From Requirements to Software: Research and Practice

Scientific Editors

Piotr Kosiuczenko, Michał Śmiałek

This publication was supported
by the **Ministry of Science and Higher Education**
within the program related to the implementation
of tasks of science dissemination
(Decision No 530/P-DUN/2015 on 27/01/2015)

POLISH INFORMATION PROCESSING SOCIETY

From Requirements to Software: Research and Practice

Scientific Editors

Piotr Kosiuczenko, Michał Śmiałek

Warszawa 2015

Authors

Marek Majchrzak, Łukasz Stilger – ***Chapter 1,*** *Aneta Poniszewska-Marańda –* ***Chapter 2,*** *Mariusz Postol –* ***Chapter 3,*** *Krzysztof Wnuk, Emilia Mendes –* ***Chapter 4****, Zbigniew Huzar, Małgorzata Sadowska –* ***Chapter 5,*** *Bogumila Hnatkowska –* ***Chapter 6,*** *Dariusz Gall, Anita Walkowiak* ***– Chapter 7,*** *Stan Jarzabek, Kuldeep Kumar* ***– Chapter 8,*** *Sławomir Samolej, Tomasz Rogalski* ***– Chapter 9,*** *Rakesh Rana, Andrzej Ratkowski, Miroslaw Staron, Christian Berger* ***– Chapter 10,*** *Jarosław Wojciechowski* ***– Chapter 11***

Reviewers
Zbigniew Banaszak, Włodzimierz Bielecki, Leszek Borzemski, Krzysztof Cetnarowicz, Zbigniew Czech, Włodzimierz Dąbrowski, Mariusz Flasiński, Janusz Górski, Adam Grzech, Piotr Habela, Leszek Maciaszek, Jan Madey, Lech Madeyski, Karolina Muszyńska, Jerzy Nawrocki, Mirosław Ochodek, Piotr Poprawski, Andrzej Stasiak, Krzysztof Stencel, Jakub Swacha, Zdzisław Szyjewski, Lech Tuzinkiewicz, Bartosz Walter, Andrzej Wasowski, Janusz Zalewski, Krzysztof Zielinski

Scientific Editors
Piotr Kosiuczenko, Michał Śmiałek

Technical Editor
Tomasz Klasa

ISBN 978-83-60810-72-9

Edition: I. Copies: 200.
Publisher: Polish Information Processing Society
Printed by Westgraph Łukasz Piwowarski, Przecław Poland

Contents

Preface

In the late 1960's, the discipline of software engineering emerged as a response to the so-called software crisis caused by the growth of software complexity. In response, methods from various engineering disciplines were applied. New methods for software development and maintenance and new modelling languages have been developed. Various tools supporting software development have emerged. Contemporary software life-cycle contains various disciplines that range from requirements engineering to software construction and software evolution. Requirements engineering includes such elements as requirement elicitation, analysis, specification and validation. Software design is devoted to defining software models and entire architectures of software systems, including their components and interfaces. The discipline of software construction includes creation of good quality executable software through coding, validation and testing. Software maintenance and evolution covers all activities aimed at supporting the usage of software, such as correction of existing faults, improvement of its performance and introduction of necessary extensions.

An important aspect of software engineering is software life-cycle process engineering. It concerns such topics as: specification, implementation, assessment, measurement, management and adjustment of the process that leads to producing quality software. Software engineering management includes activities such as planning, coordination, measurement, monitoring, controlling, and reporting, in order to properly direct software development and maintenance. As part of it, configuration management includes identification of systems' configurations at distinct points in time in order to control their changes, maintain their integrity and traceability throughout their life-cycle.

The above mentioned topics within the software life-cycle call for significant levels of methodological and tool support. Contemporary tools cover both the engineering and managerial aspects. They support creating various software artefacts, and automate transition from requirements to design, code and tests. On the other hand, software project managers can introduce dedicated tools that support organization of software teams and facilitate progress reporting and controlling.

In general, software systems are characterized by high and constantly growing complexity. This continuous growth cases the need to improve and further develop existing methods, languages and tools, but also to propose new ones. In this monograph, we report selected advancements in this area. The monograph covers topics starting from requirements management, to software design and implementation. First, it approaches these topics as parts of an overall engineering process that has to be managed efficiently. Then, it presents methods to automate translation from initial (requirements-level) artefacts down to design artefacts and code. Finally, it focuses on applying specific techniques to the currently predominant embedded and web systems. Although the focus of the monograph is mainly scientific, it includes some reports concerning practical applications.

Part 1 is devoted to software management. In its first chapter, Marek Majchrzak and Łukasz Stilger report on their experience concerning introduction of the Kanban methodology into an automotive software project. Kanban is a managerial method originating from Japan, with the emphasis on just-in-time delivery. Authors argue that in numerous software projects boundaries between traditional and agile approach methods disappear and that they require continuous scheduling of tasks without dividing them into sprints or strict project phases. Moreover, customers expect more flexibility and responsiveness from software vendors. To achieve better results in this field, authors used Kanban, in particular different Kanban boards and stakeholders. They describe its main advantages, ways to improve customer cooperation and stakeholder relationships, visualisation of task statuses and risk management.

In the second chapter, Aneta Poniszewska-Marańda and Rafał Włodarski report on adapting Scrum in the context of academic education. Scrum is the most widely used agile process framework for software development. It is iterative as well as incremental and defines a flexible and holistic development strategy where developers work in coherent units. It enables them to self-organize by encouraging physical co-location or close online collaboration and daily face-to-face communication. The key factor is the recognition that during a project, requirements may be changed by customers. Scrum aims at maximizing the units' ability to respond to new or modified requirements and to deliver conforming software quickly. This chapter is specifically devoted to the issue of Scrum adoption in the academic setting in case of specific requirements.

In the next chapter, Mariusz Postół discusses agile management of research projects in the context of contracts. Agile management approaches guide software development projects towards valuable outcomes and take into account unpredictability of project development. The author proposes to apply such approaches to high risk innovative research projects based on fixed-price contracts. He proposes also a methodology based on Scrum and supporting tools, which allows one to harmonize and embed agile principles as contractual rules. The discussion is based on a corresponding case study.

The last chapter of this part, authored by Krzysztof Wnuk and Emilia Mendes, is devoted to a literature review concerning project management perspective on software value. The starting point is the claim that companies, in order to remain competitive and to grow, have to change from cost-based decision-making to value-based decision-making in a way to maximise the software value and the overall value creation. The objective is to complement and expand an existing classification of value and other aspects within the context of product management and development. The authors identify nine primary studies in two snowball iterations and derive three categories: finance, risk analysis and process improvement based on value identification.

Part 2 of this monograph is devoted to software modelling and code generation. It begins with chapter 5, authored by Zbigniew Huzar and Małgorzata Sadowska, proposes the creation of complete business process models. The authors notice that the Standard Business Processes Modelling Notation neglects modelling of data and information. Therefore, they define a compound model aimed to integrate BPMN diagrams with UML diagrams describing data structures. Based on an analysis of BPMN models with respect to their internal consistency and completeness in the process of requirements elicitation, the authors propose a business model based on the integration of BPMN with UML class diagrams and state machines. A simple example illustrating the compound model is worked out and evaluated.

Chapter 6, the second of this part, authored by Bogumiła Hnatkowska, is devoted to automatic translation of ontologies written in Sumo to UML. Ontologies are treated as the source of domain knowledge. The author notices that ontologies can be extracted and used to create domain models. The extraction process can be supported by tools that enable searching for relevant notions in an ontology and to automatically translate selected elements to other notations. The chapter presents a proof-of-concept tool for translating ontologies expressed in Sumo to UML. This tool shows that such transformation is feasible and, under some conditions, can produce high quality, consistent, correct and complete models.

The topic of the seventh chapter is a pattern-aware method transforming Platform Independent Models to Platform Specific Models. Dariusz Gall and Anita Walkowiak describe a method to transforms structural and behavioural aspects of PIM models into PSM models, with implementation details for the Java platform and show how to generate system implementation. The described method arranges the ingredient transformations, each of them with own execution parameters, into a transformation chain. The authors argue that the resulting transformation chain can be seen as a specification of a software architecture satisfying selected requirements.

This part ends with chapter eight, which addresses the problem of separation of tightly coupled concerns with the help of generic program representations. Separation of Concerns and genericity are fundamental principles allowing one to better manage software complexity during the software life-cycle. Some of the concerns identified at the concept level can be separated during the design or the implementation phase through modularisation and other techniques such as AspectJ, AHEAD or MDSOC. Whereas other concerns may not be so easily separable due to complex interactions with the rest code. In this chapter, Stan Jarząbek and Kuldeep Kumar show that generic program representations can not only avoid repetitions and consequently simplify programs, but also can enhance the visibility of inseparable concerns. They offer a weaker, nevertheless useful, form of Separation of Concerns. The authors explain dependency between these two principles and argue that there is an overlapping area where the goals of the two principles, as well as means to achieve them, are the same.

Part 3 of this monograph is devoted to embedded and network systems. In chapter nine, Sławomir Samolej and Tomasz Rogalski discuss the development of an experimental real-time pitch angle control application following the ARINC 653 and ARINC 664 standards. The authors describe the concept of Distributed Modular Electronics and explain the ARINC 653 specification. They report on developing and evaluating an application of ARINC 653, which is a real-time avionic control system. They consider specific timing parameters, built-in self-testing procedures and final system tests.

