students who applied the new user interaction ideas. Later on, the tool continued to be used by a few customers in the industry, mainly Océ.

1.2 Stakeholder Analysis

In this section we identify the stakeholders for the DF project. According to ISO 42010 standard, stakeholder is an individual, team, organization, or classes thereof, having an interest in a system. In this section, we analyze the roles of stakeholders of DF project.

Stakeholders were identified in three organizations: TNO-ESI, Océ, and Technical University of Eindhoven.

Figure 1 illustrates an overview of all stakeholders of the DF project grouped based on the organization to which they belong.

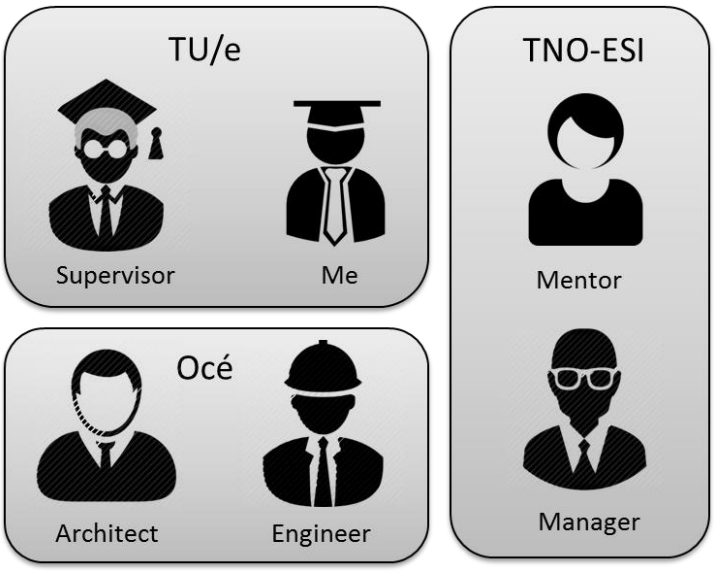


Figure 1 Stakeholder Analysis Overview

Table 1 lists the responsibilities of these stakeholders as well as their relationship with the DF.

Table 1 Stakeholder roles and responsibilities

Stakeholder	Responsibility	Relation to the DF
Me	Redesign, improve, and extend the functionality of the DF	Develop (Analyze, design, and implement) the system
TU/e Supervisor	Supervise the DF project from TU/e perspective	Provide suggestions for a consistent design and quality report
Project Mentor	Supervise the DF project from TNO-ESI perspective, provide requirements and validate results	Provide suggestions for a consistent design, implementation and complete documentation
Project Manager	Keep track of the progress of the DF project in order to use it in a broader context.	Update other stakeholders at Océ about the state of DF
End user (Software architect, designer, engineer)	Design a system at Océ using DF	Define system design, document design decisions, capture design rationales

1.3 Project goals summary

The current development level of the DF tool is that of a good prototype, but it is not mature enough for operational application by end-users in industry. In the meantime extra functionality has been added along the lines of quite global and implicit architectural patterns. For that reason an architectural evaluation is required including both the software design and the technology choices done. Based on the outcome of that evaluation, a decision is to be made on how to proceed with the tool development.

To accommodate future use of the DF methodology, a tool design and implementation is needed that is both acceptable for end-users from industry to be applied as part of their daily work and conduct further research in this direction. This means it must be easily testable and maintainable as well as easily extendible with new concepts and features.

To address this, we start with refactoring phase and later worked in additional features such as multi-user support and testability.

2 Domain Analysis

This chapter provides a detailed analysis of the domain.

2.1 Architectural knowledge

In the context of designing a Cyber-Physical System (CPS) such as MRI scanners and printers, designers and architects face the challenge of documenting their architectural knowledge [2]. Architectural knowledge consists of design description and design decisions.

Design descriptions contain information about design artifacts, their relations and their properties. Design decisions give the rationales about how the system is designed. The design descriptions are usually found in the specifications and design documents (if any). The architectural decisions however, usually remain hidden in the head of the architect.

According to Kruchten et al. [2], there are four types of architectural knowledge:

- **Implicit and undocumented**: The architect is unaware of the decision. It is probably a decision which is taken unconsciously without the architect realizing it. It includes the architect's previous experiences.
- **Explicit and undocumented**: The architect does not document because of some policies and time constraints.
- **Explicit and explicitly undocumented**: The architect might not document because of some tactical reasons, such as protecting his/her position in the company.
- **Explicit and documented**: The architect knows the reason and documents it. This is the preferred, but quite likely exceptional, situation.

The forth case is very rare. Knowing or storing a rationale behind the design decision makes it easier for future understanding, is the main motivation behind the DF project.

Now the question is why documentation is so hard and what we can do to facilitate the architect to document more. For capturing an architectural design, there are plenty of tools available, such as AADL, UML, and SysML [3]. However these tools do not provide a way to capture design decisions as well. Although they have tried to do so, the tools have become so complicated to use for the designers and architects that they have been hardly used.

Therefore, a simple tool was necessary for some companies, to not only document their designs but also their design rationales. Some of these companies which are targeted by TNO-ESI are Océ-Technologies, Philips Healthcare, and Vanderlande Industries. The DF is an approach and a tool that helps to link all design activities to concrete design artifacts and to track consistency of these artefacts in a multi-disciplinary environment.

After studying the concepts behind the requirements, a few prototypes of the DF were implemented. The latest prototype, which uses web technologies, is currently being used at Océ. However each time the customer requests or proposes a new functionality, the code is being extended without thorough consideration of the non-functional requirements such as maintainability, extendibility and readability of the code. As a consequence, the code-base has grown fast without a well-defined architecture.

2.2 Design Framework

The DF is an approach and a tool that helps to link all design activities to concrete design artifacts and to track consistency of these artefacts in a multi-disciplinary environment [4].

Design activities include design decisions, asking questions about and providing answers related to a design, and experiments on the models of design. In general a design activity is the process that the designer carries out. A design artifact is any object created by the designer, such as a system blocks, a parameter, a models, and even the system itself.

A multidisciplinary environment contains a group of people composed of members with varied but complimentary experience, qualifications, and skills who contribute to the achievement of the organization's specific objectives.

The DF aims to help system architects, designers, and engineers to describe a system design and the rationales which lead to its design.

2.2.1 Design Framework Functionality

The functionality of the DF includes:

- Capture the design rationale throughout the product lifecycle phases
- Relate mathematical equations (constraints) and their analyses to concrete design problems
- Deal with partial modeling
- Deal with heterogeneous modeling tools
- Validate design parameters and their dependencies
- Provide a multi-disciplinary system overview through multiple views

Each of these functionalities is explained in detail below.

Capturing the design rationale throughout product life-cycle phases

Quite often the reason for which a system is built in a certain way is not documented. Sometimes the original assumptions, which were valid at the moment when the system was built, are not valid anymore. Therefore, this architectural knowledge remains implicit most of the time. It becomes especially troublesome when people leave a project and they cannot make all design rationales explicit. The main reason behind storing design rationales is for the traceability.

Architecture rationale according to ISO 42010 [5] records the explanation, justification or reasoning about architecture decisions that have been made. The rationale for a decision can include the basis for a decision, alternatives and trade-offs considered, potential consequences of the decision, and citation to sources of additional information.

For example, in a car company a new GPS system is required. Each software architects proposes several solutions and explains the rationale behind his/her design. One of the proposals is accepted and the reason behind this decision is captured. Later on, if the same or another architect wants to change the GPS system, he/she can use this knowledge. Storing a rationale behind each alternative decision provides future architects or related stakeholders to make rational decision by not repeating the same mistakes.

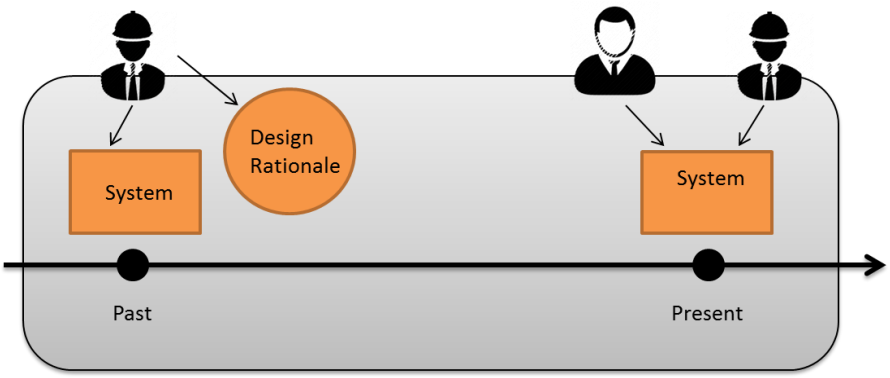


Figure 2 Design rationales are usually lost

Relate mathematical equations (constraints) and their analysis to concrete design problems

The design activities and design artifacts are related, and the DF model takes this into account. Designers use a number of equations to solve their design problems.

For example, in order to design a better engine in a car, a set of equations between the parameters of different components must be analyzed. Engineers want to know the force applied by a car when it crashes with another car in traffic. In this case, they can get the mass and the acceleration of the car and calculate the force and analyze the impact through their design.

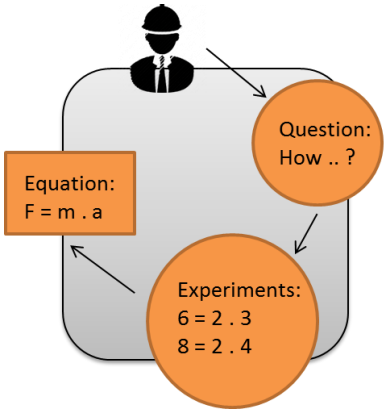


Figure 3 Relate equations to their design problems

Deal with partial modelling

A system is never modeled completely. In fact, quite often only a small part of the functionality is covered (for example the critical part, or the new part). Any information available is useful in DF once it is related to other information and/or a rationale; it does not have to be complete.

For example when designing a car engine, sometimes it is impossible to design the whole system, but only modeling parts of the system (for example the critical parts) suffices.

Deal with heterogeneous modeling tools to deal with constraints

Many modeling languages and tools exist and users use the tool that fits the needs of the problem domain best. For example, some designers use Microsoft Excel for modeling, while others use Matlab. By keeping restriction on the format of models to a minimum, DF makes it possible to use heterogeneous models.

For example, an engine designer might use Matlab because of the complexity of the equations he is dealing with, while a tire designer only uses Microsoft Excel.

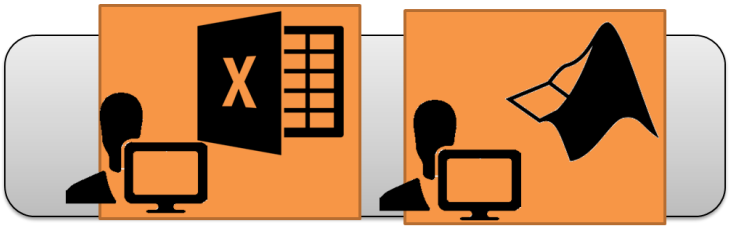


Figure 4 Using Matlab and Excel as modeling tools

Validate design parameters and their dependencies

One of the benefits of dependencies between design information is that they can be used to check or maintain consistency of the system. For example, parameters of a system in a design can be dependent and this dependency can be cascaded.

For example, while designing a car; the designer of the tire should be aware of the changes that the designer of the engine is making. If the weight of the engine increases significantly, then the weight of the car increases. The increase in weight of car gives more pressure to the axels and eventually the pressure should be supported by tires.

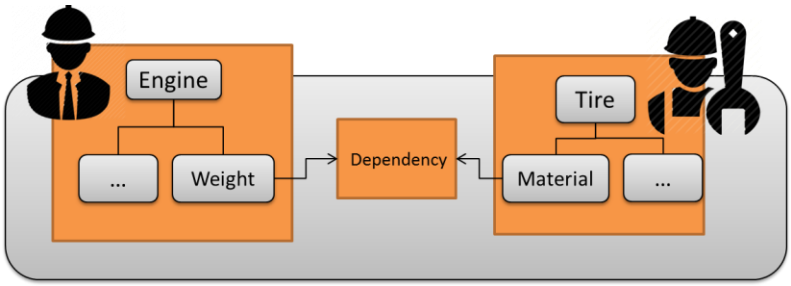


Figure 5 Dependency between the material of the tire and the weight of the engine

Provide multi-disciplinary system overview through multiple views

The views in the DF allow each architect to create an overview of the system that addresses some system aspects (for example performance or safety) while still making explicit to other disciplines.

In the example of car design, mechanical engineers, software engineers, chemical engineers, and so on are working on the same system (car). They not only should be able to see each other's changes but also they have to be able to customize their view to see the system from their own perspective. In the Figure 6, an electrical engineer is interested in engine, and a software engineer is interested in GPS system of the car

which should be deployed in engine. Similarly, mechanical engineer is interested in tire, whose manufacturing depends upon the weight of the engine and the weight of the car. All of these engineers are dependent on each other and it is very important to have proper communication between them. Providing them with a multidisciplinary view of the same system will help them to collaborate effectively.

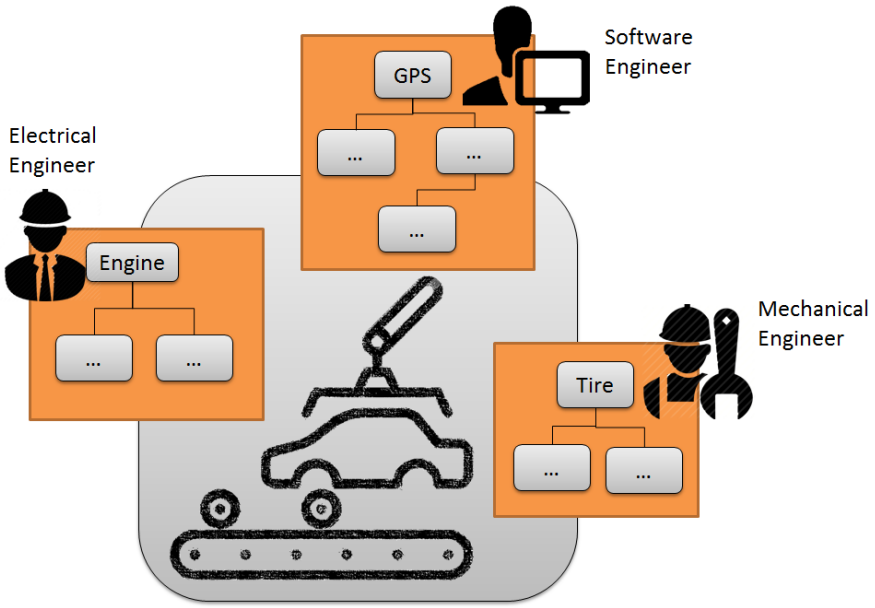


Figure 6 Multidisciplinary system overview of a car manufacturer

2.2.2 Design Framework Model

Figure 7 represents the main concepts and building blocks of DF model. In the DF, three levels are identified for both the design activities and the design artefacts. Each level incorporates the process and status aspects.

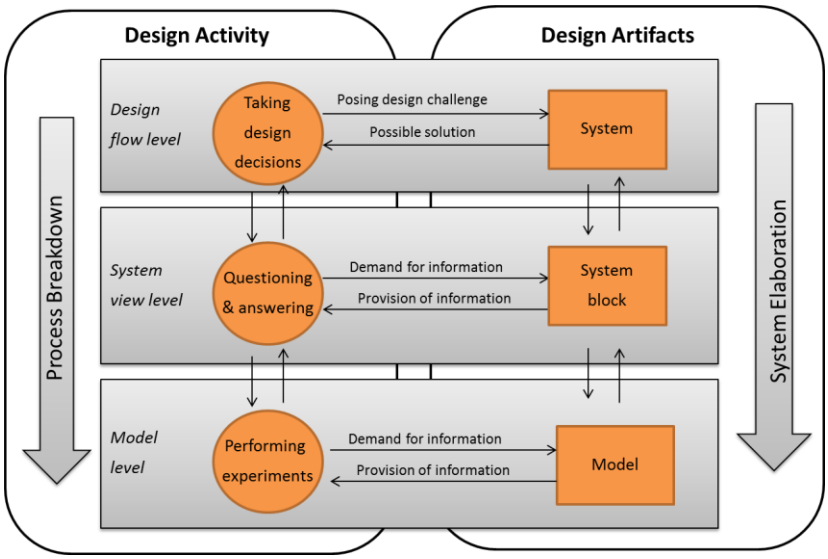


Figure 7 Building blocks of a DF model

In order to support all design activities in a design process, a generic Design Framework model was proposed [4], as shown in Figure 8.

The Design Framework consists of three abstract levels or layers: design flow, system views, and models. Each level incorporates the process and status aspects. The framework is designed to be generic and to be able to fit in any existent design process or development life cycle, any discipline or specific view, and to enable the usage of any formalism required in the working process.

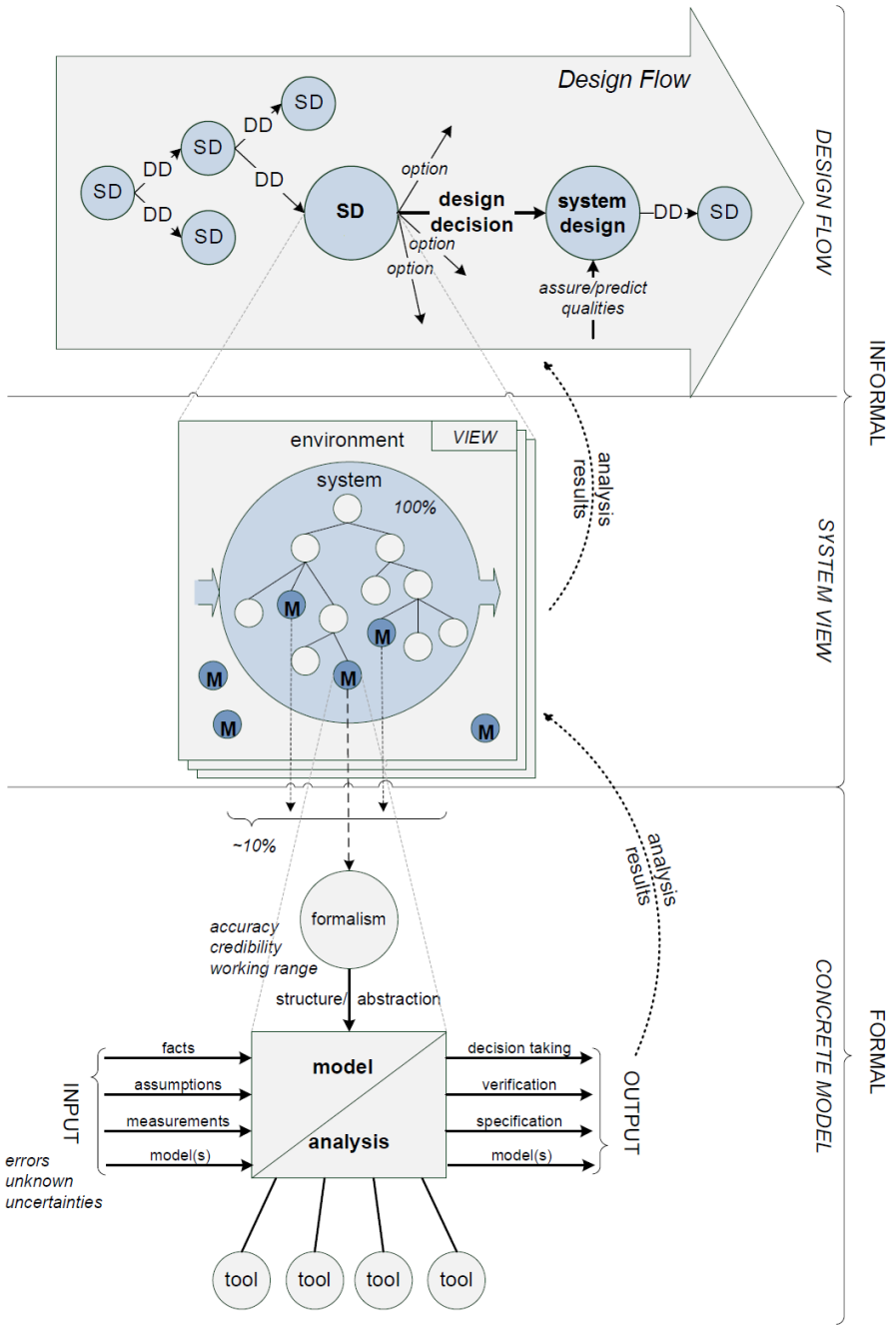


Figure 8 DF model

The three layers are described in the next section.

2.2.2.1 Design Flow level

Designing as an activity can be described by two main ingredients – taking a design decisions and, based on it, refining the designed system until the moment in time that a set of options enforces the next design decision to be taken.

If we try to represent this process, it will be tree-like and composed of a number of nodes (system designs) and edges (design decisions or options) at design flow level. Often, some design activities do not lead to the desired system: either dead branches/dead ends are reached or some possibilities are not explored further. In the cases where the design activities lead to the desired system, the designers work on a few options in parallel in order to gain better in-depth insight in particular aspects of each option.

2.2.2.2 System View level

A view is a representation of a whole system from the perspective of a set of concerns (IEEE 1471, 2000). Views are themselves modular (hierarchy of system components) and might therefore contain one or more models (of some components).

A design view may be considered as a specific representation of the system. It might represent a discipline, e.g., software, hardware, mechatronics, electronics, and material flow, or a specific aspect, e.g., performance or safety. Every view bears a unique decomposition of the system under design. Basic structural blocks are used for the decomposition. They form the skeleton of the view. These blocks may contain or may be contained in other blocks.

The basic blocks are used as a container of the detailed models. The blocks are formalism-independent and consider the model from a black box perspective. One block may contain multiple models, each of which is developed in order to analyze a specific concern or quality of the system or parts of it.

Every block is characterized with a set of parameters. Each parameter might have a (range of) value(s) and a unit, and may have dependencies to other parameters. The parameters are used as a mechanism to couple blocks either within one or between multiple views.

2.2.2.3 Model level

The need for modeling arises from the need of gaining in-depth knowledge of the system under design or of its parts. Each model is expressed in its own formalism, which is suitable for analysis of specific aspects/questions. The model, at model level usually has a number of inputs: a set of facts, assumptions, measurements, or even other models due to model transformations.

In general, multiple experiments can and will be performed on one model. The type of experiment is limited by the set of tools and their abilities. The decision to store the results of a particular experiment is under control of the designer. With any experiment, data about the tool that is used should be stored, along with its parameters, and the results. The results may be used for further decision taking, verification of assumptions or system qualities, or specification of some system parts.

2.2.3 DF editors

The DF tool consists of two editors:

- Flow editor: This editor deals with architectural decisions. The Flow editor represents the Design Flow layer and holds the elements defined in this layer. The Flow editor is used to edit the following semantic elements: Questions, Proposals, System Design, and Design Decision.

- View editor: This editor deals with architectural design. It represents the System View layer and holds the elements defined in the view and model layers. The View editor is used to edit the following semantic elements: Block, Parameter, Model, Relation, Transformation, Validation, Image, Text, Unit, and Representation Type.

2.2.4 DF keywords

The DF provides the user (system architect) with a set of keywords to build a system specification. These keywords are explained in this section.

View editor

The main difference between the DF and SysML is that it is much simpler to learn. The DF also does not support behavioral modeling as opposed to SysML. The numbers of concepts are minimal and it satisfies the need of the industry in the Brainport area.