The next chapter is about improving dependability of embedded software systems using a framework called Fault Bypass Modelling. Fault injection techniques are widely used to test dependability in case of hardware electronics and software systems. Increasing complexity of embedded software systems, in particular in the automotive sector, has driven the use of Model Based Development and of virtual test environments to construct and test designs before code generation. In this chapter, Rakesh Rana, Miroslaw Staroń, and Christian Berger argue that fault injection techniques can be effectively used for assessing and thus increasing the dependability of embedded software systems. They analyse a problem that arises when fault injection is used during virtual simulation of such systems and evaluate the Fault Bypass Modelling framework as a potential solution to this problem.

Chapter eleven discusses a process that leads from an academic project to production software based on the Java web-tier Content Management System application. Jarosław Wojciechowski identifies the aspects of tiered applications which are essential in case of production-ready software. The author uses an example of custom techniques for functional solutions like multi-hierarchy, multi-domain operability in application performance gain practices. These were applied in a web-tier Content Management System application developed and used at the Lodz University of Technology.

Piotr Kosiuczenko and Michał Śmiałek

I. Software Management

Chapter 1

Experience Report: Introducing Kanban into an Automotive Software Project

1. Introduction

Lean thinking is important because it can dramatically reduce error rates. It has been shown that when applying this approach in the manufacturing or in service organisation, the productivity has at least doubled. Moreover, this method also significantly reduces delivery time for new products and decreases overall costs [14,9].

We shall also describe 2 software projects in the automotive industry, which have employed the Kanban technique in an evolutionary way. In each of these cases, Kanban was used to optimize a different process and was motivated by other business problems. However, the mutual characteristic was the simplification of processes and evolutionary adaptation of both the developer teams and the collaborating client teams.

2. Lean Software Development

The principles of Lean thinking focus on value added for the customer [6]. By removing the unnecessary processes, activities and artifacts, and on the other hand organizing work as a continuous flow, which recombines labor into cross-functional teams dedicated to that activity and constant improvements across the entire company we have been able to develop, fabricate and sell with half or less of the human effort, tools and overall costs. By introducing ’Lean thinking’ and its associated style of operation, we have been able to react faster and more flexibly to the ever-changing needs of our Clients and the modern market. Lean thinking requires continuous learning, growth and most importantly, commitment and understanding from the personnel of any level including management.
Lean Software Development is the application of Lean Thinking to the software development process. The Poppendieck and Poppendieck [16] illustrated how many of the Lean principles and practices can be used in software engineering context. They have proposed 7 principles eliminating and managing the waste in software development:

- EliminateWaste - Do only what adds value for a customer, and do it without delay.

- Amplify Learning - Use frequent iterations and regular releases to provide feedback.
- Delay Commitment - Make decisions at the last responsible moment.
- Deliver Fast - The measure of the maturity of an organization is the speed at which it can repeatedly and reliably respond to customer needs.
- Empower the Team - Assemble an expert workforce, provide technical leadership and delegate the responsibility to the workers.
- Build Integrity In - Have the disciplines in place to assure that a system will delight customers both upon initial delivery and over a long period of time.
- See the Whole - Use measurements and incentives focused on achieving the overall goal.

3. Kanban in Software Engineering

The name ”Kanban” originates from Japanese and could be translated as ”signboard” or ”billboard”. It is a flow-control mechanism for pull-driven ”just-in-time” production. The idea behind Kanban is to execute Lean principles in practice.
David J. Anderson has defined [1] 5 Kanban core principles, which are mainly overlapping Lean principles.

- Visualize the workflow - You have to understand what it takes to get an item from request to completion.
- Limit WIP - Limiting work-in-progress implies that a pull system is implemented on parts or all of the workflow. New work is ”pulled” into the new activity, when there is available capacity within the local WIP limit.
- Manage Flow - The flow of work items through each state in the workflow should be monitored and reported.
- Make Process Policies Explicit - The process needs to be defined, published and socialized explicitly and concisely.
- Improve Collaboratively (using models & the scientific method) - The use of models allows a team to make a prediction about the effect of change (or intervention).

In IT projects, using ”Kanban” is becoming increasingly popular regardless of the project stages or production methods. A very interesting aspect of this technique is that it becomes an inside tool in both waterfall and agile processes.
Kniberg [7] points out that Kanban is less prescriptive than other agile methods like RUP, XP or even SCRUM.
Scrum, XP and RUP are highly adaptive, but Kanban leaves almost everything open. The only constraints are Visualize Your Workflow and Limit WIP which make it a great tool for quick and effective workflow and process-management tool. Especially, in case when prescribed rules and artifacts don’t fit the project’s needs. Scrum prescribes the

use of timeboxed methods, but in the case of a support team or in the case of a fire-fighting team, it is hard to plan tasks in a sprint timebox.

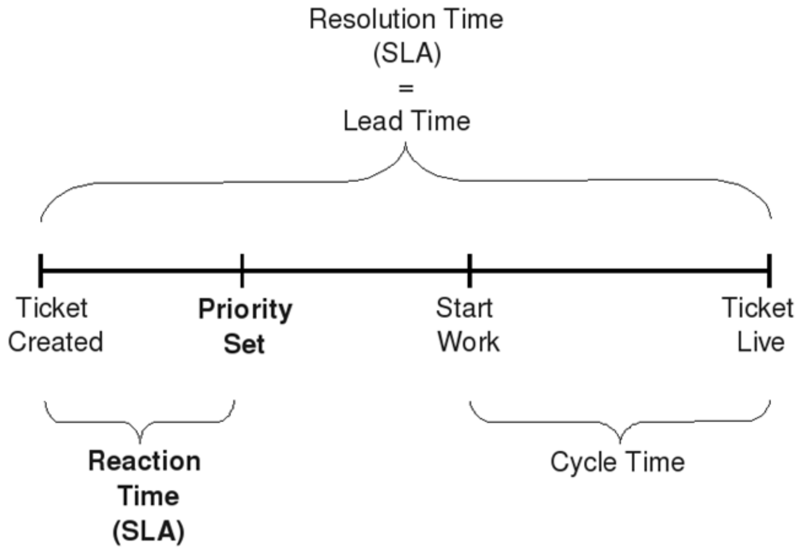


Figure 1. Lead Time & Cycle Time

3.1. Metrics - the way to observing facts and finding bottlenecks

In order to make decisions, the management of a given project requires capabilities for adequate situation analysis [15]. This role is served by project metrics, correctly selected criteria, according to which defined parameters can be observed.

A simple visualisation is a fantastic way of investigating the workings of a team and the current state of progress. However, it is mainly employed in the day-to-day planning. When more accurate analysis based on a larger volume of data is needed, it is crucial to create metrics, or information-gathering schemes. Metrics are collections of updated and adequately represented data used for problem identification and decision-making. The key concepts in measuring work efficiency are:

- Lead Time: a total time measured from task creation until its finish. Lead time takes into consideration all of the corresponding events between, both predicted as well as unpredictable.
- Cycle Time: is the correct volume of work.

Figure 1 is used to portray these two concepts. It must be noted here that entry and exit points for work units, as well as in-between points are defined in each project. The purpose of both of these metrics is to show the current work efficiency and potential decrease in time and costs of delivering a valuable work unit.

4. Releated Work

Mattias Jansson, Operations Engineer at Spotify[1], introduces [11] Kanban in the operations team as an answer to the growing number of different kinds of tasks. The team before they started testing Kanban, noticed that although they were quite efficient, they weren't able to plan far in advance. The problem was, they were re-active and not pro-active. The growing amount of 'urgent' jobs from other departments always were more important that the internal tasks and the context switching lessened the team's effectiveness. They realized that the company was growing faster than we could accommodate. Soon after Kanban's introduction, they've noticed that their lead times have become shorter, they have gotten more internal tasks done, and the departments they interface with have been happier.

Kniberg in his book "Lean from the Trenches" [8] described PUST - a digital investigation system for the Swedish national police authority. Due to the project scale, the teams, as well as the kanban boards, were divided into subsystems. Beside having WIP limits in case of regular tasks, they also decided to limit the number of bugs reported in the bug tracker. In case of blocker priority, the bug had to be fixed immediately, if less important, it had to be replaced with an existing one from the top thirty, otherwise it would be ignored. He claimed that such an approach not only allowed for effective communication (lower number of bugs, highly prioritized bugs were immediately fixed etc.), but also avoided long change control meetings to manage a long lists of bugs, which would probably never be fixed anyway.

Ikonen et al. [5] conducted a study in the middle size project (13 developers) in the R&D field. The investigation focused on the following project work aspects: documentation, problem solving, visualization, understanding the whole, communication, embracing the method, feedback, approval process, selecting work assignment. The presented results indicated considerable benefits of the Kanban technique including team motivation and control over project activities. Most of the work aspects were positively supported by Kanban techniques inside the Team.

Middleton and Joyce in their BBC Worldwide case study [13] showed that following a Kanban technique introduction, the lead time to deliver the software improved over 37% and the number of defects reported by customers decreased by 24%. They noticed much the same obstacles, which may occur after lean introduction connected with the environment and work space as a tension within the existing corporate standards and processes. Especially: office space designed inappropriately for Kanban boards, Kanban and reduction of WIP won't work with milestones and Gant charts, close team co-operation with the customer may be seen as working beyond the remit and self managing team of specialist may be challenging to the managers.

They have noticed that the Agile approach, especially Scrum, has some similarities. However, they also noticed that the Kanban technique and Lean have several advantages over Agile/Scrum approach. They claim that WIP limits and pull work model

[1] Spotify is a music streaming service for desktops and smartphones, which aims to provide a wide-ranging music collection

compared to Scrum Push and timeboxed approach, reduced delivery time and allowed for software with better quality. They also noticed that the ownership and responsibility of the Scrum ”impediments list” is diffused. On the other hand, Lean team because of limited WIP and visualization on the kanban boards must solve the problem immediately if they are blocked. In this case, all staff members are obligated to eliminate the bottlenecks.

In another case study by Middleton et al. [12] he analyzed a Timberland company practicing ”Lean thinking” for two years. They noticed many steps in their processes not adding value. A survey among people in the company showed that the majority supported lean ideas and thought they can be applied to software engineering. Interestingly only a minority (10%) was not convinced of the benefits of lean software development. The company showed a 25% gain in productivity and time for defect fixing was reduced by 65% - 80%. The response on the product released using lean development from customer site was overall positive.

5. Discovering Kanban

In this section we shall describe two different approaches and two different perspectives of Kanban introduction. In Project A[2] 2 Kanban has been introduced as a tool for dealing
with unplanned tasks in Sprint. In Project B the main goal was to unblock communication in an extensive stakeholders structure.

5.1. Project A

5.1.1. Background

The system, under investigation, covers all aspects of car purchasing in one of the premium car manufacturers in Germany. The system was designed for experts and is being used internally by customer. Basically, it allows buying, leasing or renting cars by customer employees, institution or VIPs, management of car fleet and used cars reselling.
Currently the system consists of 2 main components. One is the new version of the system developed as a modern web-based application. The second component written in COBOL is the old system version, which is to be replaced by a new version step by step. Both systems are always available and data is synchronised between them in real time.
The project uses the Scrum framework with certain small additional procedures, like additional Scrum of Scrums meeting and PO-Team meeting. A typical Sprint takes 3 weeks, some of the user stories (US) are approved during the sprint, some at the end - during the demo. The majority of US are confirmed and tested by the PO-Team, however in the case of larger epics there are more people involved, including a number of external IT specialists

[2] Due to a commercial agreement, the project names have been anonymous.

5.1.2. A Timeline

The old system version has been developed since 1990 using waterfall software development model. The new version of the systems was developed at the beginning also using the waterfall software development model. Development started in 2010. First release after 16 months showed that we were not able to integrate with the old system and we did not cover minimal end user needs. In 2012, it was decided, that in order to improve cooperation between 2 systems and ensure faster delivery, new requirements for the whole project will be developed as one Scrum project. After final transition in 2013, the entire team as well as the customer were using the aforementioned Scrum framework. Currently, a new version is being released quarterly. In case of urgent requirements we will provide minor releases extending the latest production version.

5.1.3. A Timeline

The Team consists of 35 persons. Around one-third of them are connected with the project from initial stage. Approximately half of the workers have got several years of experience in leading enterprise projects. The entire team is divided into 7 sub-teams (see Figure 2), some of them are virtual. The team member could be assigned to more than one team because of his/her function.

- JEE Development Teams (x3, DT) are responsible for the new system version, they use Scrum. Each team has about 6 members and a Scrum Master.
- Host Team is responsible for developing old system in a Cobol technology. The team consists of 5 members and Scrum Master.
- Fire Fighting and Support Team (FT and ST) consists of 6 members. In most cases they are nominated in the sprint beginning from each development team. The team is responsible for integration and production of bug fixing and for providing 3rd line support. Members of the team change every Sprint session.
- Cross Functional Team (CT) consists of technical leaders from each development team and solution architects. Focuses on long-term technical and business decisions. Designs new components and supports customer.
- Product Owners (PO) Team consists of 3 business architects focused on new User Story development. They work and agree new functionality directly with end users and major stakeholders within the organization..

Persons, who lack industry experience are used up to a 70% capacity in the initial few months of the Sprint.The slack time is being used for internal project training provided by experienced software developers and architects.
Project Teams are located in 3 different cities.This type of work has been organised in accordance with the Distributed Scrum concept as described by Majchrzak et al. [10].

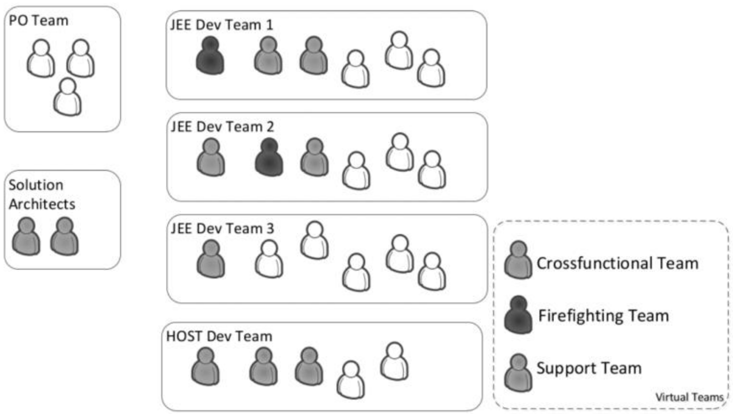


Figure 2. Project A - Team Structure

5.1.4. Engineering Practices

From the very beginning, we were focused on XP techniques [4] which could be applied in both the waterfall and then in Scrum framework:

- test-driven development (unit tests);
- clean code instead of code documentation;
- automated end 2 end (E2E) testing covering the user stories;
- continuous integration after each source code change, nightly build includes long running e2e regression tests;
- source control software and rigorous configuration management;
- bug-tracking software (JIRA [2]);
- documentation in wiki (Confluence [3]).

This results in high test coverage. The team is able to provide a new release after each sprint. Due to E2E testing, business and technical complexity, results are not always stable. About 15% of the tests are failing regularly. The problems before release are being checked manually in order to ensure that the problem reported occurs because of new functionality or because of bugs. Every time a bug is found, it is promptly reported in the issue tracking system.

5.1.5. Kanban introduction stages

The main impulses for employing Kanban were doubts the team was having in regard to bug correction and new feature implementation, which had not been explicitly defined during the sprint planning. It is worth noting that the problem did not consist only of what and in which order a given task or mistake was to be corrected, but also a decision determining the incorrect workings of an application, as well as the decision about who would be the sponsor of a given change.

Step 1: Identify work to be done. The first step aimed at systemizing the volume of work being done outside the sprint was to create one list of errors and problems from various sources, and then organize it in the JIRA system. Within the project, because of diverse users and conditioning, the aforementioned errors and queries could be called in using many ways, i.e. by email, by telephone but also with the help of HPQC and Peregrain. From the developer's perspective, many sources of those were impossible to accommodate and respond to, much the same which decision had a higher priority.

Unified bug list wasn't the right solution. Very quickly we have found the following drawbacks:

1. A Developer had to arrange who would be the sponsor of change or error - fixing.
2. In the case of lack of symptoms, many tasks had the status of In Progress. It is worth noting, that a developer was usually only assigned for approximately 2 weeks to the FT team. As a result of it, the said developer would have begun many tasks (the work in progress was not defined or limited) and practically would not complete any in real life. Then tasks would be returned to the 'To Do list' and repeated all over again. Consequently, some errors would wait for weeks before being solved or rejected.
3. All of the agreements and contacts with PO and stakeholders were happening in a chaotic way. From the point of view of each user group their bugs or tasks had the top priority. Undoubtedly, it resulted in misunderstandings and also caused additional arduous communication by email.

Step 2: Identify workstreams. The next step was to identify the workstream and WIP arrangement. For instance, persons working on subjects connected with support received their own boards or were able to use the common one, but their tasks were assigned with different colours. Similarly, persons working on errors (FT) would own their own boards. Very quickly we noticed another problem emerge, this one connected with the project characteristics. Some of the bugs had to be fixed using a different budget, which meant for example that only 1 FTE[3] could be assigned. Another problem was the fact that a number of all errors were marked as a new feature (CR) and from the project's standpoint, they were then investigated within the in the scope of a different budget and with the use of different resources. The constraints mentioned above required the introduction of additional Kanban boards. The question arose then concerning the person who would make a choice between workstreams. Of course, we added additional boards, where experienced developers or solution architects investigated the issue and decided where it belonged.

Because of the large amount of boards, this approach might seem very complicated. However, from the FT point of view, the 'To Do lists' have been shortened and the focus was only on bug fixing.

Step 3: Improve the communication. The introduced division between work streams was optimal from the FT and ST point of view. On the other hand, from the management's and the client's perspectives (PO-Team), the existing work streams did not always meet their needs.

[3] FTE - Full-time equivalent is a unit that indicates the workload of an employed person.

Because of this, in order to improve communication and effective conduct of prioritization meetings, we have defined a number of options which grouped the chosen work streams, but still retained simplicity. For instance on the Figure 3 we identified *Change Control Board* formed from *Support Board* and *Change Request Board*.

Unified bug list wasn't the right solution. Very quickly we have found the following drawbacks:

5.1.6. Results

After nearly a year since the introduction of the above process, we have conducted an interview similar to the one suggested by Ikonen et. all [5]. In relation to the high complexity of the project structure and different expectations, we have tried to get opinions from each of the teams. Due to the fact that the main work stream is done in sprints, we have focused only on selected work aspects.