- View: Views frame specific system concerns for multidisciplinary architects. They are the equivalent of viewpoint in ISO 42010.
- Block: It is a system entity. Block is also defined in SysML.
- Parameter: Parameters hold a value and they specify the properties of a block. In SysML constraint parameters are defined for the same purpose.
- Model: The model is a constraint block that holds mathematical equations. The equivalent in SysML is constraint block.
- Relation: It defines a simple relation between blocks. It does not have any effect on the parameters and properties of the blocks but only indicates that there is a relationship. The relation is between blocks. It is the equivalent of binding connector in SysML.
- Dependency: It is a kind of relationship and the only difference is that it is between parameters and not blocks. There is also always a model (constraint block) attached to a dependency. It is the equivalent of dependency in SysML.
- Transformation: It is a kind of dependency that has several inputs and several outputs.
- Validation: It is a kind of dependency which has only a Boolean output which is true or false.
- Experiment: By assigning values to parameters in a model (constraint block), several outputs are generated. A set of assignments is called an experiment.
- Image: An image can be assigned to a block or experiment to illustrate a fact.
- Text: A text also can be assigned to a block for a further explanation.
- Unit: A parameter can have a standard unit.
- Representation type: Blocks and relations can be represented in different ways (colors and font). This is called a representation type.

Flow editor

The concepts in the flow editor do not exist in SysML. The DF differs with SysML in that is it captures design decisions as well. Following set of concepts are used in flow editor:

- Question: A question usually is imposed on a system asking to add a functional or non-functional requirement.
- Proposal: To address a question, a set of proposals is proposed.
- Accept: A proposal can be accepted by a higher level manager or architects or through a consensus.
- Decline: A proposal can be declined or rejected by a higher level manager or architects or through a consensus. The reasoning behind this decision is captured and can be tracked in the flow editor.

2.3 Diagrams in SysML

There are many types of diagrams described in SysML. The four relevant to the DF are:

- Structure diagrams
- Behavior diagrams
- Requirement diagrams
- Parametric diagrams.

The DF has very closely related concepts for modeling a system except that it does not divide its diagrams into four types. In DF, there is only one type of diagram. This diagram is able to represent all but behavioral diagrams of SysML.

The parametric diagram is intended to support system analysis (performance, reliability, etc.) by defining constraint blocks. A constraint block expresses a mathematical equation and its parameters, some of which may correspond to system block properties. Whereas the diagram in DF not only models system blocks but also binds the values to the parameters of the blocks. All of these constraint blocks, parameters and related dependencies can be viewed together in the DF, which makes it simpler and understandable.

Requirements may be modeled in a separate dedicated or distributed view through the design in the form of parameters and text to each block. The DF validation mechanism provides always up-to-date state of the requirements. As opposed to SysML, in the DF the requirements can be quantified.

Figure 9 shows a comparison between SysML structure and parametric diagrams and DF diagram. For simplicity we did not get into details of the diagram.

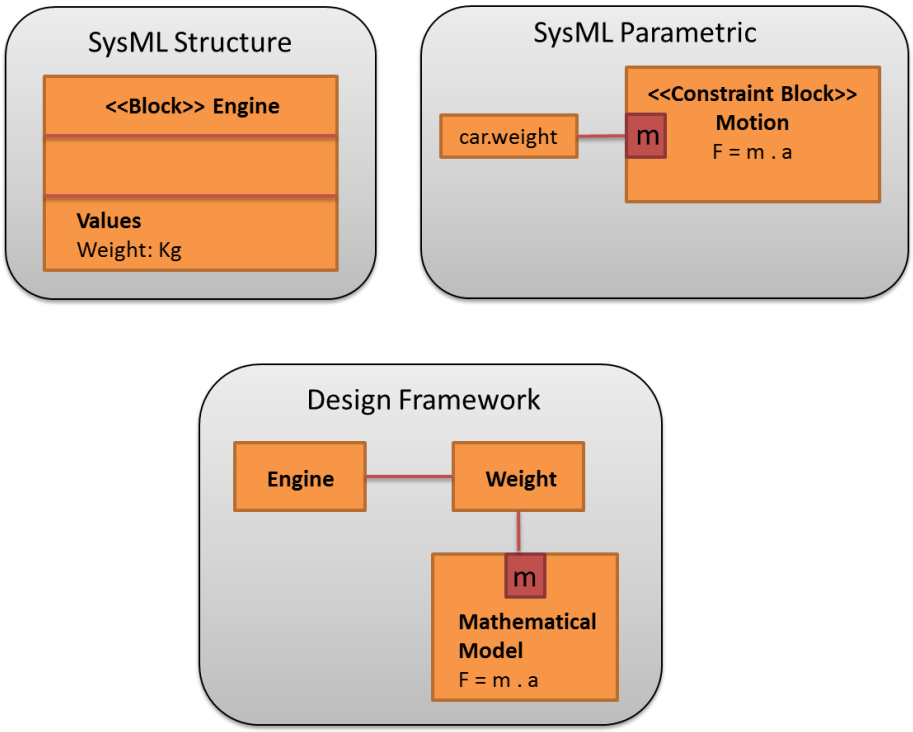


Figure 9 Compare DF with SysML

3 Problem Analysis

In this chapter, we first elaborate the problem statement and we elaborate the project goals.

3.1 Problem statement

To accommodate future use of the DF methodology, a tool design and implementation is needed that is both acceptable for end-users from industry to be applied as part of their daily work and used to conduct further research in this direction. This means it must be easily testable and maintainable as well as easily extendible with new concepts and features. In order to achieve this, the high-level process to further develop the DF, shown in Figure 10, was proposed:

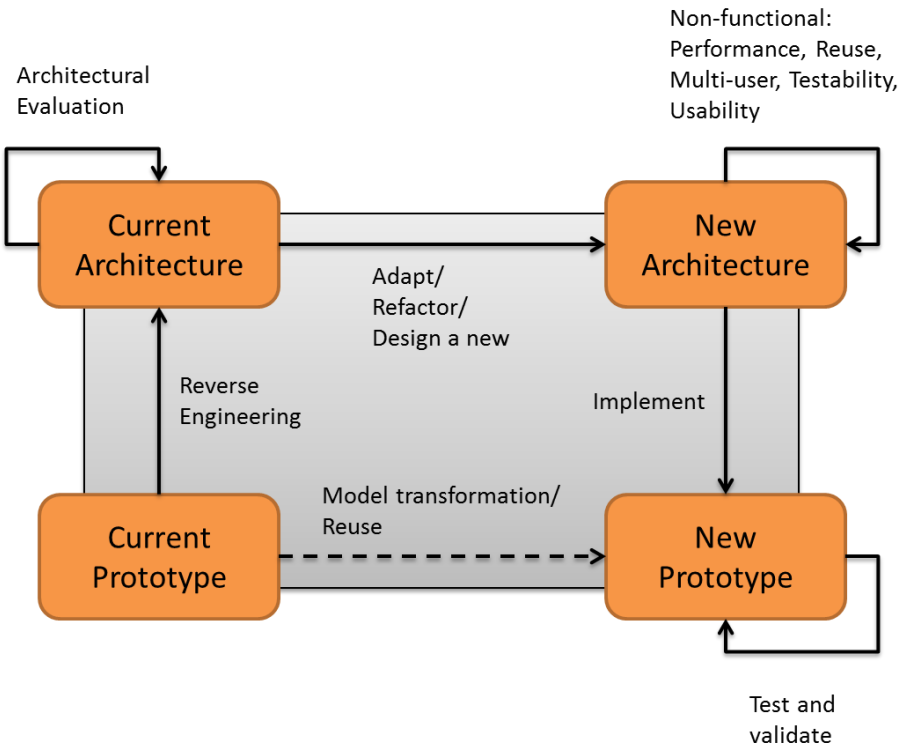


Figure 10 High Level Process for further development of the DF

The DF tool is at a good prototype level, but it is not ready for operational application by end-users in industry. In the meantime, extra functionality has been added along the lines of quite global and implicit architectural patterns. For that reason, an architectural evaluation is required including both the software design and the technology choices made. Based on the outcome of that evaluation, a decision is to be made on how to proceed with the tool development.

Two major prototypes have been made, each of which has added knowledge and experience on how to proceed with the method and the tool. The first DF tool prototype was developed using Eclipse technology. The second prototype involves a re-implementation using state-of-the-art web technology such as HTML5. The current DF tool is based on the second prototype.

Currently in Océ there are a few lead users, who are expected in the near future to grow in number. In addition, TNO-ESI expects to extend the user community by applying the DF to other projects in cooperation with various industrial users.

3.2 Project goals

The main goals in this project include the following:

1. Study and document the current functionality of the DF to obtain the current architecture from the current design. (Reverse engineering in Figure 10)
2. Document the current architecture and implementation of the system.
3. Redesign and propose a new architecture for parts of the DF tool.
4. Implement a new prototype from the new architecture.
5. Transform the old data model to the new data model. (Model transformation in Figure 10)
6. Consider the scalability of the system to be used by multiple users. (Non-functional requirement of the new architecture in Figure 10)
7. Implement a test framework for the system. (Validate new prototype in Figure 10)

The steps 1 to 5 are done in a group of two persons and the steps 6 and 7 are done as individual assignments. In this report, refactoring phase and the implementation of a multi-user support (step 6) and test support (step 7) is explained.

Since the concepts are already defined and the system is being used currently by customers, the system should be usable at every step of redesign. Therefore the graphical user interface remains the same while the core functionality and architecture continuously change.

The refactoring assignment is defined mainly because the maintainers of the DF tool have complained about the redundancies of code and data. Although not having a proper test framework makes the refactoring risky but the test framework can be developed later and deployed for future maintenance.

In the following section, requirement gathering activity of testability phase of the project is discussed.

3.2.1 Requirement Gathering

Design Framework was developed without following a specific development approach. For this reason, one of the non-functional requirements “testability” was not addressed in the current version. However, it was a task to determine how much testability adds value to the tool and what exactly testability means to DF.

To determine the value and meaning of testability, two FMEA (Failure Mode and Effect Analysis) sessions were conducted including two TNO supervisors and two engineers. First session focused on FMEA analysis from the Developer’s perspective. During this session, each member collected a set of scenarios which are important to the developer. Second session focused on FMEA analysis from End User’s perspective. Similar to the previous session, a set of scenarios which are important to end users were collected. Each scenario was then grouped to a concept and finally each concept was ranked based on the score obtained through the product of probability and impact factor.

3.2.1.1 FMEA Analysis - Developers Perspective

During the FMEA session, attendees were given time to gather a set of concerns assuming themselves as a developer. A set of points collected from attendees were grouped together into particular concepts, as shown in Table 2. Details of the result and arguments related to each concept are provided in the appendix (Table 12).

Each of the concepts was ranked based on the probability of occurrence and impact factor. Both probability and impact factor were weighted from 1 to 5. A lower probability value indicates lesser chances that a particular issue exists in the current project and vice versa. Similarly a lower impact factor indicates the lower severity if particular concept is missing during the development process and vice versa. The final scores were obtained multiplying probability and impact factor and were ranked based on the score.

The set of concepts that were obtained during the first FMEA session are given in Table 2.

Table 2 FMEA Result - Developers Perspective

Concept	Probability(P)	Impact factor(I)	P * I	Rank
Architecture and Design	5	1	5	4
Testability	3	4	12	1
Requirement	5	2	10	2
Usage	2	2	4	5
Tool	3	2	6	3
Deployment	2	5	10	2

Based on the above analysis, “Testability” proved to be the most important requirement for Design Framework and hence the decision was made to provide test support to DF.

3.2.1.2 FMEA Analysis– End User Perspective

A similar approach to the previous FMEA session was followed in this session. However, the concepts were collected from the end user perspective. Details of the result and arguments related to each concept are provided in appendix (Table 13).

Table 3 FMEA Result - End User Perspective

Concept	Probability(P)	Impact factor(I)	P * I	Rank
Documentation	4	2	8	6
Representation / Position Inconsistencies	4	3	12	3

Data Inconsistencies	4	5	20	1
Browser Support	3	3	9	5
Message Handling / Performance	5	3	15	2
UI	5	2	10	4
External Service/Server	2	4	8	6

Based on the end user perspective, “Data inconsistencies” is the most critical issue followed by performance issue and proper handling of messages. Data inconsistencies are mostly related to multi-user feature in Design Framework. Multi-user support was a pre-defined requirement and is discussed in section 7.

4 Reverse engineering

Abstract – In this chapter, we start to analyze functionality of the system, domain model, and the existing architecture and code-base. We end the chapter by addressing refactoring risks and strategies.

4.1 Design Framework tool's domain model

The result of the analysis of the DF functionalities and keywords is captured in the domain model (Figure 11).

The DF at the highest level contains users, units, representation types, and projects. A project has a root block which contains other blocks. A project also contains a set of mathematical models (constraints). A dependency uses these mathematical models.

A block can have a set of parameters, images, and texts. A block can also be connected to other blocks through a relation. Parameters on the other hand, are connected to each other through dependencies. As mentioned in Section 6.2, dependencies can be either validations or transformations. A model also can have a set of experiments.

The parameters can be measured by a value. This value has a unit of measurement. Furthermore, parameters and blocks can also be represented in different manners using representation types. The relations and dependencies (transformations and validations) can be connected to parameters and blocks using links.

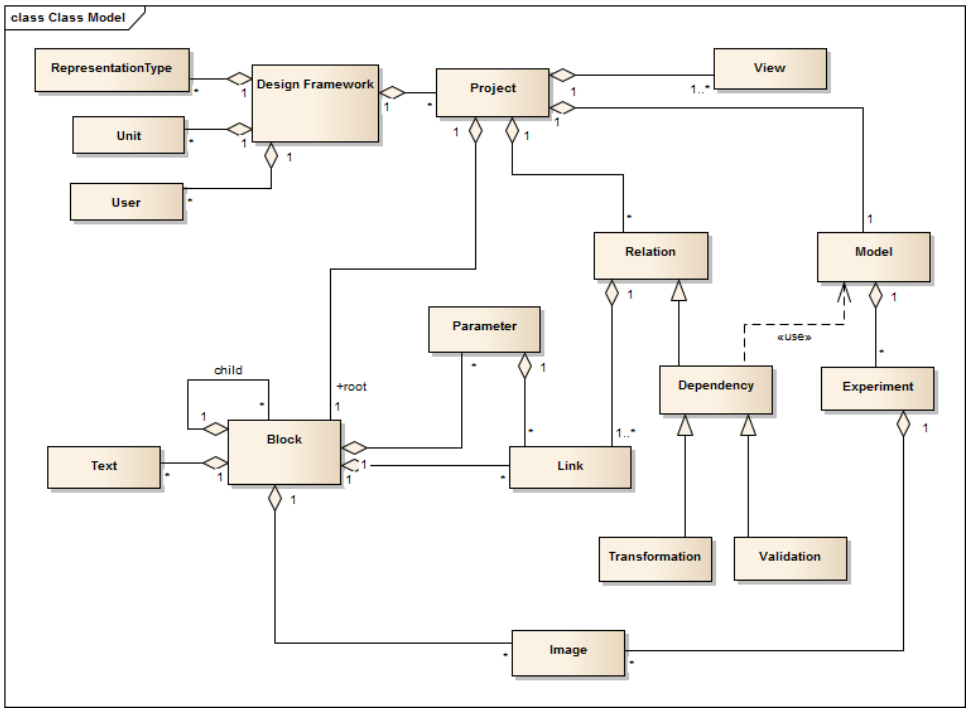


Figure 11 DF domain model

4.2 Current Architecture

The DF's latest prototype is implemented using LAMP architectural model. LAMP is an acronym for Linux, Apache Web Server, MySQL, and PHP, as this architectural model is deployed by these four technologies. Although LAMP was initially used in Linux, later it was adapted for Windows as well (XAMP) [6].

The GUI (Javascript and HTML) communicates through a set of commands to the backend (Apache and PHP). These commands are transferred to the server on HTTP requests. The responses are returned to the frontend on HTTP response encoded in JSON format.

There are four external services used in the current architecture of DF (Figure 12).

- Excel web server: DF needs to send data to this server in order to receive the diagrams and charts based on the data in the system.
- Frink Engine: DF uses the Frink engine in order to calculate the results of expressions.
- Experiment server: DF uses a server to deal with calculations which take long time.
- Matlab Engine: DF needs to send data to this server in order to receive graphs based on the data in the system.

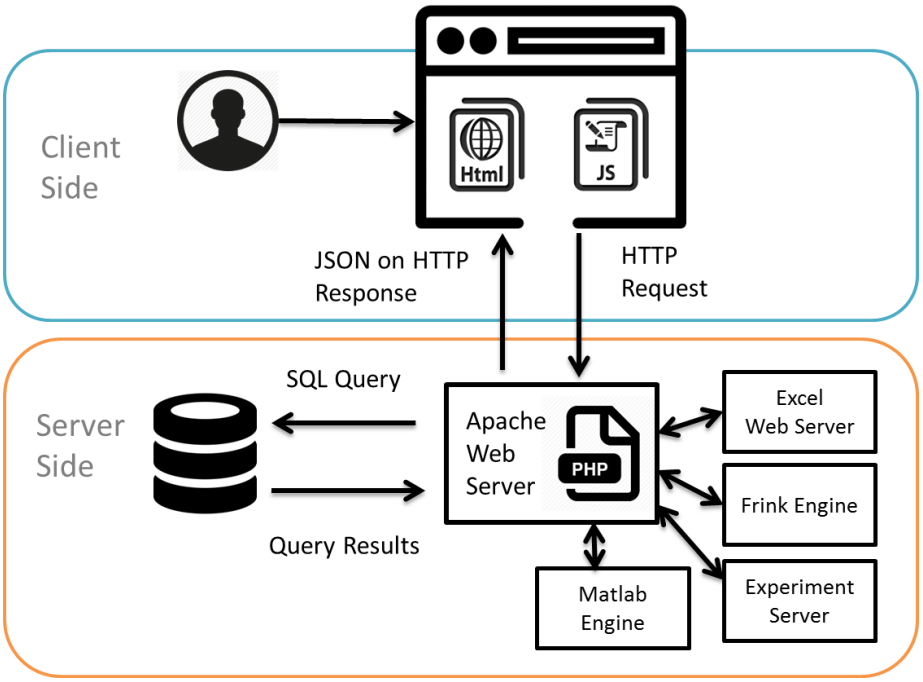


Figure 12 Design Framework's current architecture

4.3 Code Analysis

One of the goals of refactoring is to reduce the code complexity. The less code complexity the better maintenance. Some examples of a complex code are the presence of codes with long methods or codes with repeated code snippets.

After a manual inspection in the code, it is observed that code snippets are repeated all over the code base. This redundancy is especially due to the similarities between the functionality of a numbers of concepts in the DF. The concepts such as block, parameter, relation, transformation, validation, text, and image have a lot in common and adding a proper layer of abstraction would reduce this code repetition.

Two quality metrics have been considered in this section. The first one is the number of lines. In the next section we elaborate how the code complexity can be reduces. The method length and class length can be reduced as the consequence of reducing the number of lines.

Another metric is cyclomatic complexity. This metric also is used as an indication of software complexity. It is a quantitative measure of the number of linearly independent paths.

DF, as mentioned before, uses LAMP architecture. In the front-end, HTML and JavaScript are used. The Table 4 shows the result of the preliminary analysis of the number of lines of code. After refactoring we discuss the improvement in the code complexity.

Table 4 Number of line detailed analysis

Language	Number of files	Number of lines of code	Number of lines of comment
JavaScript	30	21761	3237
PHP	29	12086	5189
HTML	48	681	0

The cyclomatic complexity of the code is also computed using PHP Depend. The tool gave the number 709 as a measure for cyclomatic complexity and it will be compared to the new measurement after refactoring.

5 Refactoring

Abstract – In this chapter, the refactoring phases are elaborated.

5.1 Introduction

After determining a number of software quality metrics (number of lines of code and methods length) to improve, a refactoring process must be performed in order to increase the quality.

The refactoring was started from the backend to the frontend. The reason for this decision is that it is better to detect the inefficiencies in the backbone of the system as soon as possible before they infect the other parts.

The approach was to refactor layer by layer. The layers of the LAMP architecture are shown in Figure 14. Each time, the layer under refactoring must completely change while the upper layer is still capable to communicate with the lower layer. Therefore, some wrapper functions must be implemented in order to separate refactoring phases.

5.2 Process break down

In the process of refactoring of complex software with layered architecture with multiple functionalities, several techniques can be applied.

The first approach is to break down the software into functionalities. This means no matter how the software is built, one can do vertical refactoring where functionalities are separated and refactored individually. The prerequisite of this approach is to have a clear separation between functionalities.

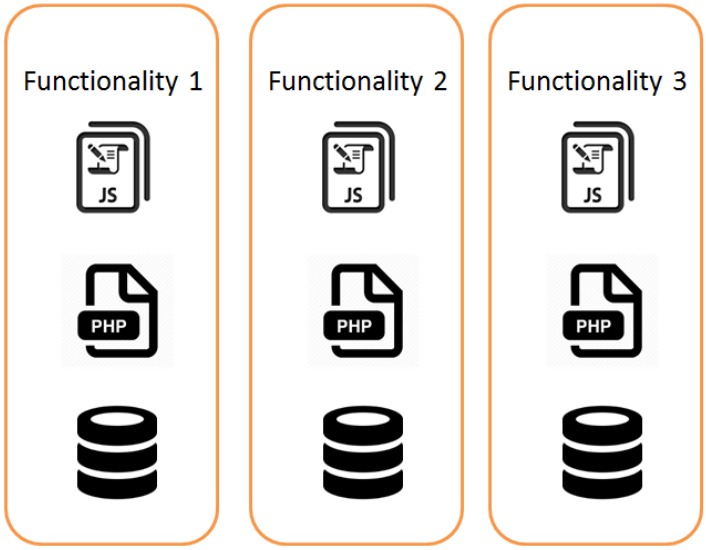


Figure 13 vertical refactoring

Another technique of refactoring is to break the software into meaningful horizontal layers. This makes sense especially when software is built with a layered architecture. The software in this case can be refactored layer by layer starting from the lower layer. Since the data is modeled in the lowest layer, and the business logic is built on

top of this model, any inconsistency in the lowest layer is appeared in the higher levels.

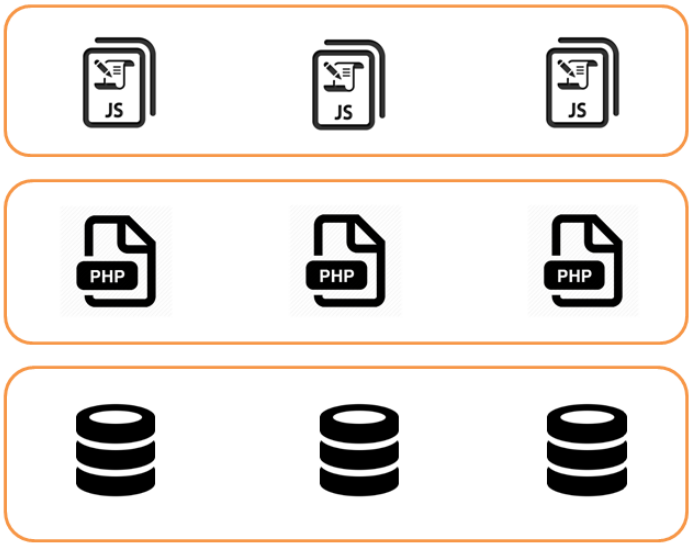


Figure 14 horizontal refactoring

5.3 Refactoring risks and strategies

As every process that changes a running (perhaps even working) system, refactoring is not immune to introducing errors. Although there are several techniques that should enable the programmer to avoid them or at least catch them early, introducing a failure with refactoring can have serious consequences. Therefore, refactoring should not be treated lightly, but instead be done with care and the possible problems in mind.

Current code base of the DF is highly inefficient and lacks a standard approach. The major findings of the code analysis were high redundancy, lack of documentation, and lack of architectural pattern. Therefore, the refactoring decision was made. Refactoring code comes with the overhead of a heavy testing load.