Documentation. Dispersed exchange of information by email and arrangements during several meetings have been replaced by cohesive comments within the scope of a given task. They have allowed us to understand each given problem and the process in which a decision was made. To a developer, it has become clear what and why needs to be done. Member of the ST additionally states:

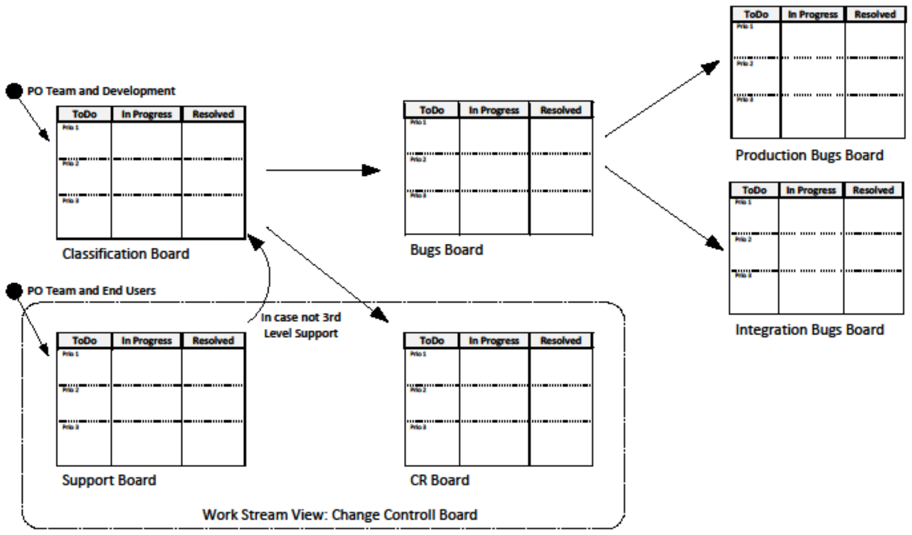


Figure 3. Project A - Kanban Boards

> ”Documentation has been improved, there is no worry about losing parts of data. Once something has appeared on a Kanban board, it will not be forgotten or omitted and it will be equipped with the correct commentary serving afterwards as a source of knowledge in similar problems. ”[ST]

Problem solving. The introduction of Kanban has allowed for easy task assignment to suitable people in the correct order. In case of a developer or customer finding a bug or requesting a new feature, he or she can easily issue a new ticket without the need for consultation with the scrum work model and budget, which made it possible to continue work without delays:

> ”It facilitates and provides a structure for works on error correction.”[FT]
>
> ”Developers are not blocked and know that the reported problem will be proper classified and solved. They are not blocked by unplanned tasks and could develop new user stories without changing the context. ”[CT]

Even though most of the team thinks that certain progress has been made, ST and CT still see some room for improvement:

> ”Using Kanban has not solved all of our problems. Too much of a mess occasionally still happens.”[ST]

In addition CT has pointed out that we have still problems due to a largely rotating team:

> ”On the other hand we have to improve process of bug fixing itself. A task once started does not always get completed before it is time for a developer to leave the FT team”.

Visualization. The process of error fixing and CR management is much simpler and more transparent from the team’s point of view. Bug statuses and workloads are always visible. Each member can easily select a given board and then the related task:

> ”Kanban boards look much better and provide more informations than a long bug list.” [Scrum Master]

The Team and stakeholders understand how the support and bug fixing process looks like:

> ”Once upon a time, I found it difficult to tell the way in which we worked. A project seems to be much more mature, once it has become clear as to how the given processes function.”[ST]

Visualization helps also people working on regular sprint tasks, they claim that:

> ”It is easy to observe whether a person is working to solve a problem, which has blocked us and what is its priority.” [DT]

Communication. Internal communication between teams and the PO has improved dramatically. Instead of having dedicated meetings to discuss each critical error, regular meetings for the selected work streams have been initiated:

> ”The number of meetings has grown, however, they have become shorter and allowed us to effectively manage the tasks on given boards.”[CT]

From the team’s point of view, information concerning prioritizing error-correction and the details concerning them have been set in place. On the other hand, from PO’s point of view, communication with the development team has been improved:

> ”The Client knows who to speak to in regard to a given task and when to expect the solution of a said error.”[FT]

Another equally important aspect is the option of regular progress monitoring.

> ”We can see immediately what is the status of the work. I don’t have to interrupt people and ask what they are doing.” [Scrum Master]

Approval Process. The basic advantage of the defined process was the improvement of work stream choices for new tasks:

> ”... it easier to verify incoming bugs and assign them to proper work stream.” [Solution Architect]

Additionally, the introduction of rules concerning the flow and identification of sponsors responsible for error-correcting has improved support work:

> ”In case if the task cannot be easily solved, because we have to deal with a non- trivial bug or simply with a new feature, we can easily move it to another board.” [Support Team]

Despite the above improvements, the team has still seen the need for improving the process:

> ”Unfortunately, as of now, we have not been able to establish a correctly-functioning approval process. We need another state (column) - Verified. Fortunately we have a proper release process, we can see on the Kanban board what got released and what didn’t.” [Scrum Master]

Selecting Work Assignment. Through a close interaction with the Customer and PO, we have been able to set our priorities right, which in turn has allowed for optimal task realization by the team:

> ”Simply, you take the first task from the first column. You don’t have to search tasks or ask others.”[Scrum Master]

Taking into account the aspect of a budget for particular error types, the choice of a task is clearly sensibly from the Team member’s point of view:

> ”Different boards helps us in finding bugs from different work streams.” [FT]

> ”Tasks are splitted into work streams and could be easily selected based on priority order” [DT]

5.2. Project B

Project involves the manufacturing of production software in a large automotive concern. A part of software supports the direct steering of car production in three separate stages: body construction, paint shop and assembly. The steering systems are critical, because each potential software error generates relatively expensive problems. The said software is employed in several dozen factories belonging to the aforementioned automotive concern. The goal of this project is the delivery of services within a specified time frame and with specified availability, namely the development of new functions and the support of current and existing functions in the production environment.The support is limited to the most difficult problems requiring changes in software or specific changes in the system configuration.

5.2.1. Composition of Team

Due to the massive system function complexity, the team has also become extended. Taking into account various functions and tasks, the project has been divided into the following teams (see Figure 4):

- *Feature Team*(x4) - these are people directly responsible for software manufacturing. The team consists mainly of Programmers and Testers. Each Team is responsible for specific business components.
- *Crossfunctional Team* - responsible for the infrastructure and continuity of project functioning in relation to technical data, namely integrating both Client's and contractor's networks as well as supporting the build and configuration of management processes.
- *Governance Team* - responsible for the management and client co-operation realizes key decisions concerning the project. It is involved in all the existing aspects of project management including change and risk management

5.2.2. Kanban introduction stages

The deployment of an agile approach is much more difficult within the realms of a large organization and extensive stakeholder structure. The above project description does not focus on organizational or business limitations, instead focusing on the employment of the Kanban techniques. Because of critical and limited functionality, all of the process changes have had to have been introduced carefully, that is with risk management, which is an indispensable element of the empirical project approach.

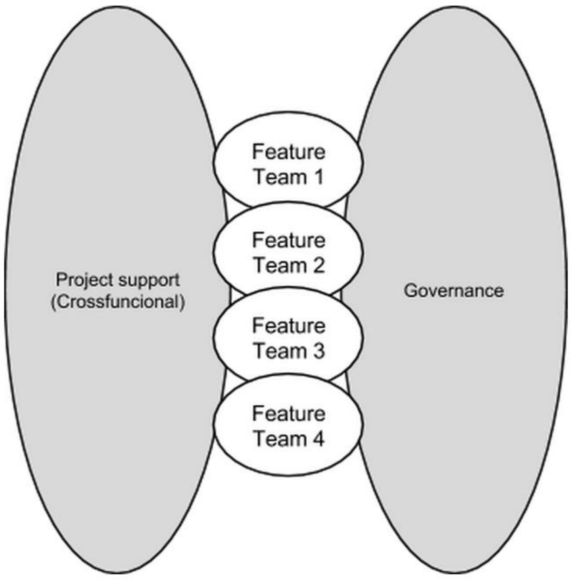


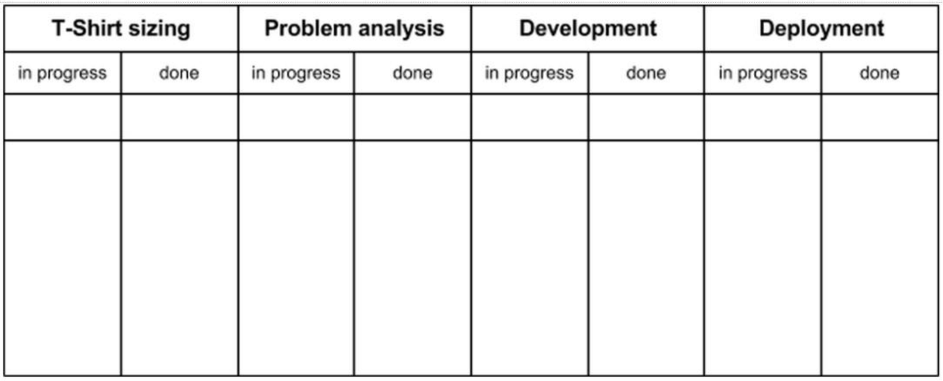
Figure 4. Project B – Team Structure

Step 1: Establishment of the common workflow. In the initial stages, the arrangement caused each of the Teams to function according to their own rules and using their own individual work flow. The following caused difficulties pertaining to the correct definition of general state of work, i.e. defining a completed task (Definition of Done), reporting on critical productive errors (escalations) and seeing the fully complete picture of work in the entire project. Having standardized the work flow, it became do-able to create a root for visualising the Kanban board. In its initial stages it comprises of everyday work (daily business), meaning the current tasks. It consists of the following stages:

- T-Shirt sizing: an initial assessment of a task, which is a relative description of size that is an outcome of complexity, uncertainty and repeatability In this stage, the estimates are not precise and the analysis itself should not exceed 4 hours. Possible values are: S - small, M - Medium, L - large and XL - extra-large.
- Problem analysis: in this stage a detailed analysis occurs based on an earlier estimate. The purpose of this stage is the definition of scope of work and its costs.
- Development: in this stage realization occurs on the basis of an earlier analysis. The purpose of this stage is the engineering of a registrable change in software.
- Deployment: the final stage is the employment of software change and in the majority of cases this is the most complex process. The goal is the delivery of change within the production environment.