Martin Fowler [7] has described a number of refactoring techniques. After the code analysis, some techniques which are appropriate to apply and give the most benefits have been selected.

Here is the list of refactoring techniques that we consider:

- Techniques that allow for more abstraction
  - Generalize Type – create more general types to allow for more code sharing
  - Replace conditionals with polymorphism
- Techniques for breaking code apart into more logical pieces
  - Extract Method, to turn part of a larger method into a new method. By breaking down code into smaller pieces, it is more easily understandable.
- Techniques for improving names and location of code
  - Rename Method or Rename Field – changing the name into a new one that better reveals its purpose

- Move Method or Move Field – move to a more appropriate Class or source file

Lack of automatic testing makes refactoring more difficult. The DF tool was not developed using a Test Driven Development approach and neither were tests added in the later stage, which makes refactoring a high risk. Since, usability is one of the major requirements of the project, refactoring is already challenging.

Certain risk mitigation plans were made to minimize the refactoring risk:

- Schedule the change for a low development cycle or a low production cycle.
- Use proven design patterns where needed.
- Isolate the change from others to make it easier to find problems.
- Communicate regularly with other developers and stakeholders about the change.

5.4 Generalize type technique

The first observation while analyzing the data model was that it has data redundancies in the model. There are certain concepts inside the DF system which share functionality. These concepts are: Block, Relation, Parameter, Validation, Transformation, Experiment, Image, and Text.

These concepts could be generalized as a root object called Node. This interpretation has an impact on the data model as well. The question that was raised was: "How are the object-oriented models implemented in the database?" To answer this question table inheritance is explained in the next section.

5.4.1 Table inheritance

The result of the code analysis shows that the concepts block, parameter, relation, validation, transformation, experiment, image, and text support the same sets of operations and therefore they can be considered as a more abstract concept called Node. The concrete classes can inherit from the abstract class Node.

There are three solutions to store inherited classes in a database. The problem is formulated as below. The node concept is modeled as entity A. Other concepts such as blocks, parameters, relations, etc. are modeled as entities B, and C. The attributes "a, b, c, d" are those which are common in all concepts such as name, comment, etc. The attributes e and f are those which are specific to a concept. For example the attribute value belongs only to parameter concept.

Suppose we have three entities as below:

Entity (attribute1, attribute2 ...)

A (a, b, c, d)

B (a, b, c, d, e)

C (a, b, c, d, f)

There are three options to model these three concepts in the database [8] [9]:

5.4.1.1 Concrete Table Inheritance

Take each concept and map it to a single table. There is a separate table per concept.

Table {column1, column2 …}

T_A {id, a, b, c, d}

T_B {id, a, b, c, d, e}

T_C {id, a, b, c, d, f}

Pros: There is no query complexity. This method is simple to implement and it is easy to read and understand the database.

Cons: There is no common interface. There are common attributes and if one common attribute is to be modified or to be added then it has to be modified or added in all tables.

If a query which uses the common attributes is needed, then query has to be made on all tables.

According to some, it is recommended to use this method only if only leaf classes are concrete.

The schema of the table design using this method is presented in Figure 15.

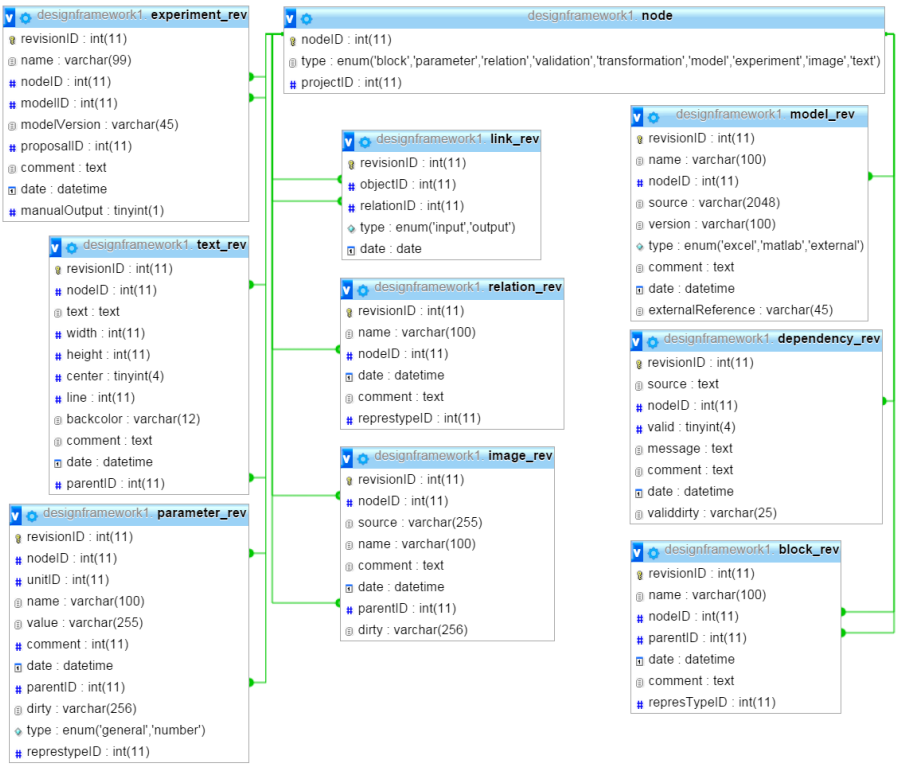


Figure 15 DF model using Concrete Table Inheritance

5.4.1.2 Single Table Inheritance

In this method all fields of all classes of an inheritance structure are mapped into a single table.

T_A {id, entity type, a, b, c, d, e, f}

The entity type can be B, or C.

Pros: There is no query complexity. It is simple to implement. By querying only one table, one can retrieve meaningful information about all entities. For example one can retrieve all entities with a certain value of the attribute "a" only by one query.

Cons: It is more error prone because some fields remain null for some concepts.

It is recommended where sub-tables do not have many additional columns. The schema of the table design using this method is presented in Figure 16.

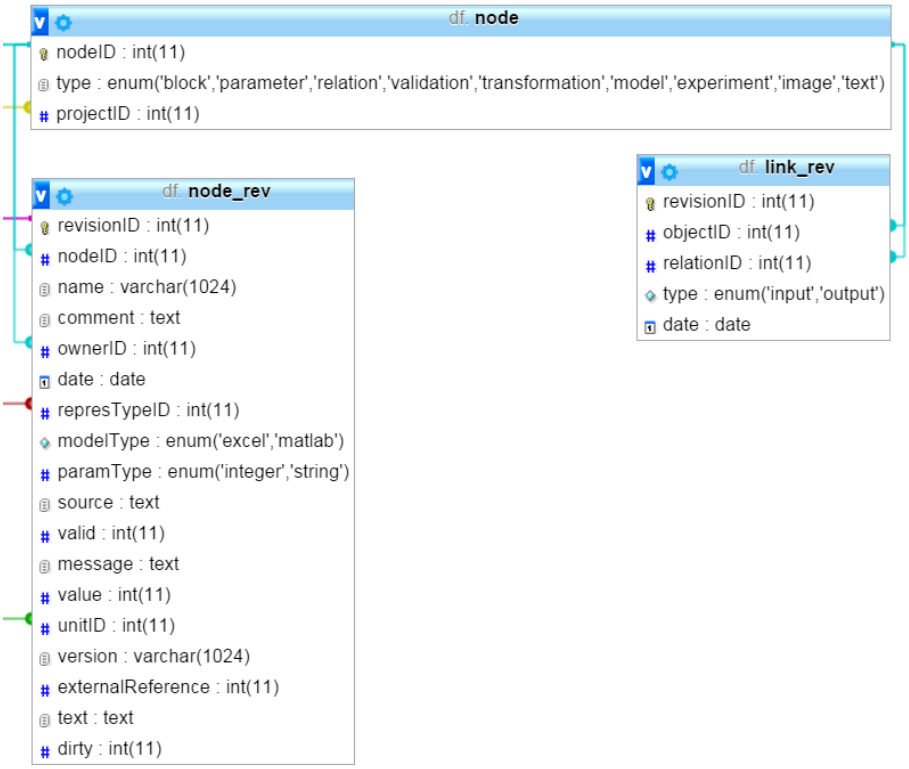


Figure 16 DF model using Single Table Inheritance

5.4.1.3 Joined Tables Inheritance

Each class is represented in one table and the attributes of a child class are retrieved using a join between the class and its parents.

A {parent_id, a, b, c, d}

B {id, parent_id, e}

C {id, parent_id, f}

Pros: It has low storage footprint.

Cons: Increased query complexity to join tables and performance loss.

The schema of the table design using this method is presented in Figure 17.

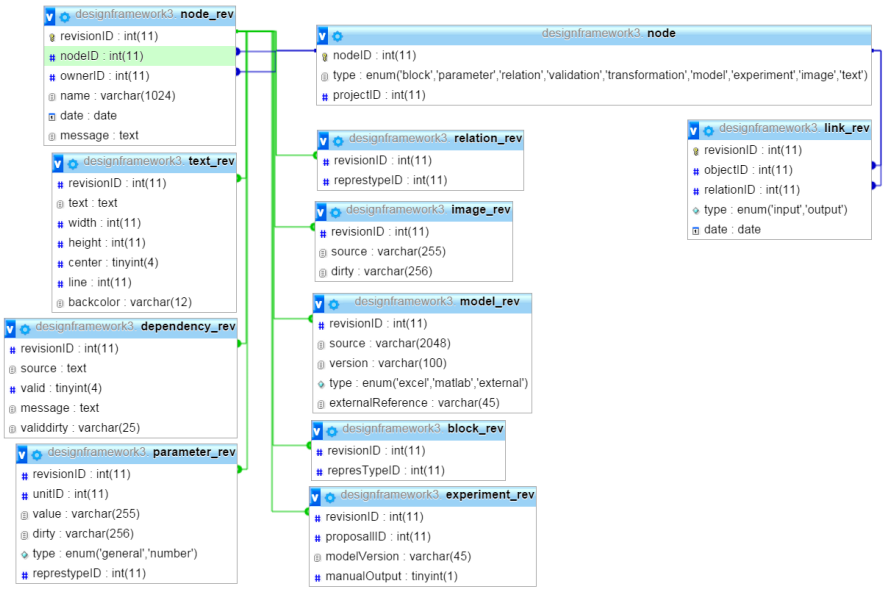


Figure 17 DF model using Joined Table Inheritance

After analyzing the above solutions, the Single Table Inheritance was selected. The reason is that in Single Table Inheritance, the number of joins is reduced compared to Joined Table Inheritance and therefore it is preferable.

It is also preferable to simplify the backend code; therefore by choosing the Single Table Inheritance over Concrete Table Inheritance, the business logic is removed from the backend. This means that the backend is aware of only the node and not the concrete objects.

5.5 Renaming technique

Two issues are addressed by renaming technique in the refactoring. First, the naming conventions are not always respected. Second, the method and variable names are not clear and understandable.

In order to deal with these issues, new names should be used which reveal the purpose of the method or variable better and follow standard naming convention. This improves the readability of the code. The maintainability of the code is improved as well in the sense that developers are able to understand the code much faster which leads to a better maintenance.

5.6 Extract method technique

5.6.1 Front-end and back-end communication

The communication between the frontend and the backend consists of a set of commands on HTTP. Since the business logic is removed from the backend, the communication is also simplified. The block, parameter, relation, transformation, validation, image, text, and experiment concepts are not in the communication anymore and we have only nodes and links. Table 5 shows the number of use-cases after refactoring.

Table 5 Frontend-Backend communication after refactoring backend

Concept	Functionality
Node	Create, Update, Delete, Change Parent, Get History
Link	Create, Delete
Project	Create, Delete, Open, Update, Export, Import
View	Create, Update, Open, Get List
Unit	Create, Update, Delete, Get List
RepresentationType	Create, Update, Delete, Get List

5.7 Refactoring results

After the refactoring, the code is again analyzed to obtain the performance of the refactoring process.

Since only the PHP is refactored the number of lines after the refactoring process is recalculated. This number is decreased from 12087 to 8980. This means a reduction of 25% in lines of code. This makes the code simpler and easier to maintain.

The number of tables is reduced from 30 to 10. After refactoring tables contain only necessary information redundant data model is fixed.

The cyclomatic complexity of the code is also reduces from 709 to 611. These numbers are obtained from the tool called PHP Depend.

6 Database migration

Abstract – Users have been working with an old version of a database and they have data that has to be preserved. After refactoring the database, a new requirement is defined, which is database migration. Database migration also helps to validate the refactoring result.

6.1 Introduction

Database migration is a one-time task. Once all old databases have been migrated to the new ones, the migration project is not useful anymore, although it can always be used to specify the semantics of the database.

6.2 Use case

Each user uses the database migration only once. The user inserts the old data base configuration and the new database configuration and the system creates a new identical database as the old version in new format. The schema changes but the content of the data remains the same.

6.3 Migration process

The migration process includes reading data from the old database, performing the changes, and inserting them into the new database.

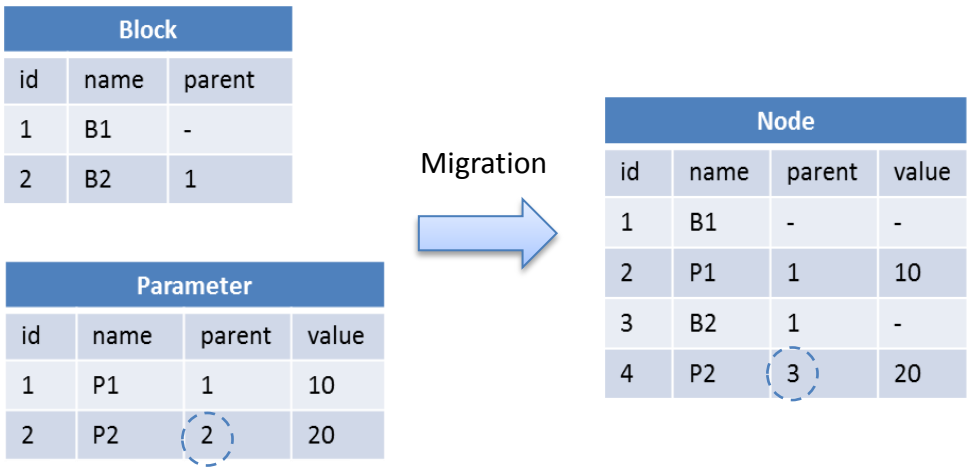
There are three types of main changes in the database which should be considered:

Generalize type changes: As explained in the refactoring section, the generalize type technique is used in order to reduce the number of redundant data types. In the previous design, for example, the concepts of block, parameter, image, text, etc. were considered as different data types and therefore tables were designed for each of the concepts. The result of studies showed that it was not necessary to keep a table per concepts. One table called "node" therefore was allocated for all concepts and the migration projects is responsible for reading those concepts from the database and inserting them with new identifiers in the new database.

6.4 Migration results

The migration process is done for all concepts in the database of the design framework. Although it is tested manually for a number of test database but there is no proper validation.

An approach to validate the result would be to reverse migrate the migrated database and validate if the result is identical to the original database.



Migration takes care of the changes of IDs.

Data distribution: During the refactoring phase certain tables have been removed and data has been distributed among the other tables. Some data was redundant in the previous design and the redundant data is removed.

Name changes: During the refactoring phase, certain names have been changed in order to create more meaningful names. The migration project must take care of the name changes and perform the migration accordingly.

7 Multi-user support

7.1 Introduction

The DF is a tool that allows access to multiple users at the same time on the same project. This can create synchronization problems. For example, if a user is modifying an object, the tool should be able to notify other users which are modifying the same object in such a manner that the data remains consistent. Currently the DF has multi-user support but it is not at a mature level. The assignment is to design and implement a more robust multi-user support.

The DF is used as a visual modeling tool besides capturing the rationales during the design process. Designers use certain concepts to model their systems. An example of a concept is a *block*. A block can represent architectural elements such as components, requirements, and structures. An instantiation of the block concept is a *block node* which has data describing its attributes such as name, parent, and comment. Since DF is used in a multi-user environment, there is a need to synchronize this data.

There is yet another type of synchronization besides data synchronization, which is synchronization of the representation. The objects have other data describing their representation, for example, their position. The examples of these two types of synchronization are described in sections 7.3.7 and 7.3.8.

In this chapter, first we study the git version control system because of its similarity to our version control system. Then we explore the concepts in DF to support multi-users, study some scenarios, and make a list of requirements of both types of synchronization and their related requirements and scenarios. Finally we propose solutions to address requirements.

7.2 Version control system

A version control system is used to manage the changes of documents. In this chapter we study the DF as a version control system. In this section we study the object model and merge mechanism in git, one of the most popular version control systems. For more (precise) information on git please refer to git documentations.

7.2.1 Git object model

In this section we describe how the git works internally. There are four types of objects defined in git: blob, tree, commit and tag. Every object consists of three parts: type, size and content.

Blob: It is used to store file data. The Figure 18 illustrates a blob. If a file does not change from one revision to another, git uses the same blob to represent the file data.

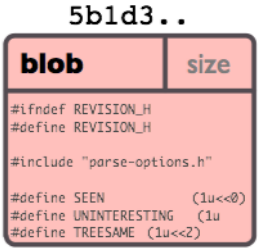
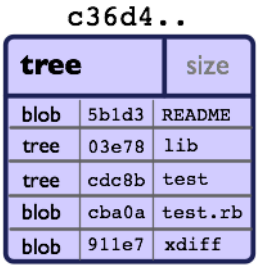


Figure 18 Blob structure in git

Tree: It is used like a directory to contain a number of files or trees.



Commit: It points to a single tree to represent what a project looked like at a certain point in time. It also contains meta-information such as a timestamp, the author of the changes, a pointer to the previous commit(s), etc.

Tag: To indicate a specific commit, tags are used.

Git builds a structure using these four types of objects and it manipulates them once something changes.

For each commit, it creates a tree referencing to all folders and subfolders (trees) and files (blobs). Once a change has been made, a new commit object is created and a new tree structure is created for the new commit. For the files which are not changed the previous blobs are used. For the files which are changes a new blob is created.

The Figure 19 shows the tree structure in git.

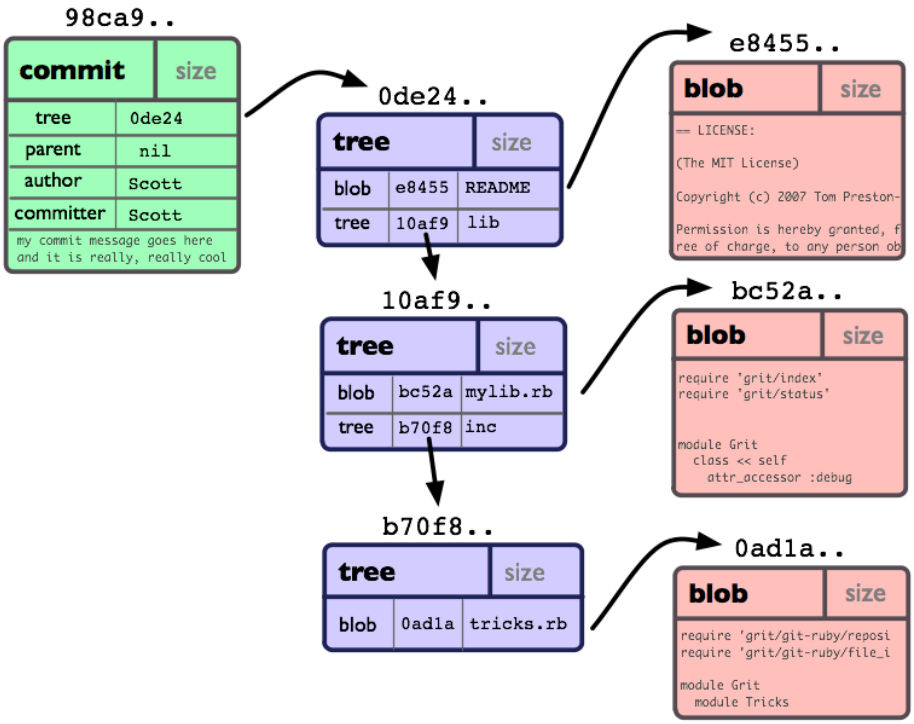


Figure 19 Object structures in git

7.2.2 Git merge

In git, documents are located on a branch called master. When a user wants to work, he/she creates a branch and makes the changes on his/her own branch. When the changes are completed, he/she merges the changes back to the master branch. The Figure 20 shows a simple scenario.

A revision commit is when a user makes changes and submits his/her changes. Commits are represented by circle in the Figure 20.

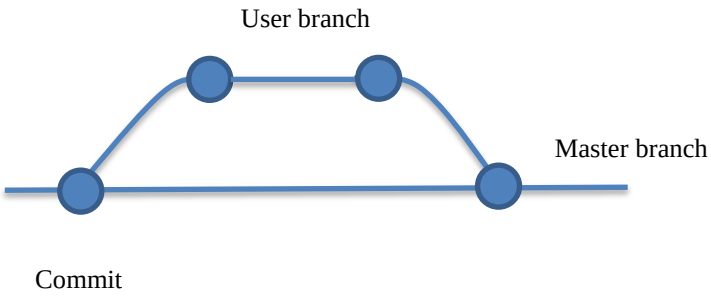
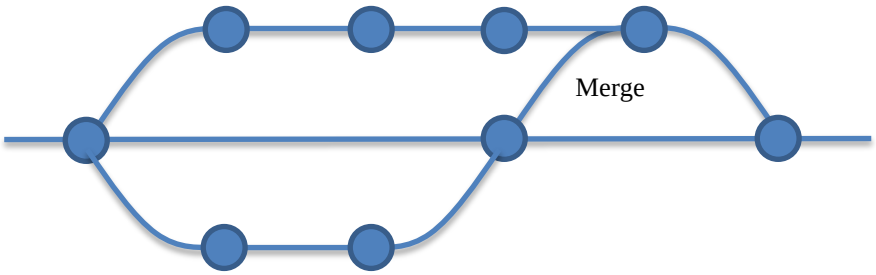


Figure 20 Merge in git version control system

Now suppose while the user was modifying his/her branch, another user makes changes on another branch. The user has to merge the changes on his branch before committing to the master.



7.3 Concepts of DF versioning

As described in Figure 8, the design flow layer of the DF model contains a set of system descriptions. The set of system descriptions, maintained in a DF store, is organized as a tree. Each system description can have four states namely accepted, declined, submitted, and pending.

The state is initially pending. It can be changed to submitted state. From submitted state, it can be accepted or declined.

Each system description is derived from its predecessor by a design decision. This design decision which addresses a question leads to a transformation of the system description which is called proposal. The result of this proposal is a new system description.

Each system description can have a set of successors. One and only one successor can be in the accepted state. If one of the successors is in the accepted state all other successors are declined. Only the latest system description can have pending and submitted system descriptions which are called alternatively proposals. Each set of proposals address a question.

Each system description has one/multiple owner(s). If a system description is pending, there is only one owner and only the owner can see the pending system description. If a system description is submitted the ownership is shared and all users are owners and they can access the design description. The ownership of an accepted or declined design description is the one who accepted or declined. These accepted or declined design descriptions are accessible to everyone.

The Figure 21 shows all the described concepts.