It is possible that a problem gets solved on each of these levels, which then completes the process.

Step 2: Visualization processes. Visualization is the best way to achieve common understanding the state of the project, the best way to keep shared vision. We can find the bottlenecks only when everything is measured and visualized to the whole team. The reality of communication is that every stakeholder can have different interests. In this phase of introducing Kanban, it became crucial to start collaborating in the same ”language”. A Kanban board has been created on the basis of an earlier study of the said workflow. (see Figure 5).



T-Shirt sizing		Problem analysis		Development		Deployment	
in progress	done	in progress	done	in progress	done	in progress	done

Figure 5. Project B – Team Structure

Visualization is not only communication improvement, it is also a key element to achieve shared vision and promote it around the project. After the introduction of the visualizations following behaviors were noticed in the teams:

- the processes were described and changes were continuously supplemented,
- the board was continuously adapted,
- the processes were always visible to all members of the the team and they were proposing improvements (feedback loop).

Step 3: Introducing the culture of self-improvement. Project approaches based on nimble philosophy are extremely difficult to implement for multiple reasons, among many of which are the requirements for experience and courage. A given situation can be much simpler if there exists an environment open to the Agile and Lean thinking. It is fair to say that their deployment is not possible without the culture of change and constant improvement in place.

In the process of change within a large organization, one must not forget about socio-logical processes. One of such phenomenon is the Adoption Curve (see Figure 6) [17]. It is precisely this model that became used in the process of employing change to the project and its close environment. The specific technique used was, among others, the selection of the ’so-called’ Change Ambassadors (Early adopters), who were recruited from the management of selected feature teams. It was this group, who was the main communicative target in relation to Kanban deployment techniques. In the aforemen-tioned ’Early Majority’ means a Team.

Instilling the Lean culture allows the use of techniques such as Kanban. Simultaneously, an organisation promotes an adaptive approach on a wider scale, moving far beyond the scope of this project.

Step 4: Managing improvement from the team. The Coach is a very important role in this operation and his position is not to be underestimated. However, his role is to guide the Team towards learning the process of coming to correct conclusions. Just as a parent bringing up a child teaches it to walk and then allows it to reach full independence, so does the Team Coach by pointing out specific problems and then teaching the Team members a lesson on independence.

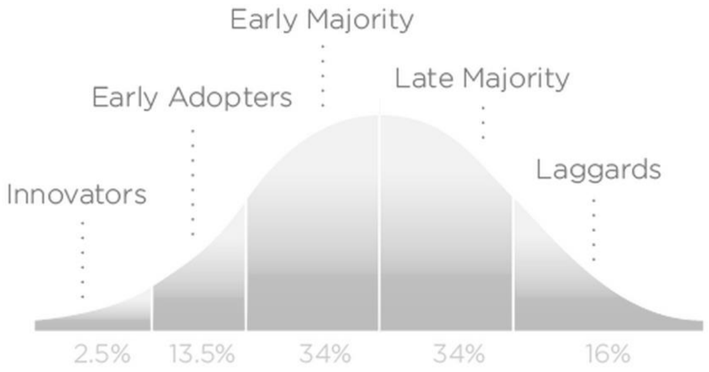


Figure 6. Innovation Adoption Lifecycle

The first problem, which has been noticed thanks to visualisation and the common workflow, was the lack of comprehensible understanding for the Definition of Done (DoD). In the beginning, each Team has defined their DoD in their own way. Unsurprisingly, it has invariably led to serious misunderstandings during the realization of said agreement, especially towards the final stages of the project.

Second of all, certain knowledge limitations have became apparent within the Team. The Kanban board has made it immediately and painfully aware as to which module lacked the necessary knowledge, where fewer tasks existed and where there was potential for certain key moves. Through the act of standardizing the workflow and programming it correctly in the JIRA, we have succeeded in improving both the executive documentation procedure as well the communication regarding production difficulties:

- documentation concerning current problems consists of the necessary meta information i.e. contact persons, references to other existing documents (i.e. change request)
- summary of existing problems is being documented in a uniform manner.

Step 5: Introducing the processes to the Customer. Now as a result of the other changes, we have become able to work with the process improvements. But how can we work on improvements together with the Customer? The first step was to visualize a set of data. A special Kanban board with set of data has been prepared. There are special instances of Kanban boards for a specific customer. Special cases for each group of stakeholders:

- IT Department: the board shows all scope and parameters. With the complete overview in front of him, the customer can easily decide and prioritize work.
- Problem management: this group on customer side is responsible for the internal customer processes, e.g. ITIL [18]. With this special case of the Kanban board, they can proceed relatively fast aligning our work to their processes.
- Local IT Departments within the factories. They have used our software directly and plan on its upgrading in the future. They have used special boards including details limited to their scope of interest (only pertaining to issues from one particular factory).

5.2.3. Results

One of the most significant consequences of Kanban introduction is the ability to measure process, as an example quoting such defined metrics as:

- an increase in the number of called-in tickets relative to the closed cases (see Figure 7),
- a possibility to measure the average time for closure and costs of fabrication (Lead Time i Cycle Time),
- cyclical report for shifts in the original estimate, which meant a comparison of adequate work estimation in the T-Shirt sizing phase to the actual volume of

work being done. The report has made it possible for an early detection of the most incorrect estimates and their causes

- the quality of task documentation coming from the Client is also being measured, which in turn has allowed for an introduction of multiple improvements. The end effect has been an improved communication with respective Client departments.

Additionally, SLA values (Service Level Agreement) are being measured within the scope of the said project based on the previously agreed parametres. The achieved SLA may shift up to a certain extent. The following SLA indicators exist in the project:

- Reaction time meaning from the moment a work unit was created until the actual work has begun (Early analysis),
- Time of the initial analysis (T-Shirt sizing).

In the analysis phase (Problem analysis, Development i Deployment), a high level of vagueness has made it impossible to introduce SLA that would measure up an end of work. An exemplary cumulative chart (see Figure 8) has highlighted the piling up of tickets in the analysis phase for the final period between September to November. Such a visualization has very much given credibility to reports for the Client.

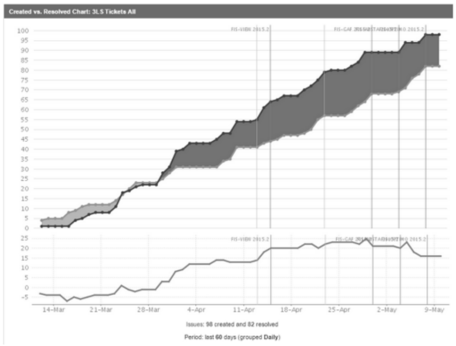


Figure 7. Created vs. Resolved Chart

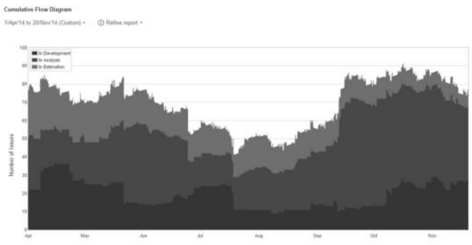


Figure 8. Cumulative Flow Diagram

6. Summary and Key Observations

In a progressively larger number of IT projects, one can easily notice a trend towards process and actions optimization within the scope of executed projects. Both the Development Teams and Clients have spotted flaws in processes imposed by diverse methodologies. The said ones are in many cases over the top, worthless and unnecessary and their existence is only justifiable solely on the merit of each selected approach. Moreover, frequently chosen software development methodologies do not encompass certain much needed processes.

Although the Kanban technique is not a subject of frequent analysis and has not been as promoted as the Scrum or XP ones, it has become more and more usable in IT projects as one of the tools of ”Lean thinking”. It can be used with positive results in each project type regardless of whether it might be a ”Agile” or ”Waterfall” style operation. It is worthy of a notice that this simultaneously basic and at the same time intuitive mechanism is a powerful tool allowing for an easy optimization of nearly every activity and process within the software projects. In both cases, we have seen noticeable profits both on the side of our Team as well as on the Client’s side over a relatively short period of time.

The most important aspects of the aforementioned are undoubtedly visualisations, regular process order and the creation of a cooperative platform, which can be easily modified and adapted to any given target group.

Whilst analyzing the consequences of Kanban deployment another perspective has emerged. Taking into account the human factor, It is noticeable that Kanban’s use triggers a gradual self-improvement from within each Team, a sort of evolutionary step towards the betterment of documentation and fabricating processes. Hence, unlike something forced upon us by the management or outside specialists, the Kanban results in an all-natural, symbiotic and adaptive process.

7. About Capgemini and Nearshore Center Wrocław

With more than 130,000 people in 44 countries, Capgemini is one of the world’s foremost providers of consulting, technology and outsourcing services. The Group reported 2012 global revenues of EUR 10.3 billion. Together with its clients, Capgemini creates and delivers business and technology solutions that fit their needs and drive the results they want. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience and draws on Rightshore® , its worldwide delivery model.

Nearshore Center of Capgemini exists in Wroclaw since 2004, and is thus part of the worldwide Righshore® of delivery centers. More than 650 IT experts currently work in Wroclaw delivering high quality services in the areas of software development, software package implementation and application life cycle services to German-speaking clients.