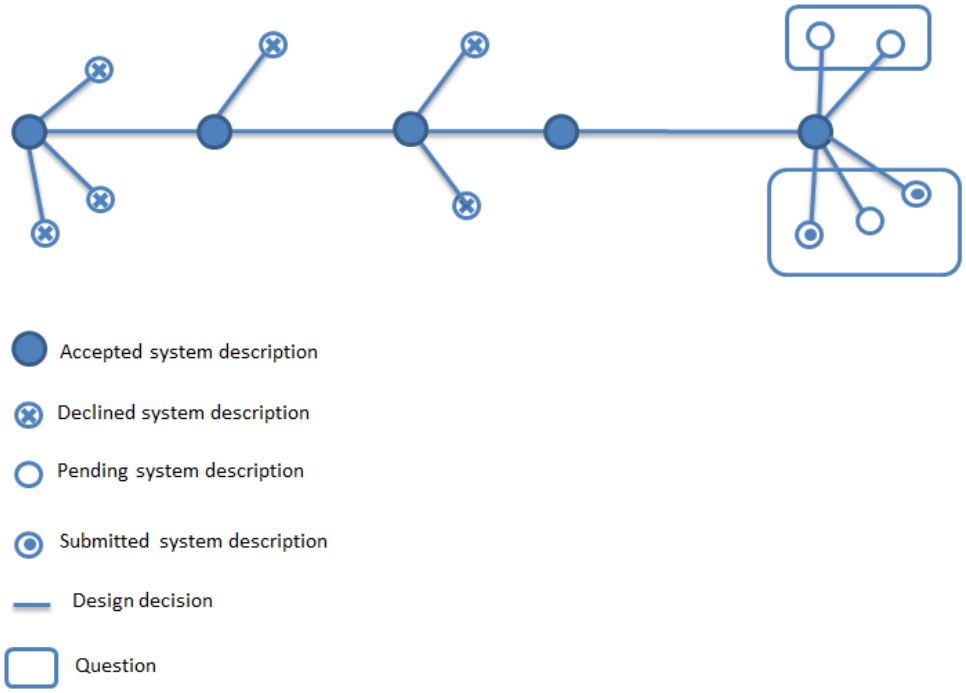


Figure 21 Tree-like system descriptions' store

7.3.1 System description

The DF is a tool that can be used to model a system. It is composed of a number of projects. A project contains a number of views. Each view is supposed to address a number of design concerns in the perspective of a designer. Each view contains a diagram associated with it. This diagram describes the design of a real system from the perspective of a designer.

Each diagram contains a number of *nodes*. All views share the same nodes within a project. Some views hide a number of nodes in order to address different concerns. Views also keep the information related to the representation of the nodes. The data is consistent is all views.

The set of nodes with representation data for all views is called system description. The system description is versioned. Any modification of an attribute of an object results a new revision of the system description. There is always the latest revision of the system description available on the server.

A system description can have four states: accepted, submitted, pending, and declined. In the next sections we describe these states. In Figure 22 the blue dots represent the accepted system description. (See Figure 26)

All descriptions are stored in the repository and the user only looks at the latest accepted system description and the pending description which are created by the user.

7.3.2 Design question

At any time during the design phase, a user can make questions addressing requirements to be analyzed for future solutions (new pending system descriptions). A question that is usually imposed on a system can ask for adding a functional or non-functional requirement, fixing an issue in the current design, or refactoring/redesigning parts of the system. (See requirement 2)

	Requirement	Current status
1	The DF must support multi-user mode in which the number of users can be up to 200.	-
2	Every user must be able to create questions.	Ok

The questions are not versioned. The project holds the questions and their description. Each question is answered with several proposals. Proposals are described in the next section. (See Figure 26)

Users create questions to address specific problems. However, there is the possibility to modify a system description without creating questions. In this case a default question is created.

In Figure 22 the blue dots represent the accepted revisions of the system description. Each revision of the system description contains nodes. Nodes can be added, modified, or removed from one revision to another. There is always only one latest revision of the system description. At any time during the design process, a question can be created.

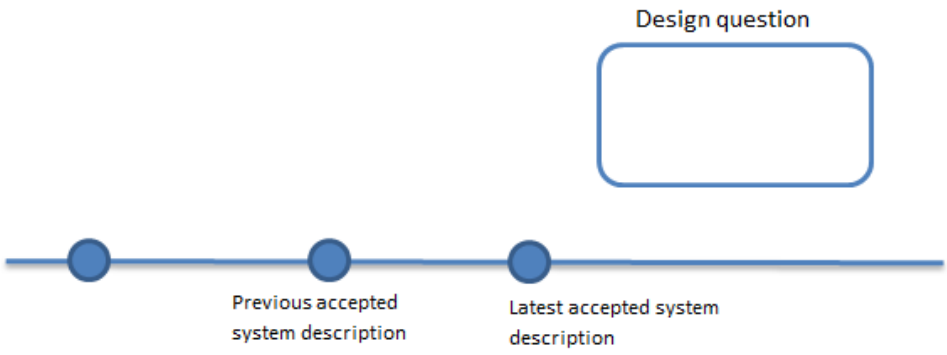


Figure 22 Design questions

In Figure 23, we demonstrate a question posed on a system. On the left side, the latest revision of the system description is demonstrated. On the right side, a question is posed.

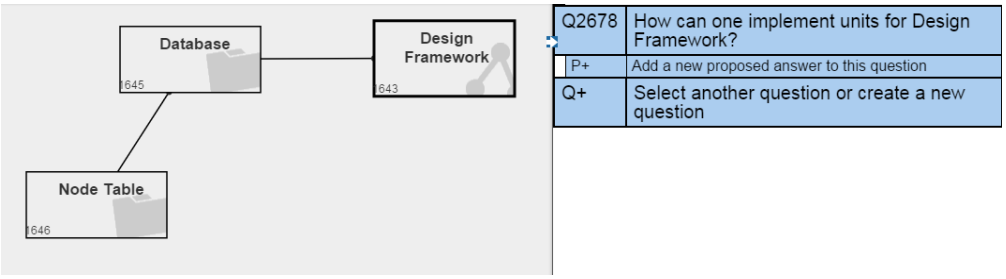


Figure 23 Example of design questions

7.3.3 Proposal

A proposal is a transformation on a system description. Proposals are made to address a design question. A number of modifications are made by a user to achieve a system description proposed in the proposal. A modification is made each time to modify attribute "X" of an object "Y". Also, an attribute can be added or removed.

In Figure 24, each dot represents a proposal of the system description. Blue dots are the accepted system descriptions. The accepted system descriptions are available on the system and accessible by all users. Pending system descriptions, on the other hand, are only available to the users working on the proposal associated to that system description.

In Figure 24, in order to address a question, three proposals are created, each having a number of modifications to achieve a system description. White dots represent pending revisions of the system description. (See requirements 3, 4, and 5)

	Requirement	Current status
3	User must be able to create proposals.	Ok
4	When a user selects a proposal, only the pending system description of the selected proposal must be displayed.	Pending system descriptions are displayed on all proposals of the user simultaneously.
5	When a user selects a proposal and makes a modification, a new pending system description must be added to the selected proposal.	Ok

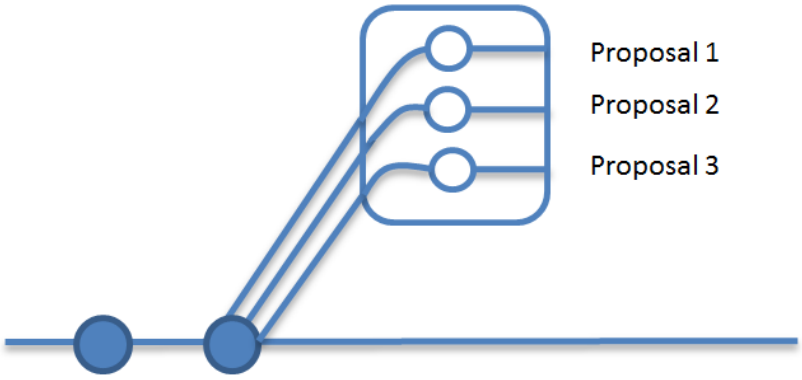


Figure 24 Proposals

In Figure 25, a proposal is created and a modification is made. Creation of the block called “Unit Tables” is a modification which results in the system description on the left side. This revision of the system description is still pending.

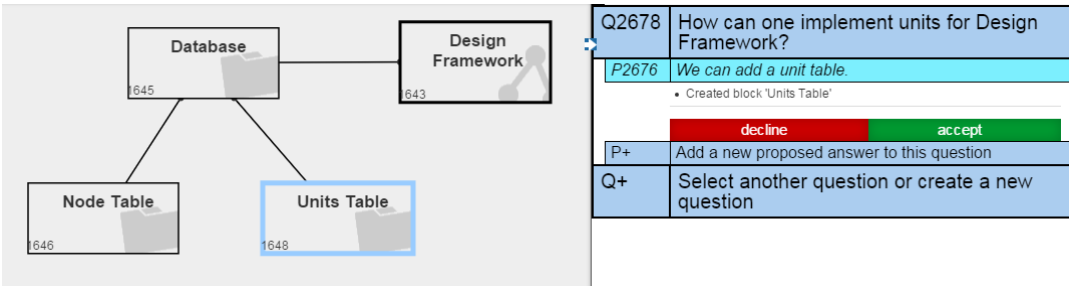


Figure 25 Example of proposals

Figure 26 illustrates the relations between system descriptions, questions, proposals, and node.

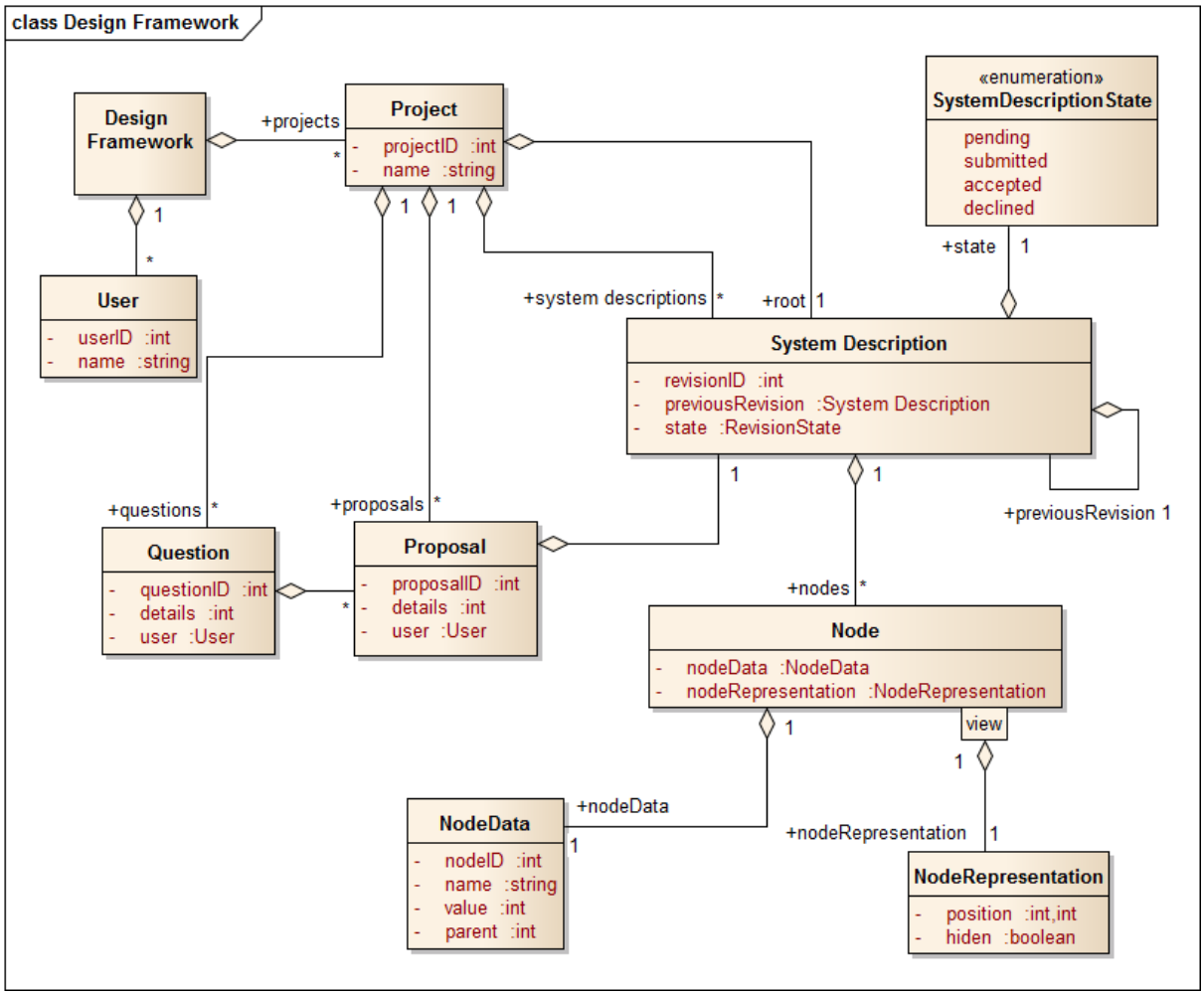


Figure 26 Class diagram of the revision control in DF

7.3.4 Submit proposal

When a proposal is ready, the user may want other users to see his/her proposal. The reason to submit a proposal is to allow other people to decide if the proposal should be accepted. The DF does not support different user types. All users are of the same type. The proposal can be accepted by the creator or another. If a proposal is not submitted, the decision will be made only between the available submitted proposals. (See requirement 6)

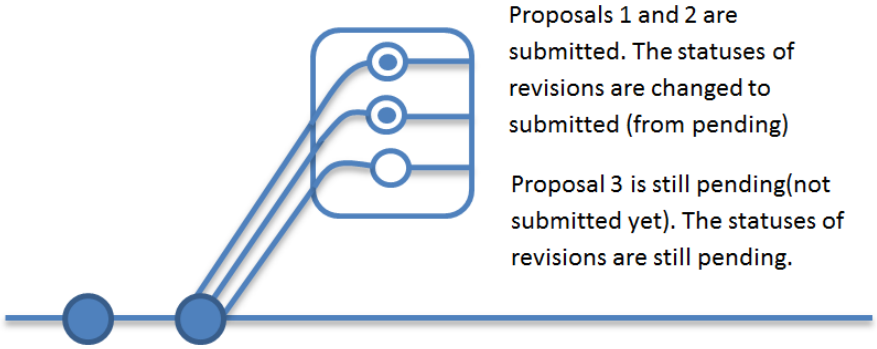


Figure 27 Submit proposal

A submit button must be designed, which is different from accept button explained in the next section. (See requirement 7)

	Requirement	Current status
6	If a user submits a proposal, other users must be able to see the system description related to the proposal.	The concept of submit currently does not exist.
7	There must be a submit button.	The concept of submit currently does not exist.

7.3.5 Accept proposal

When there is a set of proposals to a question, a user sends an accept request for one of the proposals. If there is no new revision on the server the request gets granted. As a result the state of the design description associated to the proposal is changed to accepted (from submitted). The latest accepted system description now belongs to the accepted proposal. In this case the question is closed and other proposals are declined. (See requirements 8, 9)

	Requirement	Current status
8	When a user accepts a proposal, the latest accepted system description is replaced by the proposal and the status of the proposal changes from “pending” / “submitted” to “accepted” state.	Ok
9	If a proposal is accepted, other proposals related to the same question are automatically declined but maintained in the database.	Other proposals can still be accepted.

If the accept request is not granted, it means that another user has accepted a proposal and the latest revision is not the same as the precedent of the proposal anymore. In this case, a synchronization mechanism is needed. The synchronization is explained in the next section.

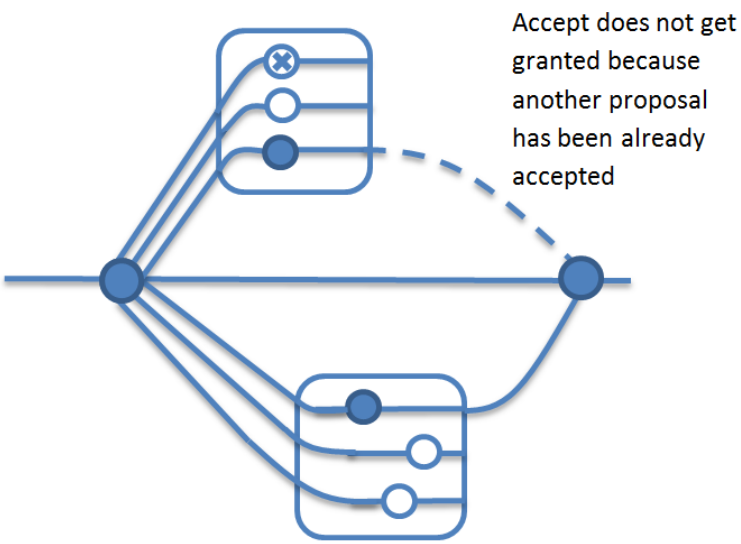


Figure 28 Request cannot get granted before the latest system description has been updated

In the Figure 29, proposal 3 is not submitted. This could be a user experimenting without having the intention of submitting anything. This proposal must be removed from the database. (See requirement 10)

	Requirement	Current status
10	If a proposal is not submitted, and another proposal is accepted, the pending proposal and related system description must be removed from the database.	Pending proposals are not removed.

The accepted proposals define a rooted path in the system description tree.

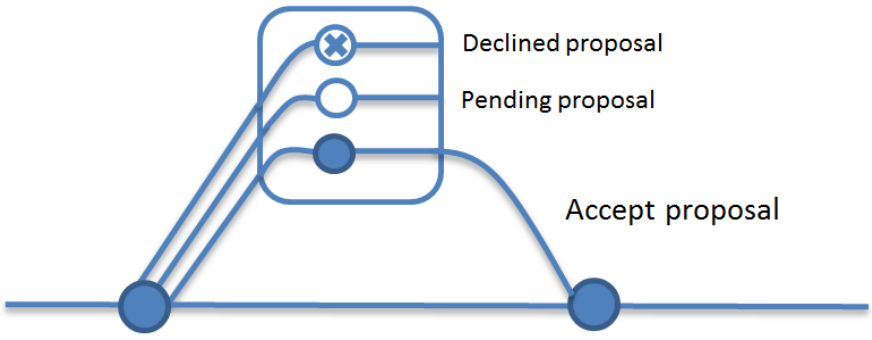


Figure 29 Accept proposal

7.3.6 Multiple users

A user can have multiple proposals for different questions. Multiple users can have proposals to the same question (See requirement 11)

	Requirement	Current status
11	Each project can have multiple questions. Each question can have multiple proposals proposed by multiple users.	Ok

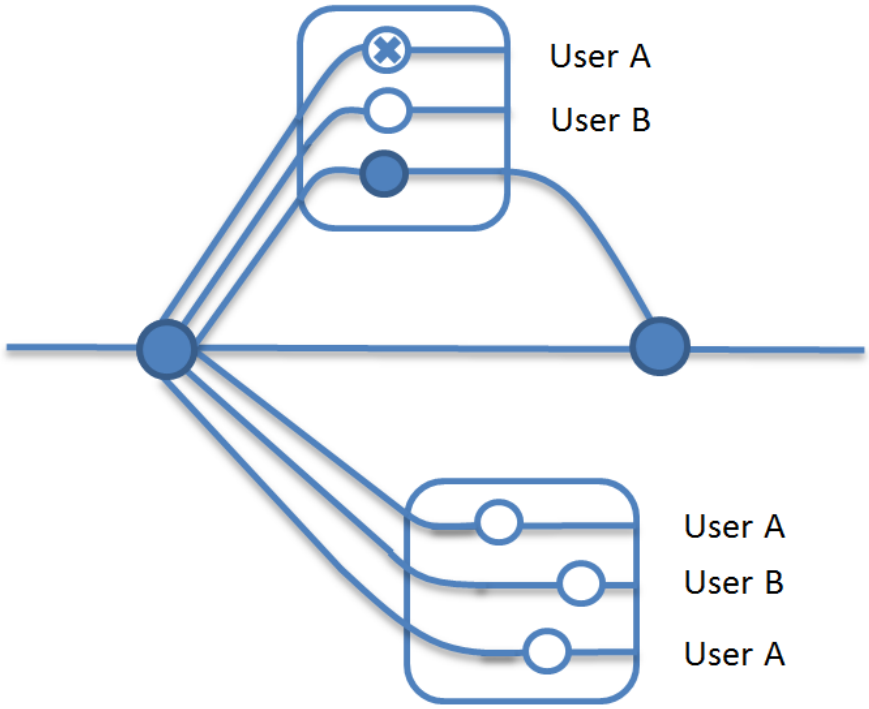


Figure 30 Multiple users

7.3.7 Data Synchronization

When user “A” accepts a proposal, the modifications are transferred to the server and the latest revision of the system description is changed. Other users (e.g. “B”) working on their proposals for other questions must be notified using a mechanism that the latest revision has changed and they are no longer the successor of the previous revision. The modifications of the users must be revised in order to resolve the conflicts if any. There are three scenarios which can happen:

1- If there is no overlap between what has been modified and accepted by user “A” and what is being modified by the user “B”, there will be no merge. For example a new node is added or a node is modified which is not being modified by the user “B”.

2- If the user “A” has added/removed information, that information is added/removed to/from the proposal of the user “B” only if user “B” has not modified that information.

3- If the user “A” has removed a node, and the user “B” is not able to modify the node anymore and it is removed from the view of the user “B”, too.

The scenarios 2 and 3 are illustrated in the Figure 31.

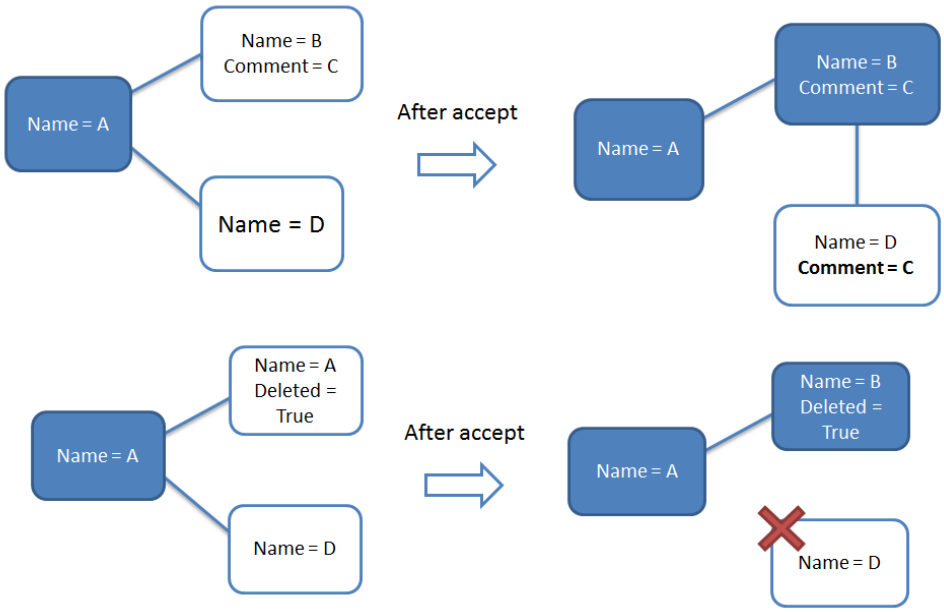


Figure 31 Merge scenarios

If a proposal is not synchronized with the latest revision, the proposal is not allowed to be accepted or submitted. (See Requirement 12, 13)

Requirement	Current status

12	Users must send a request of acceptance. If there is a newer revision of the system description on the server, the request does not get granted and the user must get the updates and resolve conflicts if any.	Not implemented.
13	If the latest revision of the system description changes, all users must be notified and their predecessor revision must be updated to the new revision of the system description. Users must get a warning within less than 5 seconds.	A warning is sent after 5 seconds. The predecessor revision of the pending revisions is not changed.