Capgemini in Poland employs more than 5000 consultants and IT as well as business process experts. Centers for IT and business process outsourcing services exist in Wroclaw, Krakow, Katowice and Opole with the main office serving the Polish market based in Warszawa.

8. References

[1] David J. Anderson. Kanban: Successful Evolutionary Change for Your Technology Business. Blue Hole Press, April 2010.
[2] Atlassian. JIRA Documentation, 2014.
[3] Atlassian. Specification - Confluence Advanced Editor, 2014.
[4] Kent Beck. Embracing Change with eXtreme Programming. Computer, 32(10):70–77, 1999.

[5] Marko Ikonen, Elena Pirinen, Fabian Fagerholm, Petri Kettunen, and Pekka Abrahamsson. On the Impact of Kanban on Software ProjectWork: An Empirical Case Study Investigation. In Engineering of Complex Computer Systems (ICECCS), 2011 16th IEEE International Conference on, pages 305–314. IEEE, 2011.
[6] Daniel T. Jones James P.Womack. From lean production to the lean enterprise. Harvard Business Review, apr 1994.
[7] Henrik Kniberg. Kanban and Scrum - Making the Most of Both. Lulu.com, 2010.
[8] Henrik Kniberg. Lean from the Trenches: Managing Large-Scale Projects with Kanban. Pragmatic Bookshelf, 2011.
[9] Julia Koplin, Stefan Seuring, and Michael Mesterharm. Incorporating sustainability into supply management in the automotive industry - the case of the Volkswagen AG. Journal of Cleaner Production, 15(11):1053–1062, 2007.
[10] Marek Majchrzak, Lukasz Stilger, and Marek Matczak. Working with Agile in a Distributed Environment. In Practice Software Engineering from Research and Practice Perspective (Eds. Lech Madeyski, M. Ochodek), Scientific Papers of the Polish Information Processing Society Scientific Council:41–54, 2014.
[11] Mattias Jansson Michael Prokop. Use of kanban in the operations team at spotify. InfoQ, sep 2010.
[12] Peter Middleton, Amy Flaxel, and Ammon Cookson. Lean Software Management Case study: Timberline inc. In Extreme Programming and Agile Processes in Software Engineering, pages 1–9. Springer Berlin Heidelberg, 2005.
[13] Peter Middleton and David Joyce. Lean Software Management: BBC Worldwide Case Study. Engineering Management, IEEE Transactions on, 59(1):20–32, 2012.
[14] Taiichi Ohno. Toyota Production System: Beyond Large-Scale Production. Productivity, Cambridge, MA, 1988.
[15] Kai Petersen and Claes Wohlin. Measuring the Flow in Lean Software Development. Software: Practice and Experience, 41(9):975–996, 2011.
[16] Mary Poppendieck and Tom Poppendieck. Lean Software Development: An Agile Toolkit. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
[17] Everett M Rogers. Diffusion of Innovations, 5th Edition. Simon and Schuster, 2003.
[18] Arraj Valerie and Compliance Process Partners LLC. ITIL®: The Basics. OGC. Whitepaper, jul 2013.

Chapter 2

Adapting Scrum Method to Academic Education Settings

1. Introduction

In the field of software engineering, every project should be supported by a particular development method in order to be successful. In consequence in the past years this field has not been shy of introducing different methodologies, more and more distinguishable from the traditional approach presented by the Waterfall model or Spiral model. To address their main drawback of being heavily regulated and often not incremental, new means, called agile emerged. Agile project management is a style of project management whose focus is placed on iterative and incremental development and early delivery of results. It emerged in the mid-1990s and caught the attention of project managers as well as customers and became widely acclaimed and employed in different branches of the industry. While similar problems are usually encountered during implementation of students' projects, little attention was paid to applying software development methods in academic setting thus far.

Agile approaches emerged as a response to the need of new approach of project management and it is beneficial to familiarize one's self with its history and aim. Waterfall and other traditional methods which put emphasis on up-front requirements capture and design are not likely to be successfully applied by student, who usually do not know well the environment in which they have to work, they have little knowledge of the imposed problem and are supposed to learn during the whole process of project development. Moreover, their initial plans and assumptions look feasible on paper, however turn out to be troublesome in practice, very often causing technological difficulties and require a re-design and choosing a different direction to solve the problem. Intuitive resolution of these issues would mean applying a method from agile family, which deals with aforementioned obstacles by providing maximum flexibility and appropriate control.

The presented paper focuses on the Scrum method, one of the most widely used agile process frameworks for development of software [13]. This method was adapted for certain requirements to be applied in the academic setting. Moreover, a complementary tool that students could use to employ Scrum practices in the creation of their university courses projects was also developed. The paper is structured as follows: section 2 gives the outline of agile software development and Scrum framework, while section 3 deals with the adaptation of Scrum method the academic needs and requirements, presenting how to modify Scrum's core to make it suitable for students' projects and research work.

2. Agile Software Development and Scrum

Using outdated management and development approaches can cause problems, project failures and in consequence financial losses. Fortunately, the growing problems have been recognized over the past decades and new, so-called lightweight development methods emerged in the mid 1990s. Their main goal was to oppose heavyweight methods like Waterfall, which were regarded as heavily regulated, regimented and overly incremental.

The values that agile process proponents adhere to are: firstly, the *importance of the relationship, spirit of cooperation and belonging of software developers* [1]. Human role in the process of development is far more significant and this of tools and institutionalized processes. What it means in practice is that people can respond quicker and transfer

ideas more efficiently with direct interactions rather than through reading and writing documentation. Secondly, the crucial point of agile development is to continuously provide *tested and working software or its modules*. Releases are carried out frequently, varying from even once a day to once a month. Agile methods embrace the idea of simplicity, where no more code is produced than necessary and no documents attempting to predict the future are created, as they inevitably become out-of-date. Third, *participation of the client in the project and close cooperation with the development team is* critical and more important than strict terms of a contract. A client whose needs are well understood and addressed is a satisfied client and in the end the expected business value is delivered along with a fulfilled contract. Fourth, based on the assumption of *project transparency*, where both software developers and customer representatives are well informed and have necessary knowledge and skills to make changes to the predicted plan as the development of the project progresses.

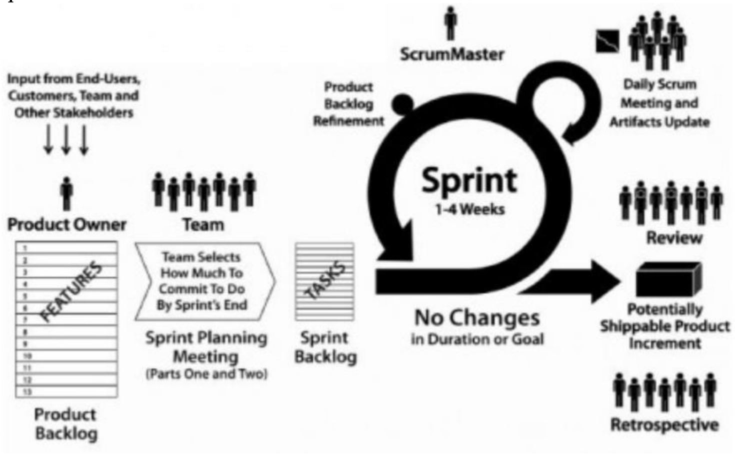
Scrum is one of the members of Agile family, consequently it is iterative and incremental, with work being structured in entities called Sprints [3,13]. They are commonly a couple of weeks long, not exceeding a month's time. Moreover they are of fixed duration, finishing at a specific date, even if the planned objectives have not been met. Each spring begins with a spring planning meeting, where members of a cross-functional team select customer requirements from a prioritized list and grant to accomplish them by the end of the sprint. As a part of the development process, every day the team carries out a daily Scrum meeting to briefly discuss their progress and re-plan their work (Fig. 1).

3. Scrum in Academic Environment

A handful of IT corporations has had success with Scrum method, however not much attention has been paid to applying it in academic setting. While characteristics of everyday work differ significantly in commercial industry and research fields, adopting Scrum's traditional proceedings may yield promising results as the point was proven by numerous universities.

3.1. Motivations for Using Scrum in Academia

Owing to its overwhelming popularity in management of IT projects in recent years, Scrum is familiar to many teachers in academia and to most of Software Engineering course participants as it is introduced to the students as an example of a prominent agile methodology. But there can be done much more than just presenting a theoretical background, which will fade away from undergraduates' memories as soon as they finish the course in question.



ScrumMaster
Input from End-Users, Customers, Team and Other Stakeholders
Product Backlog Refinement
Daily Scrum Meeting and Artifacts Update
Sprint
1-4 Weeks
Product Owner
Team
Review
FEATURES
Team Selects How Much To Commit To Do By Sprint's End
TASKS
No Changes
in Duration or Goal
Potentially Shippable Product Increment
Sprint Planning Meeting
(Parts One and Two)
Sprint Backlog
Product Backlog
Retrospective

Figure 1. Model Scrum process [12]

Fruitful results can be seen in study on University of California where Scrum was applied as a part of the practical assignment in Software Engineering course [5]; University of Duisburg-Essen used Scrum to guide Academic Game Development course [6] while staff from University of Maryland managed a group of researchers by the means of slightly modified version of Scrum [7]. These and other examples motivate to provide an adjusted set of Scum principles and practices to make it applicable to students' curriculum projects by first learning and getting practical experience within a Software Engineering (SE) course. Unfortunately, typical SE course is highly theoretical and its objective is to teach software development process, including all phases involved – requirements gathering, design, implementation, testing and quality assurance. In most cases however no chance to practice creation of software according to good practices is given to the students.