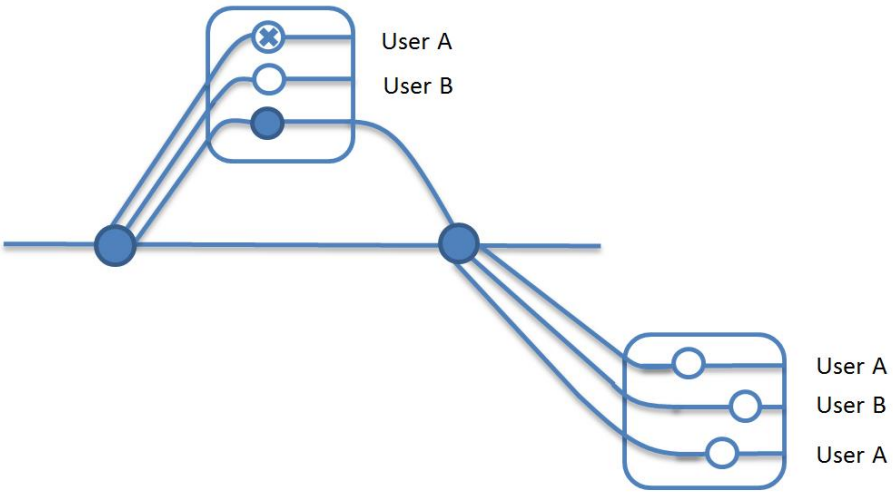


Figure 32 Data synchronization

The following scenario explains the expected behavior of the system.

- User "A" modifies the block "T."
- User "B" modifies the block "T."
- User "A" submits and accepts his modification of the block "T."
- The system notifies user "B" that there are modifications from other users and he must get the latest revision. Note that this notification is sent immediately and it might take few seconds. If user "B" sends an accept request before the notification has arrived, his request is not granted and he is asked to update before accepting. He also gets a list of conflicts to resolve.

In case user "A" and "B" modify different blocks, the same procedure happens except that there will be no conflict to resolve.

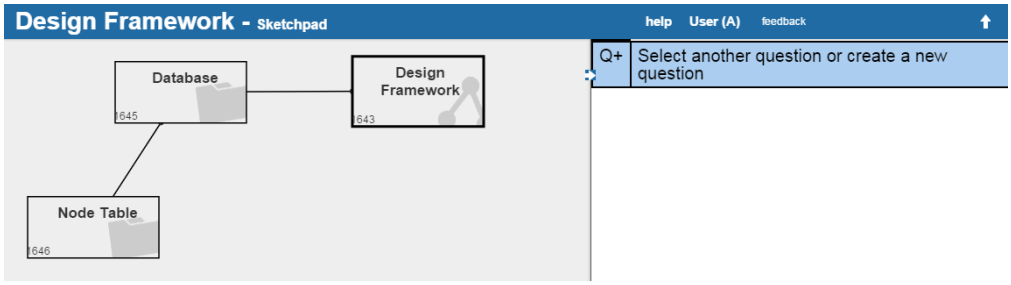
Data synchronization in different views:

A proposal can affect elements in all views. If the proposal has not been submitted yet, the owner of the proposal is able to navigate between different views and see the changes there. (See requirement 14)

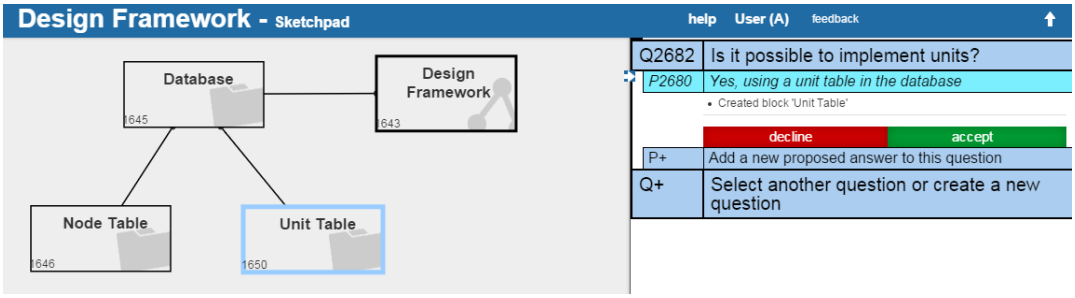
	Requirement	Current status
14	A selected proposal can be viewed in all views of the user working on the proposal.	Ok

7.3.7.1 Scenarios

- The latest revision contains three concepts (DF, Database, Nodes Table)

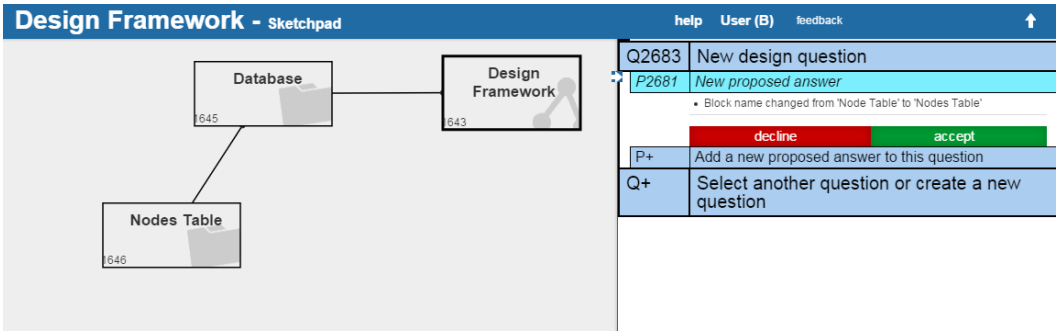


- A user asks a question “Is it possible to implement units?”
- User “A” adds a proposal “Yes, using a unit table in the database”, and he adds a Unit Table to the diagram.

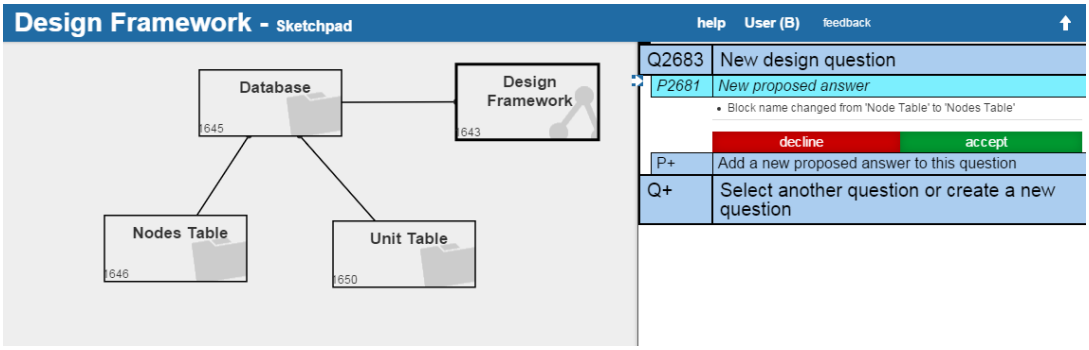


Scenario 1 - Users modifying different objects

- In parallel, user “B” is working on the same diagram. He finds a typo. The block “Node Table” must be called “Nodes Table” instead. He decides to make a modification. He modifies the property name of the block object 1646. While he is making the modification, he is unaware of the changes of user “A.”



- When user “A” submits and accepts his modification, the new modification must not override the changes of user “B,” but user “B” must receive the latest updates from user “A.”



In the Figure 33, the scenario 1 is demonstrated.

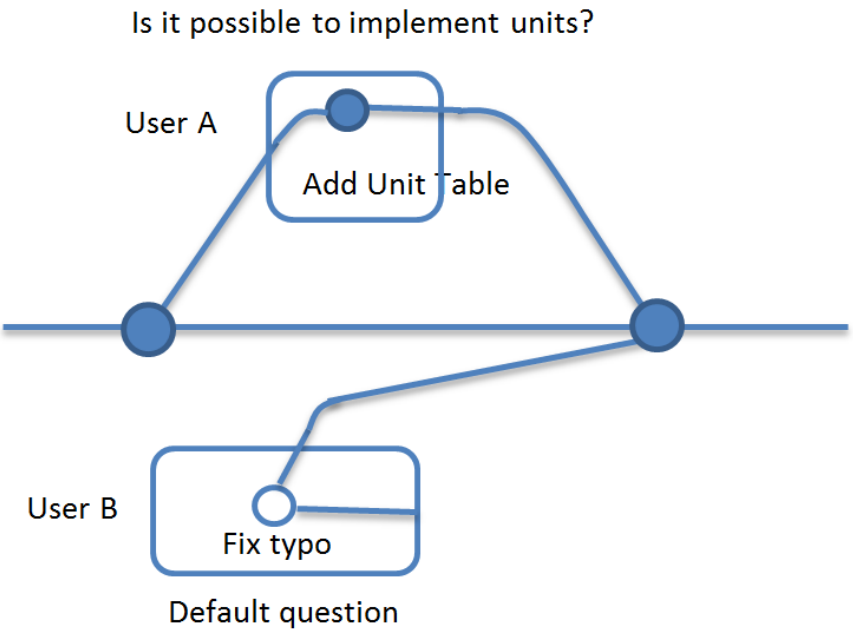
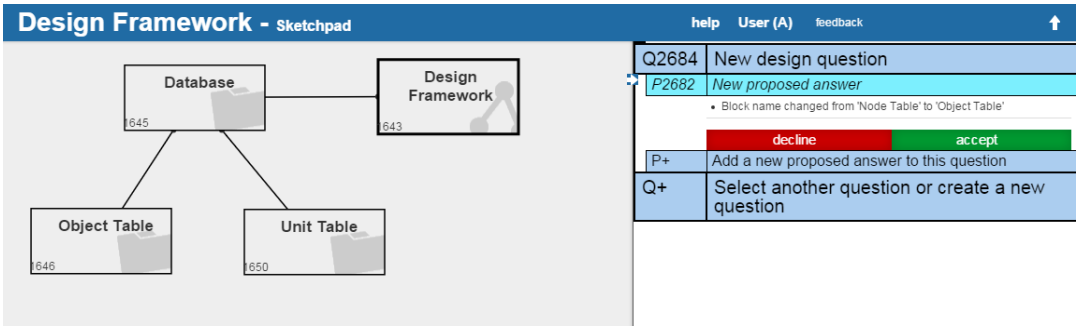


Figure 33 Scenario 1 - User modifying different objects

Scenario 2 - Users modifying the same object

- User "A" decides that the "Nodes" must be called "Objects." As a consequence the "Nodes" table is renamed to "Objects". User makes the modifications.



- Before user "B" accepts his changes, user "A" accepts.
- User "B" using an update mechanism receives the changes of user "A." He/she cannot submit or accept any proposal before getting the updates. When he receives the updates he knows that now the "Node Table" is called "Object Table". He/she decides if he wants to continue his changes using a new name.

In the Figure 34, the scenario is demonstrated.

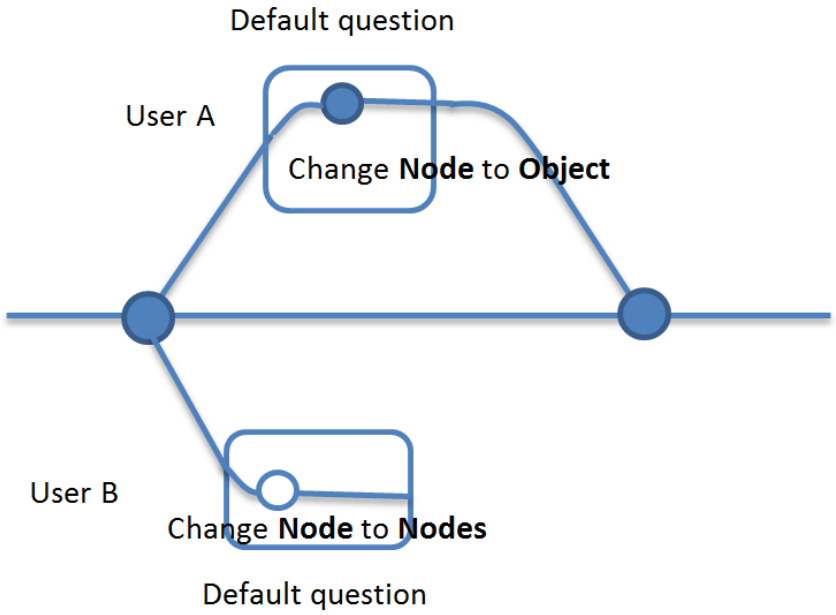


Figure 34 Scenario 2 - Users modifying the same object (before merge)

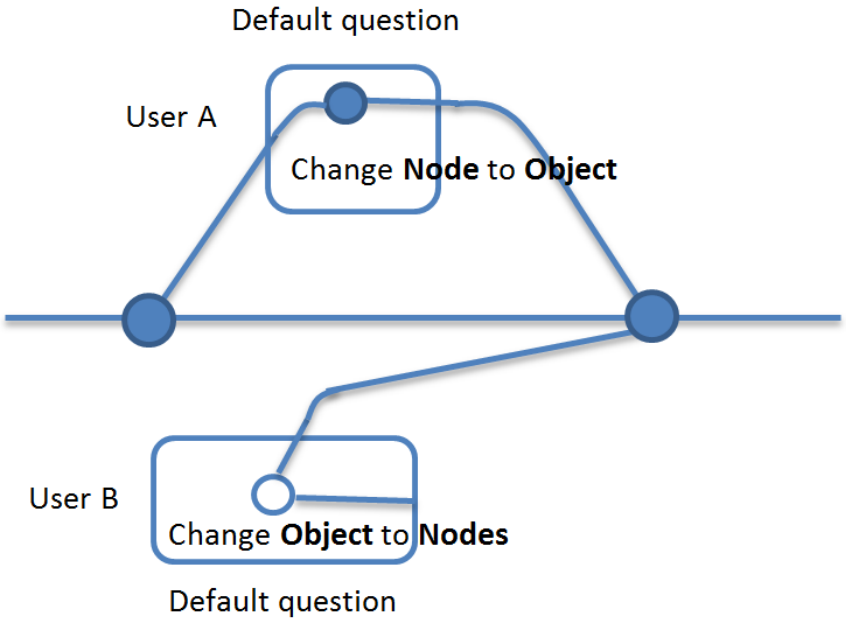


Figure 35 Scenario 2 - Users modifying the same object (after merge)

7.3.8 Representation Synchronization

The DF uses views to frame specific system concerns for multidisciplinary architects. It is the equivalent of viewpoint in ISO 42010. Each view owns a number objects. Some objects can be viewed in more than one view. For example a block “A” can be present in only two views and not present in the third view. The block in each view is placed in different places.

In this context, the representation means whatever attributes which are related to the representation of data such as position and visibility. The representation synchronization aims at keeping representation data of multiple users coherent.

7.3.8.1 Current issues

There are two major problems in the current implementation, which leads to two requirements.

First, the history of the representation data is currently not logged. It means that by reverting to the system description in the past, one cannot find the representation data available.

Second, when a user modifies the representations, other users do not get any notification unless they update their projects. If other users had also changed the same object, their changes are immediately overridden by the update. (See requirements 16 and 17)

	Requirement	Current status
15	When something has been modified in a proposal, the system must verify whether this modification has been previously modified in the same proposal. If yes, a new revision must not be created. Only the existing one must be updated. This holds true for the representation data as well.	Not implemented.
16	Log the history of the representation data	Not implemented.
17	Representation data must be part of the revision of the system description. When accepting a proposal, the data must be sent.	Not implemented.

7.3.9 Solutions

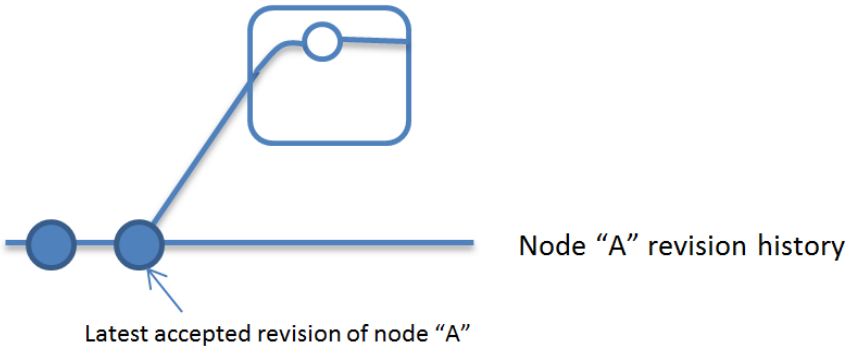
In this section we explain the structural and behavioral solutions related to multi-user operations in the DF. There are two solutions proposed, one based on the legacy code and one based on a redesign.

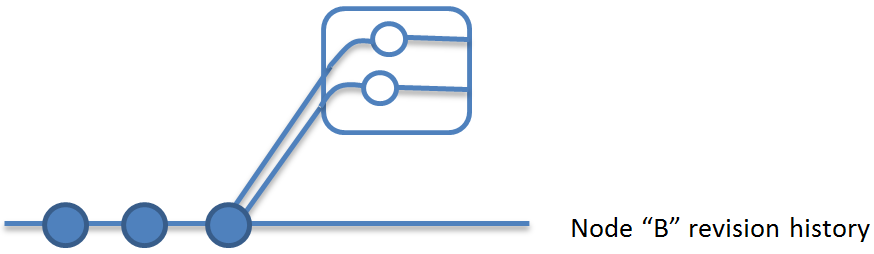
7.3.9.1 Solution based on the legacy code

In the legacy code, each node is versioned separately. Therefore, instead of System description Revision, we have Node Revision.

When a project is opened, the latest state of the system is attained. In order to do so, the latest accepted revisions of all the available nodes are calculated. Then the latest pending revisions for the user making those changes for all nodes are calculated and all are sent to the user.

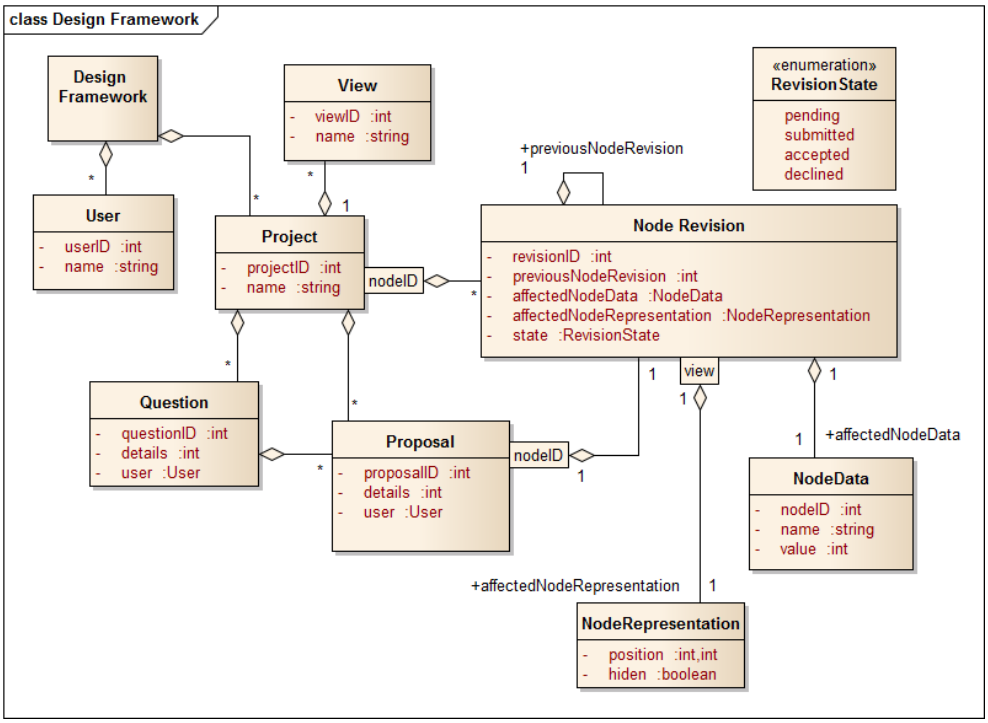
After the implementation of the submit button, the submitted proposals for all the nodes are calculated and they are sent to all users.





7.3.9.1.1 Class diagram

Here the class diagram of the proposed solution based on the legacy code is represented.

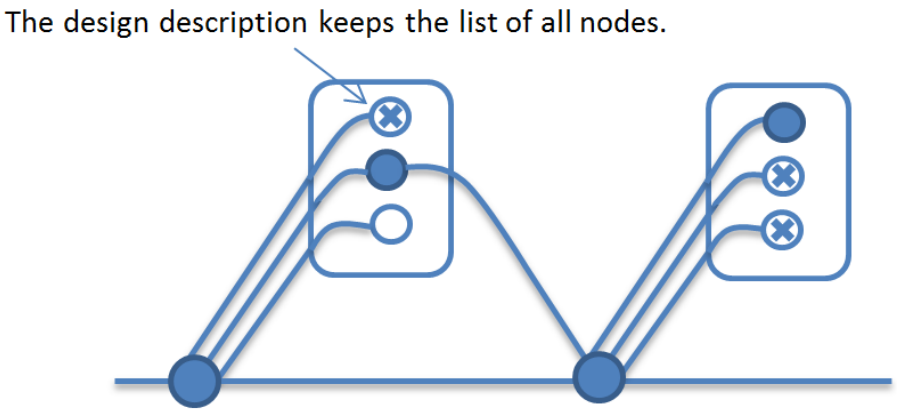


7.3.9.2 Solution based on a redesign

There are two main disadvantages of the current solution. The first one is its impossibility to revert the project to a certain moment in time. In the current design, it is possible to get the history of individual nodes, but it is not possible to see the snapshot of the system in a certain moment in the past.

The second disadvantage is that the system creates a new revision for each modification of a node. This does not necessarily add any overhead but it makes the revision tree more complicated than required and difficult to analyze. This is addressed in requirement 15 and it can be solved in the legacy code as well.

In this section we explain a new solution which is inspired by the design of the git revision system. In this solution, at any moment there will be a revision of a system description which includes all design objects. Thus, it solves the first disadvantage of the legacy design.



As explained earlier, at each system description revision, a set of node revisions are stored. If a node is modified in a new system description, the set of node revisions changes and the references to the new node revisions are updated.

Here we demonstrate the solution in an example. In the example described in Table 6, the first system description contains the node revisions 1, 2, and 3 which are in Table 7. This means that at this revision, which is the first revision, there are three nodes A, B, and C.

A user creates proposal 2 and proposes to rename node B to BB. In this proposal, nodes A and C are still the same. Therefore, the system description points to node revisions 1, 4, and 3 (only the reference to the new node is updated but still the list of all nodes is kept)

In this way, at any time, there is a revision that contains all the data and the system can be easily reverted to any revision.

Table 6 an example of the System description revision list

System description Revision	Previous Revision	Node Revisions	State	Project	Proposal
1	-	{1,2,3}	Accepted	1	1
2	1	{1,4,3}	Declined	1	1
3	1	{1,5,3}	Accepted	1	1
5	3	{6,5,7}	Accepted	1	2
6	3	{8,5}	Declined	1	2

7	3	{1,9,3}	Declined	1	2

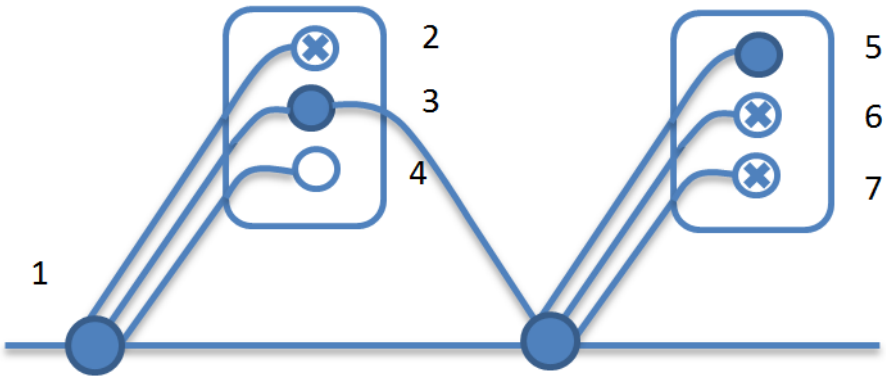


Figure 36 Proposal numbers related to the system descriptions

Table 7 an example of the Node revision list

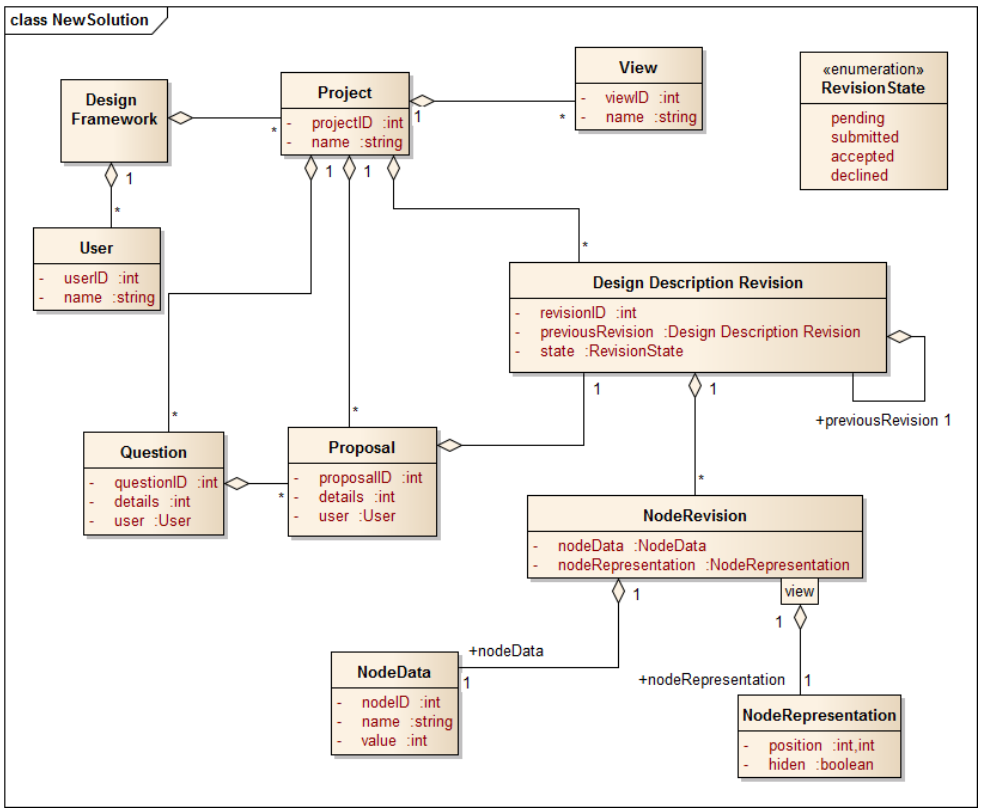
Node Revision ID	Node ID	Name	Value
1	1	A	10
2	2	B	20
3	3	C	30
4	2	BB	20
5	2	BBB	20

6	1	A	100
7	3	C	300
8	1	A	1000
9	2	BBB	200

In order to revert to a particular revision or even in order to get the latest revision, only one query on the system description table is required to get all the node revision references, and using these references, nodes can be retrieved from the node revision table.

7.3.9.2.1 Class diagram

In the new solutions revisions of nodes are kept separately. In each system description, a set of revisions are stored.



7.4 Results

In this section, the requirements for the multi-user support are analyzed. After a thorough analysis, certain problems are recognized and categorized in a list of requirements.

There are four main activities:

- Synchronization of data based on the legacy code is developed.
- The functionality of submitting a proposal is analyzed and developed.
- The construction of the design description tree is modified so that the tree cannot have a sequence of pending descriptions.
- Proposals can be switched and viewed separately.

These functionalities are developed and tested manually for simple scenarios. Additionally, the main functionalities are tested through set of test scenarios described in section 8.7.1.2.

8 Testability

Abstract- This chapter provides a detailed analysis of the Testability phase.

8.1 Introduction

The testability phase is started to provide test support to the DF development. The DF tool was developed without following a specific development approach. For this reason, one of the non-functional requirements "testability" was not addressed in the current version.

After conducting two FMEA sessions as mentioned in section 3.2, the requirement "testability" was finalized.

In one of the common test based development approach TDD (Test Driven Development), the test-first development approach is used. Which means, first a set of test cases are written, and then development code is written to make those tests pass. However, in the DF development, no such approach was used; hence test support is to be provided for the legacy and future code.

8.2 Web Application Testing

The web application testing technique is exclusively adopted to test the applications that are hosted on the web; the application interfaces and other functionalities are tested. Apart from functional testing of individual and integrated components, some of the testing types such as Performance, Security, Cross-browser and Responsiveness are also considered in web testing [10]. Major web testing techniques are listed below:

1) **Functionality testing**

   It includes testing all the links in web pages, database connection, forms used in the web pages for submitting or getting information from users. Besides, it includes test of workflow of the system and test of data integrity.

2) **Performance testing**

   Web application should sustain to heavy load. Web performance testing includes:
   - Web Load Testing
   - Web Stress Testing

   Web load testing includes the test against accessing or requesting the same page by multiple users. The application should handle many simultaneous user requests such as, large input data from users, simultaneous connection to DB, heavy load on specific pages.

   Web stress testing includes the test against the stress provided on input fields, login and sign up areas. The application should be able to handle the stress and recover from crashes.

3) **Interface testing**

   The main interfaces in web application are: web server and application server interface, application server and database server interface. Interface testing includes the test to verify proper execution of interaction between the servers and proper error handling. If database or web server returns any error message for any query from the application server then the application server should catch and display these error messages appropriately to users.

4) **Compatibility testing**
It includes the test for the compatibility of web application against different browsers, operating systems, mobile devices, and printing options. Applications should be able to execute the request in proper manner in a variety of diverse platform.

5) **Usability testing**
It includes the test to check if the provided instructions are correct and satisfy purpose of the application. The application should be easy to use. Instructions should be provided clearly.

6) **Security testing**
It includes test against security breaches. All transactions, error messages, security breach attempts should get logged in log files somewhere on web server. Test includes: pasting URLs directly in the browser without login, changing parameter of internal URLs, checking system reaction on all valid input on input fields.

Among the checklist provided above, we mainly focus on functionality testing and some usability support (e.g. HTML form validation and verification).

8.3 Functionality Testing

8.3.1 Traditional Approach

In a web application, there are two common approaches for functionality testing: End to End testing and Unit testing. However, these two approaches are not mere substitutes of each other; many times a mixed approach is preferred.

8.3.1.1 End to End Testing

End-to-end tests consist of requests made at the client side and observing whether the correct response is returned.

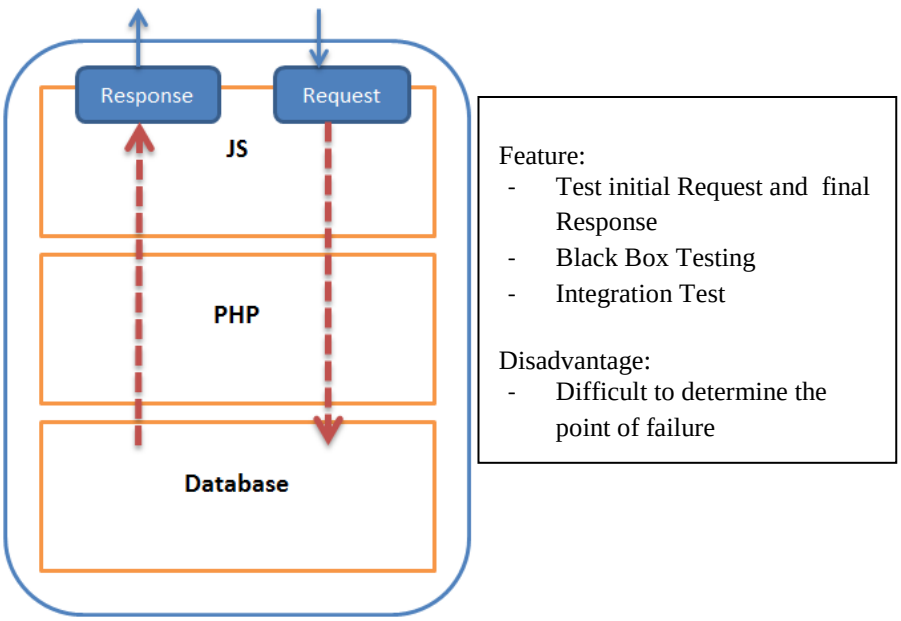


Figure 37 End-to-end Testing Approach

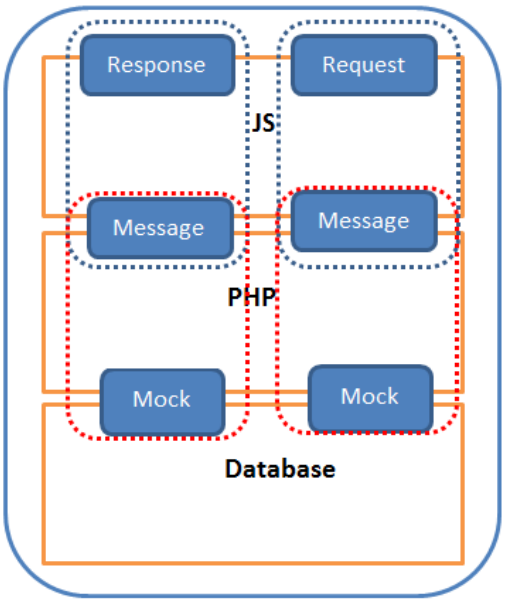
Example:

Create a DF Project:

1. Log in to the system (Pre-condition)
2. Hover over “Projects” menu
3. Select “Create project”
4. Fill in the project name
5. Press “OK”
6. Receive “Project successfully created” message

8.3.1.2 Unit Test (Isolating each layer)

Unit test consists of a test to confirm if each layer transmit and process message in a correct form. In the figure below, four units are shown. In the JS layer a unit test would be to check if correct message as intended is sent to the PHP layer. Similarly in PHP layer, test would be to check if correct response is send to client side after the correct request. A mock is used to deal with the database layer. Using a mock means the data to and from database is correct.



Feature:
- Test each layer separately
- White box testing

Disadvantage:
- Sometime units behave differently when integrated.

Figure 38 Unit Testing Approach

Example:

Create Project (JS):

1. Log in to the system (Pre-condition)
2. Hover over “Projects” menu
3. Select “Create project”
4. Fill in the project name
5. Press “OK”
6. Send “createProject” message to PHP Layer

Create Project (PHP):

1. Receive “createProject” request with project name from JS layer
2. Send project created message
3. Send list of projects

7.3.1.3 Hybrid Approach

In the hybrid approach, both end to end test and unit tests are performed. Some key features are selected for the end to end test and a similar list of key units is tested.

8.3.2 Behavior Driven Development

8.3.2.1 Introduction

Behavior Driven Development (BDD) is a software development process that emerged from the test-driven development approach [11]. Behavior-driven development combines the general techniques and principles of TDD with ideas from domain-driven design and object-oriented analysis and design to provide software development and management teams with shared tools and a shared process to collaborate on software development.

In this approach the team provides a significant portion of functional documentation in the form of user stories with executable scenarios or examples. Instead of referring to 'tests', a BDD practitioner uses the terms 'scenario' and 'specification'. BDD aims to gather in a single place the specification of an outcome valuable to a user, generally using the user stories, as well as examples or scenarios expressed in the form of given-when-then; these two notations are often considered the most readable.

In emphasizing the term 'specification', the intent of BDD is to provide a single answer to what many Agile teams view as separate activities: the creation of unit tests and 'technical' code on one hand, and the creation of functional tests and 'features' on the other hand. This should lead to increased collaboration between developers, test specialists, and domain experts. Rather than referring to 'functional tests', the preferred term is 'specifications of the product's behavior'.

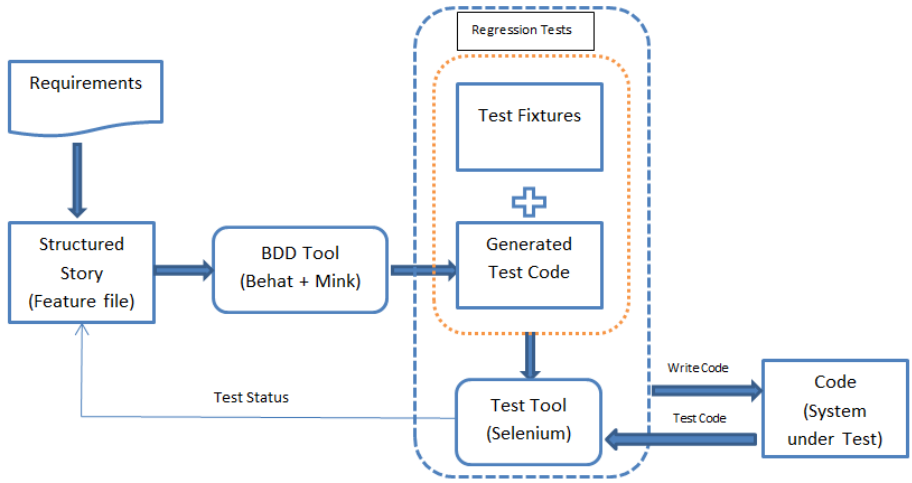


Figure 39 BDD Workflow

Example of executable specification written in .feature file:

```
# features/project.feature

Feature: Project
  In order to use DF Features
  As an end user
  I should be able to create project

  Background:
    Given I login as "user1"

  @javascript
  Scenario: Create Project
    Given I click on menu "Projects"
    And I wait for the page
    When I click on by id "createProject"
    And I wait for the page
    And I fill in "createProjectName" with "project1"
    And I click on "a.ok"
    And I wait for the page
    Then a project with id 1 and name "project1" is created
```

Feature indicates a user story which may contain one or more scenarios. The feature section is used to describe the actual feature whereas **Scenario** define possible test case. Scenarios are defined in a special format with given-when-then. Such structured requirement specification is defined in a special language called ***Gherkin,*** which is a business readable and domain specific language that allows describing software's behavior without detailing how that behavior is implemented.

In the Design Framework development, BDD addresses two different issues gathered during the FMEA session: "Testability" and "Requirements". Since BDD uses executable feature file as a test case, it couples requirements and tests together. Beside that the major advantage of using BDD is a shared understanding. It brings technical and non-technical people to the common understanding of features. Any team member can write tests cases and each of those test cases are easily understandable. Due to these advantages, BDD approach is used to address testability issue in the DF development process.

8.3.2.2 BDD Tool Support

Some available frameworks for BDD are listed in Table 8.

Table 8 Available BDD frameworks

Frameworks	Platform
Cucumber	Ruby
Behat	PHP
Behave	Python
Codeception	PHP

The Design Framework tool is developed using HTML5, JavaScript, PHP, and AJAX. In the current state, the most suitable BDD framework to test PHP-based web applications is Behat, so it is used as a test support framework for testing the DF tool.

Behat in combination with Mink provides a powerful test support for PHP based web applications. Mink was developed to address two different issues in headless browser emulator like Goutte and browser automation tool like Selenium [12]. A headless browser is a web browser without a graphical user interface. In other words it is a browser, a piece of software, that access web pages but doesn't show them to any user. They're actually used to provide the content of web pages to other programs. Selenium is slower than a headless browser but provides JS/AJAX test support. While testing web applications, we need both features (JS/AJAX test support and fast execution), and that is why Mink was developed.

Mink removes API differences between different browser emulators providing different drivers (Selenium Driver, Goutte Driver, etc.) for every browser emulator and providing the easy way to control the browser, traverse pages, or manipulate page elements.

Some features and tool support related to Behat framework is described in the Table 9.

Table 9 Features and Tool Support

Technology	Key features
Behat framework	- Open source BDD framework for PHP5.3 and above. - Recommended BDD framework for PHP based applications
Mink	- Open source bowser controller/emulator. - Combination of browser controller and headless browser emulator. - Works with Behat
Selenium tool	- Browser automation tool - Used for creating browser based regression automation tests - Slower than headless browsing tools - Open Source
PHPUnit	- Programmer oriented testing framework for PHP - Open Source
PhantosJS, Goutte (Headless browsers)	- APIs to crawl websites and extract data from HTML/XML responses - Fast execution - Unable to test JS/AJAX pages

Table 10 BDD tool support for the Design Framework

Design Framework(Technology Under Test)	BDD(Tool and Test Approach Support)
JavaScript	Selenium (*End to End Test*)
PHP	PhpUnit (*Unit Tests*)
Database	PhpUnit (*Unit Tests* using Mock Objects)

8.3.3 Functional Testing Checklist

1) Testing links

- Test for the internal and outgoing links.

2) Testing for validation and negative input

- Test for invalid inputs like empty field, use of special characters (*, %, $, #, @), character length.
- Test for validation of optional and mandatory fields.

Example:

- When I leave input field empty and click “OK” in createProject form, then I should see error message “Empty name not allowed”.
- When I input special characters (*, %, $, #, @) or too long characters for project name, Then I should see error message “Special character not allowed” or “Project length too long”.

Such client based validations are handled during the HTML form design and are later tested manually.

3) Testing workflow of the system

- Test for a full flow of a functionality.

Example:

- When I click on "Projects" in menu
- And I click on "createProject"
- And I fill in "createProjectName" with "project_name"
- And I click on "OK"
- Then a project with id 1 and name “project_name” is created

4) Testing data integrity

- Test for the maximum field lengths to ensure that there are no truncated characters while storing in database.
- Test how negative integer numbers are handled by database and test the semantics between data input and data stored.
- Check the maximum field lengths to ensure that there are no truncated characters.
- If numeric fields accept negative values, test whether such value can be stored correctly on the database and whether it is correct to accept negative numbers.
- If a particular set of data is saved to the database check that each value gets saved fully to the database; i.e., beware of truncation (of strings) and rounding of numeric values.

  Example:

  - When I create a project with character length 1000, check whether it is stored fully in database without truncating.

Checklist Summary – Standard Practice

- All the mandatory fields should be validated.
- Asterisk sign should be displayed for all the mandatory fields.
- System should not display the error message for optional fields.
- Numeric fields should not accept the alphabets and a proper error message should be displayed.
- Negative numbers if allowed should be validated properly.
- Division by zero should be handled properly for calculations.

- Maximum length of every field should be defined to ensure data is not truncated.
- Confirmation message should be displayed for update and delete operations.
- All input fields should be tested against special characters.
- All the functionalities of the available buttons should be tested.
- Proper error message should be conveyed whenever any of the functionality fails.
- All the uploaded documents should open properly.
- JavaScript should be tested to ensure it is working properly in different browsers (IE, Firefox, Chrome, safari and Opera).
- All the data inside combo/list box should be arranged in chronological order.

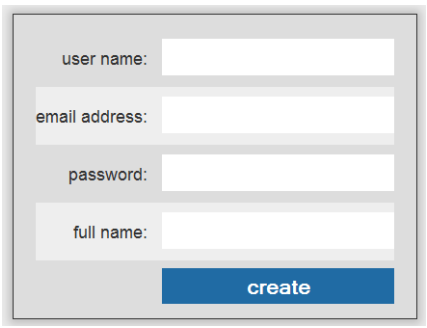
8.4 Validating HTML Forms

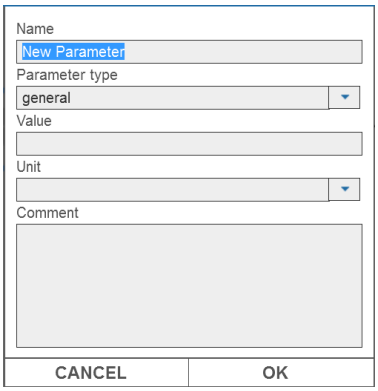
In the DF, a total of 48 HTML forms are used. Common DF-form validation issues are described below:

8.4.1 7.4.1 DF-form Validation Issues

1) Inconsistent Labeling across forms

- In the left image below, labels are in small case and with colon whereas in the right image, labels are in capital case without colon.
- Position of labels in these two images is inconsistent. In the left image, labels are in front of the input fields where in the right image, labels are above the input fields.





2) Lack of input field validation and proper messaging

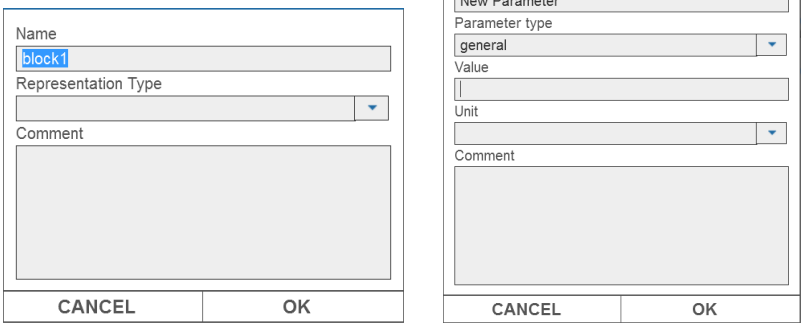
- There is no validation of input fields in most of the forms.

3) No indication of mandatory fields

- None of the mandatory fields are represented with * sign.

4) Inconsistent default position of cursor

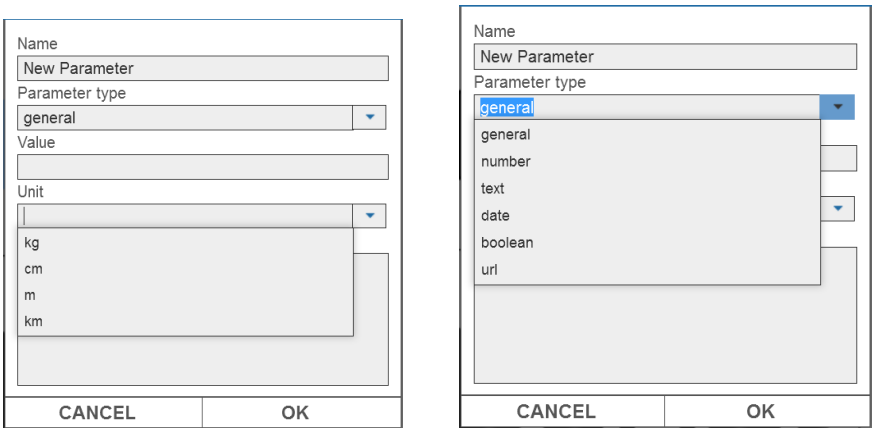
- Default position of cursor while editing blocks (Image on left) and parameter (image on right) are different. Default position of cursor while editing block is at the first input field as highlighted below, while position of cursor while editing parameter is the third input field.



5) Lack of proper handling of enabling and disabling of input fields and buttons

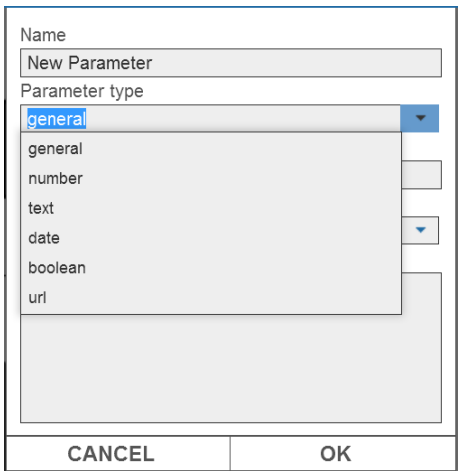
6) Inconsistent default values for dropdown

- In the right image below, dropdown of the "Parameter type" field has default value "general" whereas, in the left image, dropdown of the "unit" field does not have a default value.



7) Lack of alphabetical order of data in dropdown box

- In the image below the data in a dropdown are not arranged in an alphabetical order. As a standard practice, they should be arranged in an alphabetical order.