Limited time of every course as well as resources of each participant create the need for adaptive approach that enforces teamwork and at the same time is a great opportunity for students to learn. Hence Scrum, a flexible and light project management method is advised to carry out the practical part of a course, since project development is the best way of knowledge consolidation. Graduate with hands-on experience with project management methodologies is more attractive to potential employers, who apply modern frameworks to manage the software projects as a way of reducing costs by optimizing employees' time.

Other challenges that students face during the development of their projects is lack of work organization as they are not prepared to manage and be managed [8,9]. There is much room for improvement in the team working department and fair task division abilities within the project group. Yet another issue to be addressed is that scope of project requirements is often vague, as the instructors tend to only give in overall idea of what the expected output is, but do not provide the necessary details. Figure 2 presents the major problems that cause a ”bottle neck” in students productivity.

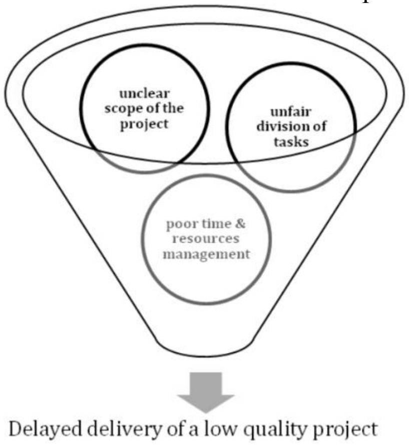

unclear scope of the project
unfair division of tasks
poor time & resources management
Delayed delivery of a low quality project

Figure 2. Major reasons for students’ projects failures that are to be addressed with introduction of ”academic” Scrum

Proposed means of adapting Scrum to academic setting will try to resolve the aforementioned problems and furthermore to have positive influence on the aspects of [6]:

- Teamwork, which it is vital in any project management as the development is carried out by multiple members and is not assigned to a particular person. This includes aspects of communication, reliance, and conflict management, thus evolving important soft skills.
- Flexibility, as schedules of students and their availability to work on a particular subject change constantly. It also concerns the issue to react on changing requirements in awareness of the general objectives.
- Empowerment, which makes individuals more confident regarding their capabilities. Instead of a central decision-making, everyone’s involvement and competence has impact on decisions made, which results in higher motivation and commitment.
- Productivity, which is crucial in respect to both the limited time frame as well as to the comprehension of tasks to be handled within the development process.

3.2. Scrum Roles

There are three primary roles to be divided among the course participants: the role of Product Owner, Scrum Master and the Development Team. This is the aspect of Scrum that varies significantly between the industry and academic setting. Firstly, there are no external stakeholders or specialists in development/management/marketing domain. In essence, the course professor/teacher masters the subject he conducts. On top of that, a person with a Scrum certification that normally takes the role of a Scrum master can only be dreamed of in most of cases. On the other hand, there are more than enough resources to form a development team whose typical size is 6 to 9 persons; in fact a couple of separate teams probably will be assembled depending on the amount of course participants.

The proposed solution, which takes into account the limited resources, is for the course monitor to play the Product Owner, one of the students to be a Scrum Master and the Development Team to incorporate other students.

There have been deviations in role dispersion in real life studies of applying Scrum in academic setting, e.g. the student would take on the role of Product Owner [5] or the professor would play a Scrum Master [9]. The suggested division of roles was proved to be successful in numerous cases [6,8,11] and seems to be a reasonable way to manage what a course professor has at hand. While a Scrum Master should be savvy about the technology and preferably possess a Scrum Master certification, evidently that is not the case in a university class. As scrum master's main occupation is enforcing Scrum's rules and making sure the team understand them and follows them carefully, it is proposed that a student preferably with a prior experience in software development according to Scrum practices would take up the role. This is more likely to happen in a graduate course, when students have already completed internships or other form of professional activity. Nevertheless, in the opposite case, a person that proved to be responsible, collaborative and committed ought to be chosen, as these are traits of character that a good Scrum Master possesses.

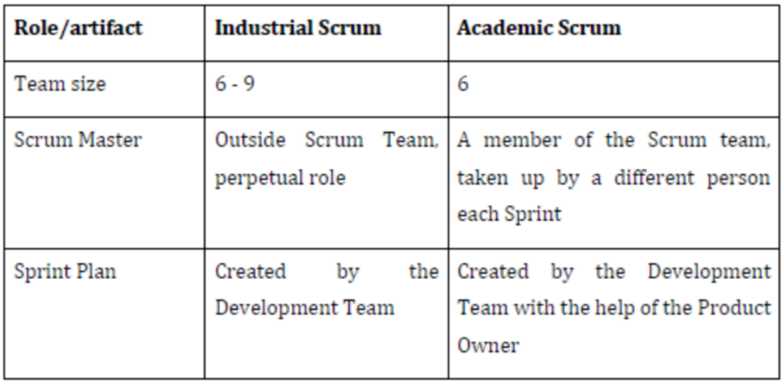
J. Schild and R.Walter [6] suggest that a new Scrum Master should be chosen for the duration of each Sprint. It is a good chance for a bigger number of students to fully grasp the objectives of this role and learn how to put the rules and practices into action. This solution however, comes with certain risk as not all of students have natural capabilities to take on that role. Finally, it is necessary to mention that a Scrum Master is still expected to work with other students on the development of the project, unless there are numerous team members and insufficient workload.

Lastly but not least importantly comes the role of a *development team member*, which naturally will be taken up by the students. The principle of industrial version of Scrum advocates the size of a team to fit between three to eight persons. While this rule might be preserved in the academic setting, a lower bound of this range is suggested. The development team of 6 people was chosen in most of the successful cases of Scrum usage in academic courses [5,6,8], hence the proposed solution retains this exact size of a team. It should be noted though, that the course teacher might concede possibility of adjusting the scope of the project and the required workload for its successful completion, so that it reflects the team's capabilities.

3.2.1. Scope of Responsibilities

It is not just the dispersion of roles that differs in both industrial and academic variants of Scrum. As each role comes along with a set of responsibilities and activities to perform, further adjustments need to be made.

Whereas in the existing cases where Scrum was used for management of the students' projects [5,6,8], it was the students that came up with ideas for the project and were expected to have a clear vision of the end result, hence create a Product Backlog; in the proposed solution a different approach is taken. Following the classical approach of Scrum, it is the Product Owner that takes the full responsibility of the Product Backlog, in consideration of his knowledge of the teaching subject and years of experience. It is highly doubtful that students are adept to come up with a solution that would fit a tight planning schedule and whose scope would be just right for the team to work on. Consequently, being able to identify all of the functionalities that the final outcome should incorporate it is the instructor who ought to present a pre-prepared list of prioritized features that students will work on.



Role/artifact	**Industrial Scrum**	**Academic Scrum**
Team size	6 - 9	6
Scrum Master	Outside Scrum Team, perpetual role	A member of the Scrum team, taken up by a different person each Sprint
Sprint Plan	Created by the Development Team	Created by the Development Team with the help of the Product Owner

Figure 3. Industrial Scrum modifications that were made so that the method can be applied in the academic setting

The course instructor's role as a Product Owner is broader and comes with a bigger amount of undertakings – for example as observed in [6], his help during a Sprint Planning Meeting is very beneficial to the development team that has no prior experience in estimation of tasks nor the knowledge of the subject matter. Students are prone to encounter problems in differentiating between features from a Product Backlog and more specific and detailed activities that are defined within a scope of a Sprint. Just as in the case of industrial Scrum and professionals in the domain of IT, students tend to underestimate tasks, the workload and time needed to complete them and this is where the course instructor should give a helpful hand and guidance. Figure 3 sums up the modifications made to the Scrum roles and activities that come along with them.

3.3. Course Structure

The day-to-day work varies significantly in the academic setting and commercial companies, employees follow a different schedule than students normally do, projects last much longer and involve more people than laboratory class undertakings. In the professional world, people can be completely committed to their work and it should takes place according to a fixed schedule. Students, to the contrary are expected to work on multiple projects in parallel, have different responsibilities, and the only thing they have in common is the course they attend together. Hence, it is the course structure that influences the organization of a project and the way it will be managed.

The Lodz University of Technology will serve as the main example of the curriculum projects carried out. In most cases, classes would take 15 weeks, with a workload of 4 hours per week, divided into two hours of lectures and two hours of laboratory/practical classes. While the proposed solution is expected to yield the best results for such a configuration, classes that take more hours seem to be suitable to apply the modified version of Scrum as opposed to shorter courses. Additional effort that following a structured methodology requires on top of the workload necessary to develop a project would seem a burden and would not be profitable to the students.

Apart from the hours spent at the University, students are expected to spend approximately the same amount of time working on the subject after classes. Most of the Scrum activities will take place in the class, thus shifting the bigger part of the development process to after-school hours. While this might seem like extra effort and more time spent by the students on one subject in question, it is expected to be advantageous, as by good organization of work, clear and fair division of tasks – in the end students will perform more efficiently and gain time in the global perspective of the development.

The industrial version of the method dictates the Sprint to take between 1 and 4 weeks, with a preference toward shorter intervals. Therefore, taking into consideration that the course lasts for 15 weeks, a Sprint is scheduled to take from 2 to 4 weeks depending on the type of course being conducted and planned workload. This was the most common solution while applying Scrum in the academic setting [5,6,8] and proved to be fruitful in the examined projects. In the case of a one week Sprint, there is not enough time to conduct all of the necessary activities (planning, demonstration of the product, retrospective) and work on the actual implementation at the same time. What's more, within the duration of a one-week Sprint students would be very unlikely to finish a demonstrable version of the application as they take part in multiple courses as a part of their curriculum and might experience scheduling and work organization problems.