8) Data integrity issues (Consistency with form data and database fields)

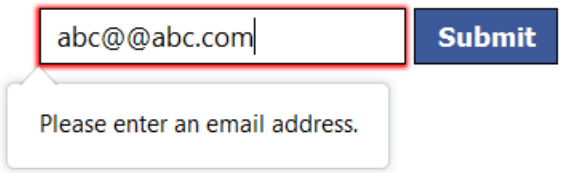
- Due to the lack of validation of input fields, length of input fields and length of same field in database are inconsitent resulting in data integrity issues.

8.4.2 7.4.2 HTML5 Form Validation Support

Unlike previous version of HTML, HTML5 comes with additional features. Proper use of HTML5 features will help to address form validation issues. Below are some of the most interesting features provides by HTML5:

1) The **type** Attribute

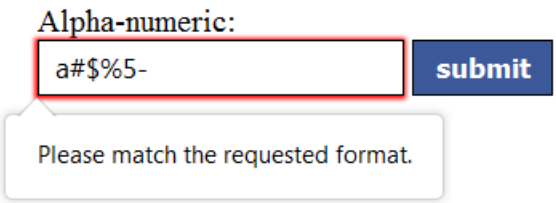
```
<input type="email" value="" placeholder="df@df.com" required />
<input type="submit" value="Submit" />
```



HTML5 added several new input types: color, date, datetime, datetime-local, email, month, number, range, search, tel, time, url, week. Just by using the attribute "type" in the above example, the input field is validated to accept only email format input.

2) The **pattern** Attribute
It specifies a regular expression to check the input value against the pattern.

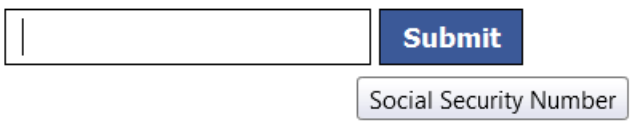
```
<input id="alpha" type="text" pattern="[a-zA-Z0-9]+" required>
<input type="submit" value="submit">
```



In the above example, use of pattern attribute with "[a-zA-Z0-9]" value, provides input field with a validation to accept only alphanumeric inputs.

3) Giving Hints – **title** Attribute
It specifies the hint for the input field.

```
<input type="text" name="ssn" pattern="^\d{3}-\d{2}-\d{4}$" title="Social Security Number" /
<input type="submit" value="Submit" />
```



In the above example, use of title attribute with value "Social security Number", provides hint to the users about the expected input.

4) The **required** Attribute

Specifies that an input field is required (must be filled out).

```
<input type="email" value="" placeholder="df@df.com" required />
<input type="submit" value="Submit" />
```

5) Validation against negative number

For the numeric input field, boundary can be set though max and min keyword.

```
<input id="id1" type="number" min="0" required>
<input type="submit" value="Submit" />
```

8.5 Test Strategy

In this section, two different test strategies which are used for testing the DF are described.

8.5.1 Unit Test

A unit test focuses on a single "unit of code" – usually a function in an object or module. By making the test specific to a single function, the test should be simple, quick to write, and quick to run. This means we can have many unit tests, and more unit tests means more bugs caught. Major benefits of unit tests are summarized below:

- It reduces the level of bugs in production code.
- It saves development time.
- Automated tests can be run as frequently as required.
- It makes it easier to change and refactor code.
- It forces developers to confront the problem head on.
- They are a measure of code completion.

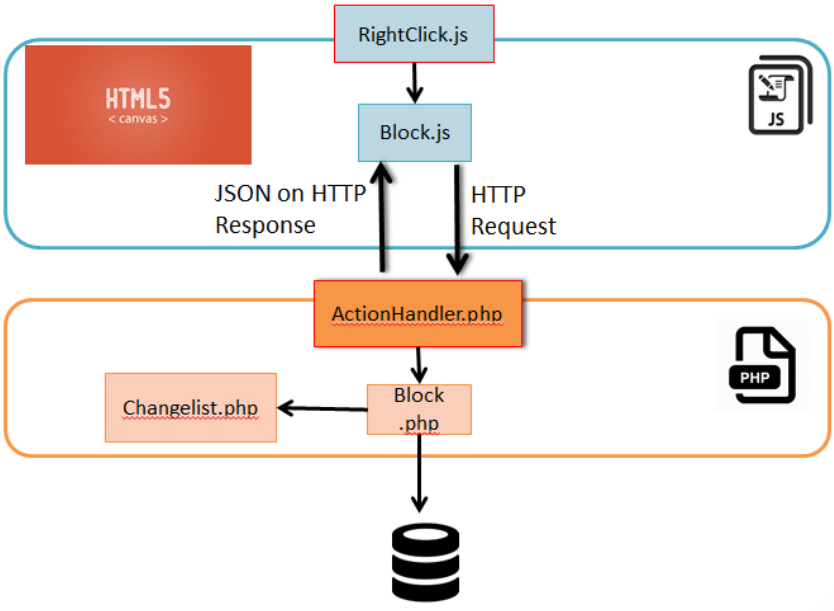


Figure 40 Create Block Scenario

In Object Oriented concept, unit testing applies to method, where each method is a unit. In the DF, features are more important, so each feature is considered as a unit while performing unit test in the DF. Figure 40 is an example of a scenario and a work flow from front end to backend. In this feature a request is send from front end

to back end through a right click event from the user. Backend will receive a HTTP request and in turn responds a respective JSON response. In the unit test, a set of HTTP requests is fed to the test and set of expected JSON responses are asserted to confirm the behavior. Figure 41 shows an example of HTTP request sent (left image) from front end and respective JSON response (right image) received from back end. This example shows the correct form of input and output expected in the test, HTTP request is the test input and JSON response is the test output. Code snippet of a unit test for creating and editing parameter is provided in appendix.

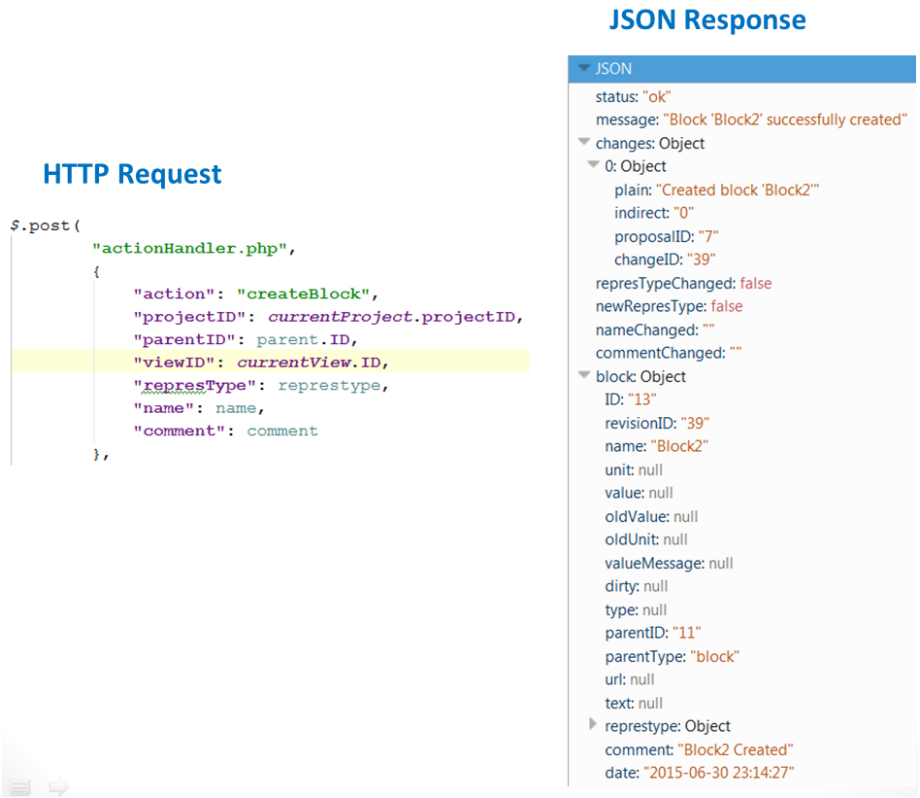


Figure 41 HTTP Request & JSON Response

Set of features to be tested are provided in Table 11.

8.5.2 End to End Test

Unit test addresses a unit in a whole feature. Unit tests are not sufficient in many occasions and similar is for the Design Framework. Sometime behavior of a system after integration may not be correct even if units are correct. Due to this reason set of End to End test is needed to make the DF even more robust.

In contrast to unit test, in End to End test Behavior Driven Development approach is used which makes test cases more readable and understandable. The main reason behind the use of BDD test approach in the DF is to minimize the gap between technical and non-technical users who are concerned with DF testing. Additional description about readable test cases and the technology behind BDD is discussed in Behavior Driven Development section.

For a “Create Block” feature as shown in Figure 40, Figure 42 is an example of test written using BDD approach.

```
# features/block.feature

Feature: Block
  In order to use block
  As an end user
  I should be able to create it

  Background:
    Given I login as "user1"
    And I created a project "project1"

  @javascript
  Scenario: Create block
    Given I right click on "block" with id 1
    And I select menu item "addBlock"
    And I wait for the page
    And I fill in "editBlockName" with "block1"
    When I click on "a.ok"
    Then a node with id 2 and type "block" is created
```

Figure 42 Create Block Feature

Each of the statement provided in Given-When-Then-And, are mapped to a particular function and hence the test is performed. Set of features to be tested using BDD approach are provided in Table 11.

8.6 Testing Challenge

In this section a testing challenge and a solution to that challenge is described.

8.6.1 Incompatible Technology

The technology used in front end of the Design Framework is HTML5 Canvas and the technology used for testing the application through record and play is Selenium.

The Canvas is stateless; it does not keep reference of object above it, which makes it impossible to record user interaction. It is not possible to inspect elements and make assertions about the state of a canvas like in a DOM (Data Object Model). DOM defines the logical structure of HTML documents and the way a document is accessed and manipulated.

Selenium follows DOM based testing approach. It searches the path inside the html tags and executes the action to that particular location. It records the position of DOM provided by HTML and can further simulate click event according to the DOM position.

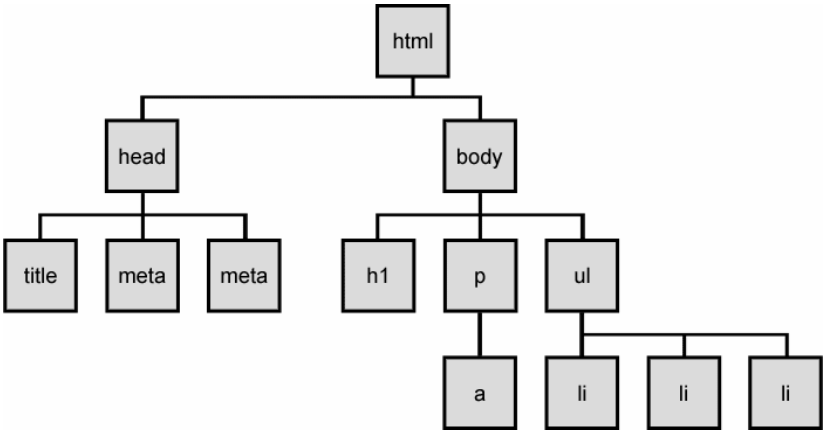


Figure 43 Data Object Model overview

When objects are rendered using canvas tags they are unavailable for Selenium tests because it does not keep reference of those objects. Since the DOM is no longer available (for free) inside the canvas tag, apps that live 100% inside the canvas tag will no longer be tested/recorded by Selenium.

8.6.2 Solution for Incompatibility

To address the above problem it is necessary to write a wrapper as an adapter between two incompatible APIs. Such wrapper should be able to access the objects inside the canvas and provide that state to selenium.

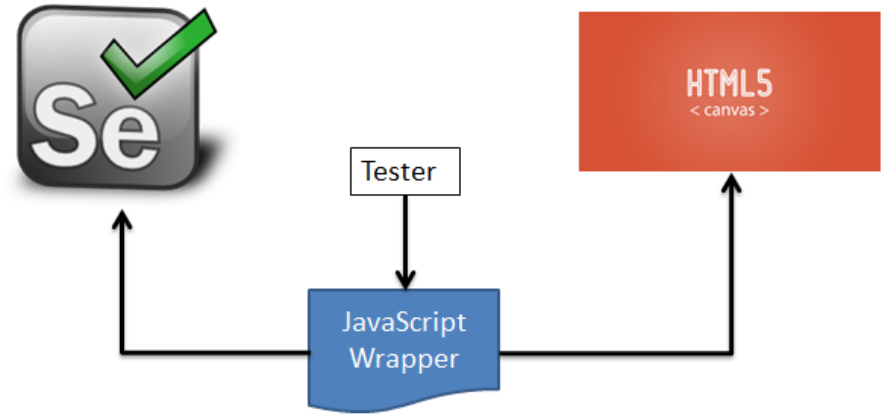


Figure 44 Use of JavaScript Wrapper

Browser cannot keep the coordinates of object clicked inside Canvas unlike normal DOM objects. Due to this reason main challenge to create the wrapper API was to store the instance of Canvas. Whenever an object was opened in a Canvas the coordinates and type of that object was stored in an object form, and later that object was exposed to Selenium.

Creating such object helped to keep the track of all objects in Canvas which removed the problem of finding where exactly the object is located. We could than search any objects in Canvas in current instance. Finally overriding the click event provided by the browser helped to click in a particular object and hence the issue of incompatibility was resolved.

Code snippet of the JavaScript wrapper is provided in appendix.

8.7 Scope and Goal

8.7.1 Features to Test

8.7.1.1 Single User Features

The Design Framework development will be supported by three different kinds of test sets. First, the PHP based Unit tests using PHPUnit, second, the JavaScript based End to End test per DF concepts(Project, Block, Parameter, Relation, Dependency, Text, Image, Experiment, Model) using BDD framework Behat, and third, the JavaScript based Flow Test (End to End) per scenario using Behat. Second and third types of test are both End-to-End test. Only a difference between the two is the second test covers all the features per concept, whereas the third type of test includes single features from all concepts covering almost all major features in one test scenario.

Set of test cases are provided in Table 11 below. Additional set of scenarios can be created as per the need.

Table 11 Features to Test

Concepts	**Test Cases**		
	Unit Test (PHPUnit)	JS Test - E2E (Behat + Mink)	Flow Test – E2E (Behat + Mink)
Project	- Create Project - Delete Project - Create Duplicate Project - Open Project - Update Project	- Create Project - Delete Project - Create Duplicate Project - Open Project - Update Project	- Create Project
Block	- Create Block - Edit Block - Delete Block - Get History	- Create Block - Edit Block - Delete Block - Get History	- Create Block
Relation	- Create Relation - Edit Relation - Add Relation IO - Remove Relation IO - Get History	- Create Relation - Edit Relation - Add Relation IO - Remove Relation IO - Get History	- Create Relation
Parameter	- Create Parameter - Edit Parameter - Delete Parameter - Update units - Get History	- Create Parameter - Edit Parameter - Delete Parameter - Update units - Get History	- Create Parameter
Validation	- Create Validation - Edit Validation - Delete Validation - Get History - Add Dependency IO - Remove Dependency IO	- Create Validation - Edit Validation - Delete Validation - Get History - Add Dependency IO - Remove Dependency IO	- Create Validation
Transformation	- Create Transformation	- Create Transformation	- Create Transformation

	- Edit Transformation - Delete Transformation - Add Dependency IO - Remove Dependency IO - Get History	- Edit Transformation - Delete Transformation - Add Dependency IO - Remove Dependency IO - Get History	
Text	- Add Text - Edit Text	- Add Text - Edit Text	- Add Text
Image	- Add Image - Edit Image - Upload Image	- Add Image - Edit Image - Upload Image	
Model	- Create Model - Edit Model - Get History	- Create Model - Edit Model - Get History	- Create Model
Experiment	- Create Experiment - Edit Experiment	- Create Experiment - Edit Experiment	- Create Experiment
Representation Type	- Create Representation Type - Edit Representation Type - Delete Representation Type	- Create Representation Type - Edit Representation Type - Delete Representation Type	- Create Representation Type
Proposal	- Accept Proposal - Decline Proposal	- Accept Proposal - Decline Proposal	

Beside the tests listed in Table 11, some input field validation tests are provided using Behat.

Example:

- Test against special characters (%&*$#@) in input field
- Test against blank name in input field

8.7.1.2 Multi User Features

"Multi User Support" is a newly developed feature of the Design Framework tool. A set of test scenarios are created and tested to verify the multi user feature. Three different test cases are listed below:

1) Test Case 1

Scenario: Change block name by both users, user2 accepts the change earlier than user1.

Expected Result: Revision of user1 changes from "Block name changed from 'b1' to 'b2'" to "Block name changed from 'b3' to 'b2'".

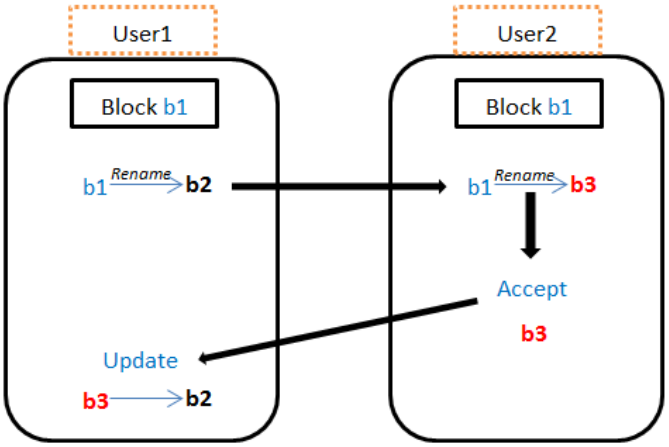


Figure 45 Multi User Scenario 1

Test case 1 can be described in feature file as in Figure 46 below.

```
# features/multiUserScenario1.feature

Feature: Multi User
  In order to use multi user feature
  As an end user
  I should be able to synchronize changes

  Background:
    Given I login as "user1"
    And I created a project "project1"
    And I created a block "b1"
    And I accept proposal

  @javascript
  Scenario: Second user accepts proposal earlier then the first
    Given I edit name of block with id 2 to "b2"
    And I check the revision history
    When I login as "user2"
    And I open project "project1"
    And I edit name of block with id 2 to "b3"
    Then I accept proposal
    And I wait for few seconds
```

Figure 46 Feature file of Scenario 1

2) Test Case 2

Scenario: Change block name by both users, user2 also adds a comment and accepts the change

Expected Results:

- Revision of user1 changes from “Block name changed from 'b1' to 'b2'” to “Block name changed from 'b3' to 'b2'”
- User1 can see the comment in b1

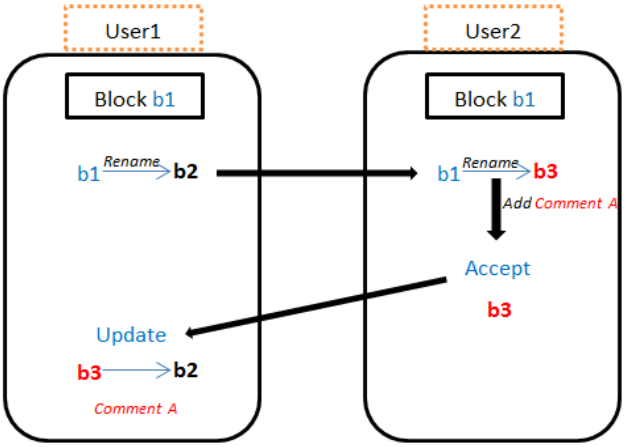


Figure 47 Multi User Scenario 2

3) Test Case 3

Scenario: Change block name by both users, user1 accepts and immediately user2 accepts before getting updates from user1.

Expected Result: User2 gets warning message about the update from other user.

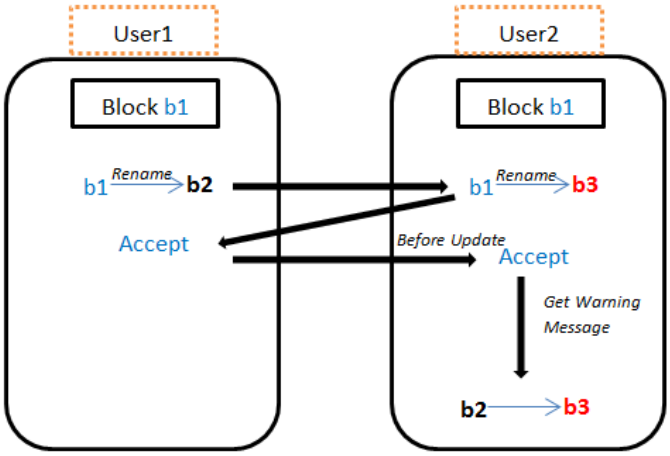


Figure 48 Multi User Scenario 3

4) Test Case 4

Scenario: User 1 changes block name and submits proposal, later created two different proposals and accepts one of them

Expected Result: User2 is able to collaborate with the changes from User 1. Accepting one proposal removes the changes in other proposal.

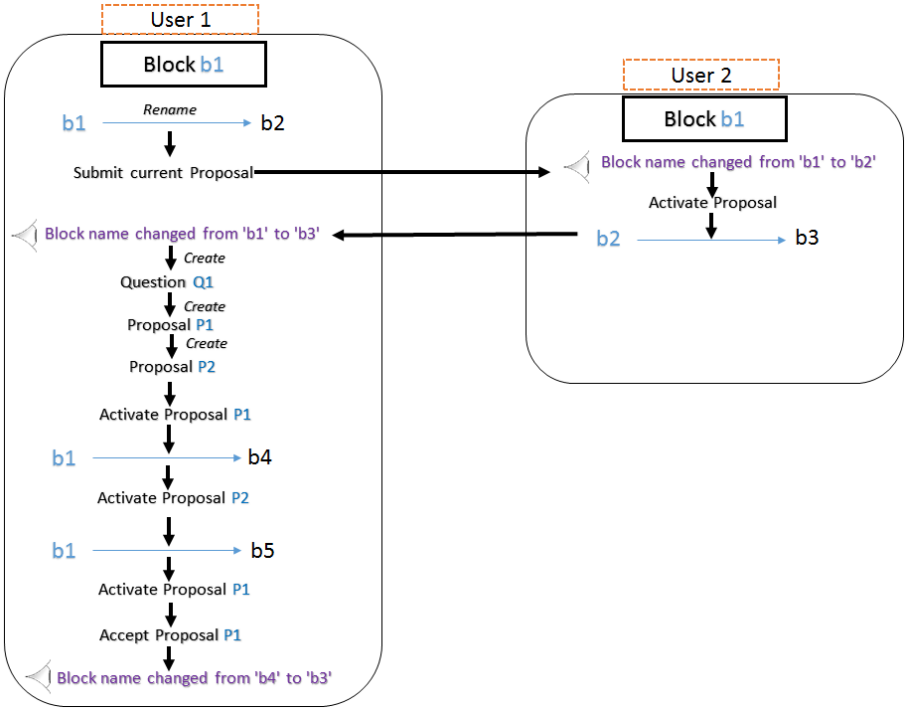


Figure 49 Multi User Scenario4

Test case 4 can be described in feature file as in Figure 50 below.

```
# features/multiUserScenario4.feature

Feature: Multi User
  In order to use multi user feature
  As an end user
  I should be able to synchronize changes

  Background:
    Given I login as "user1"
    And I created a project "project1"
    And I created a block "b1"
    And I accept proposal

  @javascript
  Scenario: Accepts proposal before getting updates from other user
    Given I edit name of block with id 2 to "b2"
    And I submit proposal
    When I login as "user2"
    And I open project "project1"
    And I check the revision history
    And I wait for the page
    And I should see "Block name changed from 'b1' to 'b2'"
    And I activate proposal with id 3
    And I edit name of block with id 2 to "b3"
    And I login as "user1"
    And I open project "project1"
    And I check the revision history
    And I wait for the page
    And I should see "Block name changed from 'b1' to 'b3'"
    Then I create new question "q1"
    And I create new proposal "p1"
    And I create new proposal "p2"
    And I wait for the page
    And I activate proposal with id 4
    And I wait for the page
    And I edit name of block with id 2 to "b4"
    And I should see "Block name changed from 'b1' to 'b4'"
    And I activate proposal with id 5
    And I wait for the page
    And I edit name of block with id 2 to "b5"
    And I should see "Block name changed from 'b1' to 'b5'"
    And I activate proposal with id 4
    And I wait for the page
    And I accept proposal "p4"
    And I check the revision history
    And I wait for the page
```

Figure 50 Feature file of Scenario 4

8.8 Results

In this section, results obtained during the testability phase are described.

After the analysis of the requirements through the FMEA analysis, test support was provided to the DF through unit tests and Behavior Driven Development approach.

Three different kinds of test sets are delivered, which are as follows:

- PHP based Unit tests, using PHPUnit

- JavaScript based End-to-End test per DF concepts, using BDD framework Behat

- JavaScript based End-to-End test for multi-user feature, using BDD framework

The delivered result does not cover all the possible unit tests and end-to-end tests. However, it covers major test cases which can be used as a reference to write more test cases in future, as per the need.

The verification and validation of the result was done with the help of test cases, following the existing use cases and newly developed multi-user feature. Existing use cases are known and were verified by other engineers involved in the DF. Similarly, test case related to multi-user feature was verified by Arash who worked in that feature.

Above test sets are currently used by TNO-ESI. Provided test sets and the approach used to setup test environment made the DF system more stable and maintainable. Current and future developers will be benefited from the provided tests and approach.

Most importantly, the use of BDD based testing approach has made it easy to write and understand tests, for both technical and non-technical people involved in the DF.

9 Conclusions

Abstract – In this chapter, we summarize the results obtained during the project as well as the lessons learned.

9.1 Results

There were two parts in this project, which are delivered separately. The first part concerns the refactoring of the DF. The second part concerns the additional multi-user feature and testability.

The result of the first part is that the refactored project is delivered successfully and it is currently used in the main line. The number of lines is reduced from 12000 lines of code to 8000 lines of code.

The reduction of the number of lines makes the code base more maintainable and reduces the complexity of the code.

The number of the tables in the database is reduced from 31 to 10. This means less data redundancy and more data consistency in the database.

The multi-user part of the project includes an analysis of the requirements for the multi-user support. The requirements are analyzed and they are partly implemented and used in the main development line.

The testing part of the project includes providing test support to the Design Framework. The requirements were gathered and analyzed and finally test support was provided to the Design Framework through unit test and Behavior Driven Development approach. A set of unit tests and end-to-end tests including the test for multi user support is delivered.

Results obtained in each phase of the project is currently used by TNO-ESI.

9.2 Lessons Learned

The first part of the project concerns the refactoring of the DF. During the refactoring process several lessons which were learned.

The first lesson is that in order to refactor a code, it is always important to have a test framework in place. The DF did not use any test framework and therefore during the refactoring process, every time a change is made, a manual test is required to verify if the system is still working properly.

The second lesson learned is that it is very important to break down the process of software refactoring into meaningful small independent processes. Each small process should maximize the number of homogenous changes which are possible to be performed in a single step preferably during one day. By trial we found a balance for the number of changes that the team was able to perform.

The third lesson learned was that it is very important to define the requirement scope as soon as possible otherwise requirement becomes vague. Requirement of the pro-

ject was not clear in both phases, so it was necessary to go through the available materials, generate set of requirement and discuss and negotiate with stakeholders.

The fourth lesson learned was that regular communication with stakeholders about the progress or issues (if any) is very important, it brings all stakeholders on the common ground. Many times technical issues lead to delay in the progress as committed but after clearly explaining the bottleneck, requirements were adjusted.

10 Project Management

Abstract – In this chapter, we explain about the project management during this project. This project is partly done in pair and partly as an individual.

10.1 Introduction

This project was separated into two parts. The first part of the project was done in a group of two and the second part of the project was done individual.

During the first part of the project, an agile approach was practiced. The requirements were split into parts. The sprint was defined as one week. Every morning, in a stand-up meeting each member described what they were going to do during the day. Tasks were estimated weekly and a burn chart was made every day after the compellation of the tasks.

The multi-user part of the project was more of a requirement analysis. The approach in project management was different from the first part. A document was prepared and risk management was done on a weekly basis to determine if the analysis was going in the right direction.

The testability part of the project started with the requirement gathering. After FMEA session testability requirement was ensured. It was an individual task, so project management was different then the first phase. Each week set of progress was demonstrated keeping the final milestone in mind.

10.2 Work-Breakdown Structure

The nine-month duration of the project was broken into three parts. The first part, which was done in a group of two, concerned reverse engineering and redesign of the initial implementation.

The second part of the project, which was done individually. One of the requirements was to implement a multi-user feature and other requirement was to provide test-support to the DF system. The final part of the project was spent on documentation.

Of the nine months of project time, five months was spent on the first part, three months on the second part, and one month on the last part.

10.3 Project Planning and Scheduling

The assignment was to refactor a code and it started in a team of two persons. In order to do so, a system analysis was required at the start of the project. The plan was to analyze the system for two months. After two months we came up with a list of problems to solve. Based on the problems, we spent one month to experiment, prototype, and propose a new design for the database. We spent one month to redesign the PHP part. At the end we delivered the first version.

After this period, we came up with a set of new issues to solve, so we had to spend half a month in order to solve new issues. At the end of this period, we delivered the second version.

From this point the project was divided into two parts. One person had to build a test framework and the other person had to implement the multi-user support for the DF.

In multi-user part, one month was spent on gathering the requirements and three months in implementing the features. Third version was delivered at the end of this phase.

In testability part, one and half months were spent on research and requirement finalizing and another one and half month for implementation. Set of test cases and the test framework was delivered at the end of this phase.

Finally both of us spent one month writing the report. The rest of time was spent on maintenance of the system.

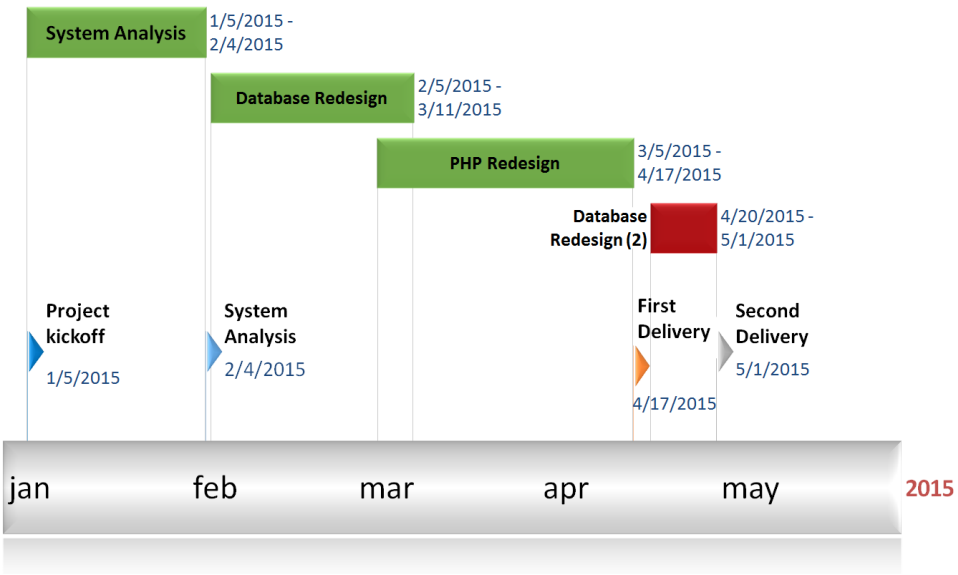


Figure 51 Gantt chart of the first phase of the project

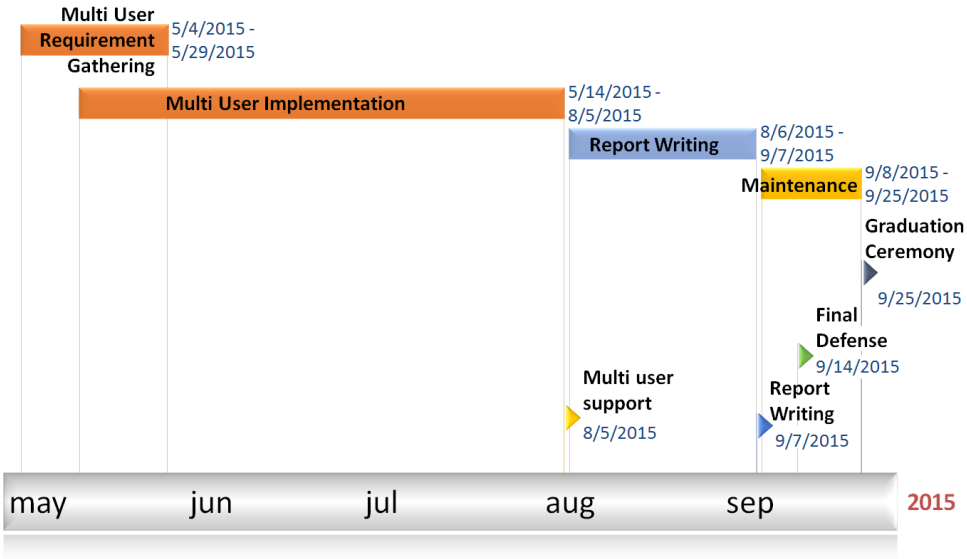


Figure 52 Gantt chart of the multi-user phase

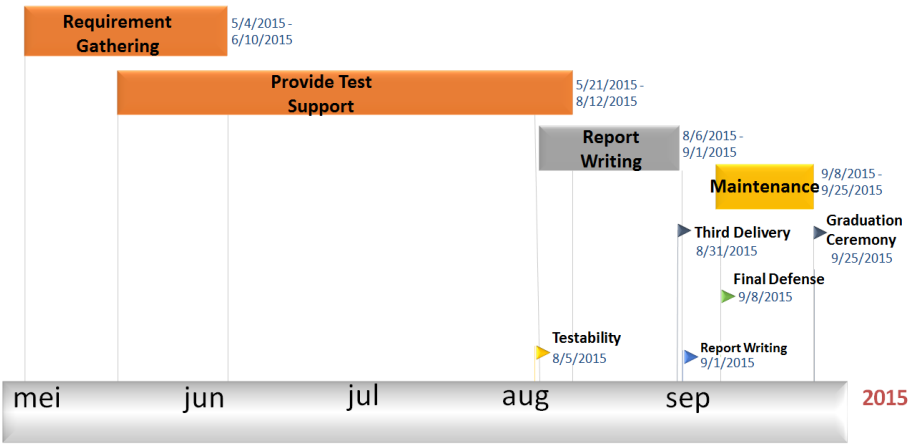


Figure 53 Gantt chart of Testability phase

10.4 Milestone Trend Analysis

There were three different milestones. First milestone was in April, after PHP redesign and initial database redesign.

Second milestone was in May, after redesigning database for the second time to reduce the inner joins. Change in database also resulted in the need of change in code. A migration code was delivered besides the refactored code in order to migrate from the old database to the new database.

Finally third milestone was at the end of August after implementing multi-user and testing support.

10.5 Conclusions

The project planning went quite smoothly, although at the start the scope was not very clear. The database design and PHP design overlapped at the start but we tried to separate them in order to take control over the process.

The multi-user requirement gathering and implementation were predicting to be in two completely separate phases but at the end they overlapped.

In a nutshell, the planning was flexible and changed a few times during the process, but the milestones were reached.

11 Project Retrospective

This chapter summarizes the overall result indicating success failure analysis of the project.

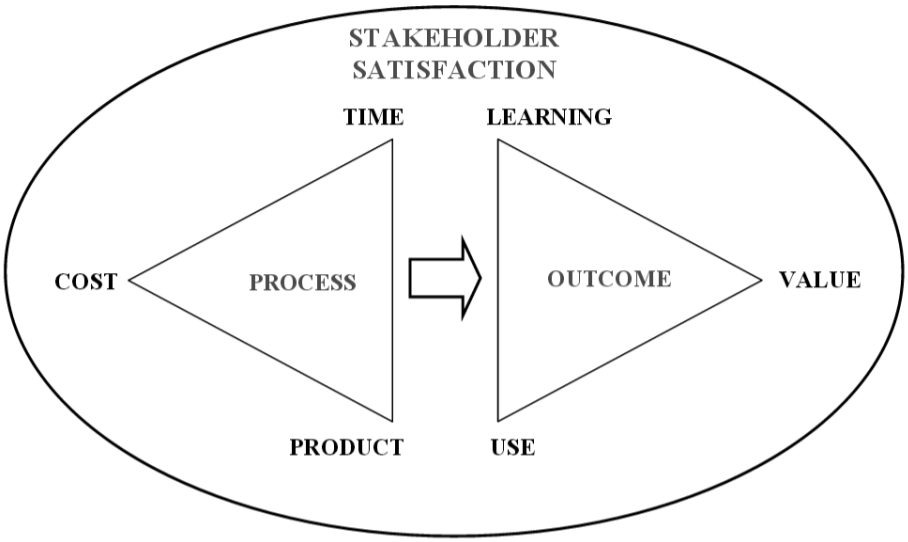


Figure 54 Project Success Criteria

Evaluating project success should include both process and outcome criteria [13], as illustrated in Figure 54.

The three process-related criteria include:

1. Time: The project came in on schedule.
2. Cost: The project came in on budget.
3. Product: The project produced a product of acceptable quality and met other product-related specifications, including requirements, usability, and ease of use, modifiability, and maintainability.

The three outcome related criteria include:

1. Use: The project's resulting product/service is being used by its target constituencies.
2. Learning: The project increased stakeholder knowledge and helped prepare the organization for the future challenges.
3. Value: The project will directly result in improved efficiency and/or effectiveness for the client's organization.

Taken together, the six criteria above yield a more comprehensive view of the project success.

As mentioned in the section 1.2 different stakeholders are interested in different outcome of the project. TNO-ESI supervisor is interested in consistent design, implementation and documentation, TU/e supervisor is interested in a consistent design and quality report, and we are interested in enhancing design skills and learning new technology which will add value for our future career.

Looking at the above mentioned six criteria, project was successfully completed. Set of goals were assigned for the pre-defined duration of nine months. At the end of the available time two different milestones were reached: First, with the team of two people to redesign the Design Framework and second, an individual assignment to provide new multi-user and testing support. Outcome from both milestones are used in the current development branch by TNO-ESI.

12 Bibliography

[1] "Design Framework Official Website," [Online]. Available: http://df.esi.nl/.

[2] P. P. L. a. H. V. V. Kruchten, "Building up and reasoning about architectural knowledge," *Quality of Software Architecture,* pp. 43-58, 2006.

[3] "SysML Official Website," [Online]. Available: http://sysml.org/.

[4] R. H. T. P. Hristina Moneva, "A Design Framework fr Model-based Development of Complex Systems," *32nd IEEE Real-Time Systems Symposium and 2nd Analytical Virtual Integration of Cyber-Physical Systems Workshop,* pp. 1-8, 2011.

[5] ISO/IEC/IEEE 42010, System and software engineering - Architecture description, International Standard Organization, 2011.

[6] "Apache Friends," [Online]. Available: https://www.apachefriends.org/index.html.

[7] M. Fowler, Refactoring: Improving the design of existing code, Addison Wesley, 1999.

[8] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley Professional, 2003.

[9] "Propel," [Online]. Available: http://propelorm.org/Propel/documentation/09-inheritance.html.

[10] "Tutorials Point," [Online]. Available: http://www.tutorialspoint.com.

[11] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Behavior-driven_development.

[12] "Mink Behat," [Online]. Available: http://mink.behat.org/en/latest/.

[13] R. R. Nelson, "Project Retrospectives: Evaluating Project Success, Failure, and Everything in Between," University of Minnesota , 2005.

Appendix

P: *Probability*
I: *Impact factor*

Table 12 FMEA Analysis - Developers

Scenario	P	I	P*I	Rank
Architecture and Design				
There are no distinctive error codes users can respond on	**5**	**1**	**5**	**4**
No clear development practice (design, develop, text, document)				
No clear design and documentation				
Build and installation is not enough documented				
Missing high level architecture document				
Design choice are not documented				
No clear iterations, sprints, releases				
No clear development practices				
Testability				
No clear test cases and no automation	**3**	**4**	**12**	**1**
Requirement				
No clear priorities in design roadmap	**5**	**2**	**10**	**2**
Initial requirements are not clear				
No clear prioritization				
No clear requirement for multi user				
No clear insight in usage(end-use scenarios)				

Usage				
No clear view of the installed base	**2**	**2**	**4**	**5**
No realistic usage scenarios from practice and industry				
No access to end user data				
Tool				
No proper IDE support for development and debugging	**3**	**2**	**6**	**3**
No proper execution trace logging or debug tool				
Deployment				
Performance not well understood	**2**	**5**	**10**	**2**
Client environment affects behavior				
Server environment affects performance				
Dependency on Oce` with little support				

Table 13 FMEA Analysis - End User

Scenario	P	I	P*I	Rank
Documentation				
No usage examples provided	**4**	**2**	**8**	**6**
No clear distinction on which DF version do I have now, which new features were introduced				
No local help (On the server and context sensitive)				
Inspirational corner “Architectural patterns”, where to find and what to share?				
Representation / Position Inconsistencies				
Unwanted position of blocks when unhiding or refresh-	**4**	**3**	**12**	**3**

ing				
Unwanted behavior of visible elements				
Screen does not look like last time opened(Expected)				
When I reopen my DF session, it should have the same look and feel even after upgrading(no messed up views)				
Another user have messed up my view				
Data Inconsistencies				
Lack of multiuser support	**4**	**5**	**20**	**1**
Changes are overridden				
Lack of consistent view				
Do not delete things that are still in use				
Error prone				
Multi user inconsistency				
I want to have warnings for inconsistencies				
Browser Support				
My browser is not fully supported	**3**	**3**	**9**	**5**
DF in tablet, only left click works				
Browser dependency				
Message Handling / Performance				
Lack of proper messaging	**5**	**3**	**15**	**2**
Slow execution				
No feedback on “longer actions”, user presses again				
Too slow in longer actions				

When DF backend is doing a time consuming task, I want to know the set of progress				
Bad performance indication				
“Busy” sign is confusing				
Better error and completion message				
Unclear error messages, what to do?				
Error messages visible only for few seconds then they disappear.				
UI				
When someone changes my models, I want to be notified	**5**	**2**	**10**	**4**
Not enough color contrast				
Too many clicks				
UI is not very intuitive (too many clicks, difficult to find function, not consistent behavior and representation)				
Representation types of other projects are present in my project which creates confusion, same for unit.				
Actions can be optimized but need end user input.				
Simplicity, I want to see only the essential concepts.				
Check progress per user “custom queries”				
External Service/Server				
Breaking server	**2**	**4**	**8**	**6**
I don’t care that service X is not started, everything should work always				
Which models are supported in this instance?				

Server is down (What to do? Whom to call?)				
If any service is down on the server, I get the notification (Example: Excel)				

JS Wrapper API Code Snippet

```
/**
 *This method checks if particular node is present in the canvas and if present right clicks on that node
 */
function clickNode(nodeType, nodeID) {
    var clickedX = -1;
    var clickedY = -1;
    var clickData = null;

    if (currentProject == null) {
        throw new Error("There is no project displayed in canvas");
    }
    else {
        if (nodeType.toString() == "block") {
            clickedX = currentProject.blocks[nodeID].x;
            clickedY = currentProject.blocks[nodeID].y;
            clickData = {x: clickedX, y: clickedY};
        }
        else if (nodeType.toString() == "parameter") {...}
        else if (nodeType.toString() == "image") {...}
        else if (nodeType.toString() == "text") {...}
        else if (nodeType.toString() == "relation") {...}
        else if (nodeType.toString() == "transformation") {...}
        else if (nodeType.toString() == "validation") {...}
        else if (nodeType.toString() == "experiment") {...}
        else if (nodeType.toString() == "model") {...}
        else {
            throw new Error("Node type:" + nodeType.toString + " does not exist in current project!!");
        }
        var SimulatedClick = new SimulatedClickEvent(clickData);
        showRightClickMenu(SimulatedClick);
    }
}

/**
 *This method simulates the click event by overriding value of pageX and pageY
 */
function SimulatedClickEvent(data) {
    this.pageX = data.x;
    this.pageY = data.y;
    this.preventDefault = function () {
    }
}
```

Unit Tests Snippet

```
public function testCreateParameter(){
    $res = Parameter::createParameter(1, 1, "param1", 1, "general", "This is param1", 5, "kg");
    $res_json = json_decode($res);
    $this->assertEquals($res_json->status, "ok");
    $this->assertEquals($res_json->parameter->{'ID'}, "5");
    $this->assertEquals($res_json->message, "Parameter 'param1' successfully created");
    $this->assertEquals($res_json->changes{0}->{'plain'}, "Created parameter 'param1'");
    $this->assertEquals($res_json->changes{1}->{'plain'}, "Comment of Parameter 'param1' changed");
    $this->assertEquals($res_json->changes{2}->{'plain'}, "Parameter type of 'param1' changed");
    $this->assertEquals($res_json->changes{4}->{'plain'}, "Parameter unit of 'param1' changed from '' to 'kg'");
}

public function testEditParameter(){
    $res = Parameter::editParameter(5, 1, 1, "param2", "This is param2", 7, "number", "");
    $res_json = json_decode($res);
    $this->assertEquals($res_json->status, "ok");
    $this->assertEquals($res_json->parameter->{'ID'}, "5");
    $this->assertEquals($res_json->message, "Parameter saved");
    $this->assertEquals($res_json->changes{0}->{'plain'}, "Parameter name changed from 'param1' to 'param2'");
    $this->assertEquals($res_json->changes{1}->{'plain'}, "Comment of Parameter 'param2' changed");
    $this->assertEquals($res_json->changes{3}->{'plain'}, "Parameter type of 'param2' changed");
    $this->assertEquals($res_json->changes{4}->{'plain'}, "Parameter unit of 'param2' changed from 'kg' to ''");
}
```

About the Authors

Navaraj received his MS in Computer Science and Information Engineering from National Taiwan University of Science and Technology, Taiwan in 2012. He did his MS thesis on Information Retrieval entitled “Reviewer Recommendation using Academic Tag Extraction based on Boolean and Vector Space Model”. His thesis proposed a recommendation system which assists journal and conference editors to find suitable reviewers for the proposal. After his studies he worked as Software Engineer in Leapfrog Technology Inc., a software development company based in Nepal. Between 2013 and 2015 he worked at the Stan Ackermans Institute of Eindhoven University of Technology, where he completed the Software Technology program.

Arash was born in March, 1987 in Isfahan, a city in the center of Iran. He graduated with a B.Sc. in Software Engineering at University of Isfahan in Isfahan, Iran and moved to France to pursue his graduate studies. He graduated with an M.Eng. in Computer Science at ENSEEIHT in Toulouse, France in 2011. For the final project of his master thesis, he joined the media system research laboratory in Singapore as a Research Assistant. He continued his research in Multimedia system laboratory at Telecom ParisTech in Paris, France. As an industrial experience, he completed a professional degree in engineering at Eindhoven University of Technology.

3TU.School for Technological Design, Stan Ackermans Institute offers two-year postgraduate technological designer programmes. This institute is a joint initiative of the three technological universities of the Netherlands: Delft University of Technology, Eindhoven University of Technology and University of Twente. For more information please visit: www.3tu.nl/sai.