The practices that are strictly connected with a Sprint – *Sprint Planning Meeting*, *Sprint Review* and *Sprint Retrospective*, as usual take place before or straight after the Sprint, so their frequency stays the same as in the typical Scrum applications.

To the contrary, a Daily Scrum evidently cannot still be carried out every day, as classes take place once or twice a week, with the latter being a more common case. Thereupon, organizing two ”daily Scrums” in a week is a goal set in the proposed solution, and should be put into action during the class time. In the case of once a week courses, the professor should impose another Daily Scrum outside of the study room, which might tend to be problematic so different approaches of organization should be taken into consideration. The possible channels of communication of a distributed Daily Scrum and their corresponding effectiveness are: e-mail, teleconferences, video conferences, direct meetings.

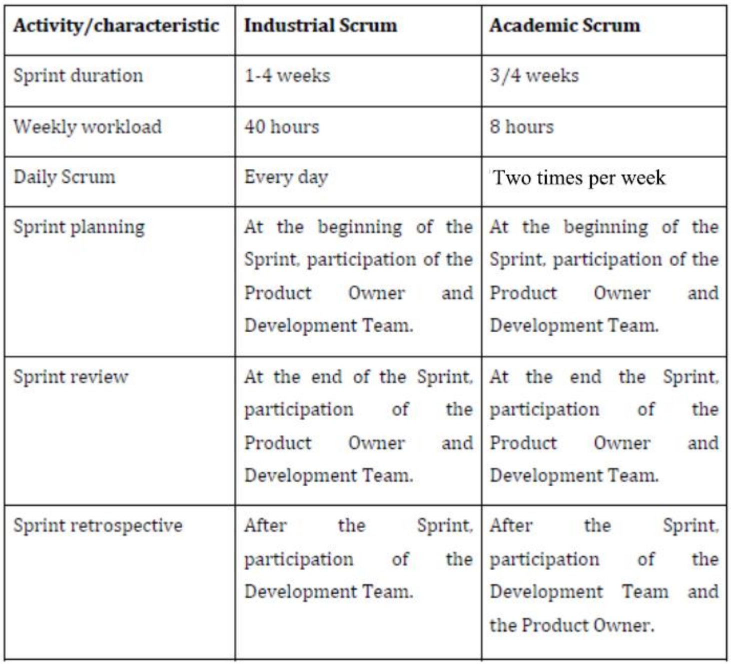
As one knows from the daily practice, the e-mail or communication through documentation approach are the least favorable, as there is no real exchange and interaction between the team members. Effectiveness of the meeting increases as the channel becomes more and more direct and resembles typical daily scrum.

As it is a short meeting, with duration of up to 10 minutes, students can organize one in between classes on some other day of the week. A report should be required from students after such a meeting in order to reinforce them to put the Scrum practices in action and motivate them to regularly work on the assignment. A voice recording from the daily Scrum would be a perfect solution, since it doesn't require any extra effort in the world of today's modern technology where almost every student has a smartphone and forces the meeting to take place at the same time involving all the team members.

Figure 4 sums up the differences and similarities concerning schedule, frequency of activities and workload predicted between the industrial version of Scrum and his academic equivalent.

3.4. Course Content and Schedule

Composition of course content in order to maximize student's profit, balance theoretical knowledge and hands-on experience is always a big challenge for a professor conducting a class. Introducing a project management methodology on top of that conveys the impression to make it even more ambitious and effortful, however importing Scrum into the classroom is supposed to make life easier for the both parties.



Activity/characteristic	**Industrial Scrum**	**Academic Scrum**
Sprint duration	1-4 weeks	3/4 weeks
Weekly workload	40 hours	8 hours
Daily Scrum	Every day	Two times per week
Sprint planning	At the beginning of the Sprint, participation of the Product Owner and Development Team.	At the beginning of the Sprint, participation of the Product Owner and Development Team.
Sprint review	At the end of the Sprint, participation of the Product Owner and Development Team.	At the end the Sprint, participation of the Product Owner and Development Team.
Sprint retrospective	After the Sprint, participation of the Development Team.	After the Sprint, participation of the Development Team and the Product Owner.

Figure 4. Compares the organization of work in both industrial and academic versions of Scrum

3.4.1. Preparation of Project to be Developed and Forming Groups

The first obstacle that the course professor will face is adjusting the scope and subject of the project that will be given to the course participants. It is advised to plan a project that involves implementation of some information system, choice of subject being left to the course monitor. It is highly important for the students to experience the essence of Scrum which is used for the development of IT projects, therefore theoretical assignments, similar to just planning and documenting a fictional project seem unsuitable in the case of Software Engineering course, which by principle deals with knowledge about software development. As stated by Hazzan [10] in the interest of teaching software development approaches and methodologies, it is essential that the students' understanding of them is "based on one's personal experience and reflection on one's creation process". He also adds that "in the context of SE (software engineering), the better one understands the process of developing a software system, the better one may understand the methodologies that guide this process".

The professors from Rochester Institute of Technology, NY [11] whose course was focused on Scrum process practices, came up with a challenging yet manageable project idea. Teams used an open source; Java based development kit to create Android applications. This type of project in most of cases does not implicate prerequisites that undergraduate students are not able to meet and what is more, proved to be a popular choice among them. In the case of curriculum classes with focus on some Information Technology domain, the subject of the practical assignment stays the same as prior to introduction of Scrum. However, some adjustments of scope and/or project milestones might be necessary to fit the proposed schedule.

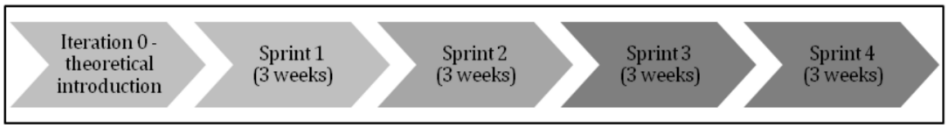


Figure 5. Division of Software Engineering course into sprints

For instance, at Lodz University of Technology multiple courses were organized in a manner where there was a small project milestone outlined every three or four weeks. Applying the "academic" Scrum will require these small undertakings being translated into set of requirements that can be developed within a Sprint and constituting one semester long project. When it comes to groups formation, professors from both University of California [5] and University of Duisburg-Essen [6] have asked the participants to write an application letter presenting their skills, relevant courses they participated in, expectations towards the subject etc. in order to balance the teams. In the case of the proposed solution, it is regarded as rather unnecessary, as the students have rather homogeneous background, thus teams can be formed quickly, still being relatively equally skilled and efficient. An e-mail can be sent prior to the first classes, asking the students to think over the team forming decisions.

3.4.2. Choosing Sprint's Duration

Having 15 weeks of classes to exploit, usually the first one being an organizational meeting, the following structure of Software Engineering course is proposed: 3 weeks of both lectures and laboratory classes being devoted to theoretical knowledge about software creation with special emphasis on Scrum, its rules, practices and activities involved. Subsequently, four blocks of three weeks that make up Sprints, where Scrum related activities are mainly carried out during laboratory classes and topics from the discipline of Software engineering are presented in parallel during lecture time (Fig. 5).

Alternative division of the course time, with three Sprints lasting four weeks each, was discarded. Primary reason for doing so is that the course should offer the students as many opportunities to practice Scrum's activities as possible and get a chance to face problems that the Scrum teams usually do during development and where the Scrum's inspect and adapt approach comes in handy. Hence the sprints were prolonged to 4 weeks so that students work more intensively on the studied subject, have a chance to deliver bigger parts of required functionality (aforementioned case of multiple small projects within one course) and lastly, have evenly distributed workload all along the course which would enable the students to avoid the crunch time just before the examination session.

A closing statement regarding Sprints is that the workload in the final Sprint should be kept minimal in regard to students' commitment to other courses and exam session that occurs in the end of the semester. As focus of the course is put on learning Scrum, the majority of development should take place in the middle of course's duration and the final weeks should be devoted to the evaluation of Scrum and carrying out Scrum-related activities.

3.4.3. Scrum Related Activities Walk-through

The proposed outline of adapting Scrum involves abundance of its core's formal meetings, but presents numerous exercises and activities that are intended to facilitate the learning process as well.

With regard to the last aspect, it is crucial to practice with students the *Sprint Planning Meeting* and activities that are correlated with it, hence there is a Sprint Backlog exercises session foreseen in the prospect. While the course monitor, who plays the role of the Product Owner, provides a full and detailed Product Backlog, it is the Development Team that needs to convert that prioritized list of requirements into set of manageable tasks. This conversion process itself will certainly be problematic to the students; the distinction between fine-grained activities that can be implemented within a Sprint and high-level features from Product Backlog is not apparent to inexperienced Development Team [6]. On that account, multiple general examples of how to write user stories along with concrete example of the assigned project's requirements being translated into user stories need to be given. To address these issues, a *Planning Poker* workshop is arranged in the laboratory schedule. Planning Poker is a widely acclaimed and most commonly applied technique used to estimate effort – relative size of product backlog items.

At the essence of Scrum lies the *Daily Scrum* – a mandatory communication meeting of the entire Development Team which is carried out every day during a Sprint's duration. While apparently due to time and schedule constraints of the course participants it cannot take place on daily basis, form of the meeting being modified as well, the goal of the gathering is preserved. One should remember that it does not serve to report status of development to the project manager (instructor) but to exchange information within the team, gain understanding of what work has been done, which tasks remain to be implemented and which team member commits to take care of them. It usually takes a couple of weeks for the team (depending on its knowledge and experience) to fully grasp the idea of Daily Scrum.

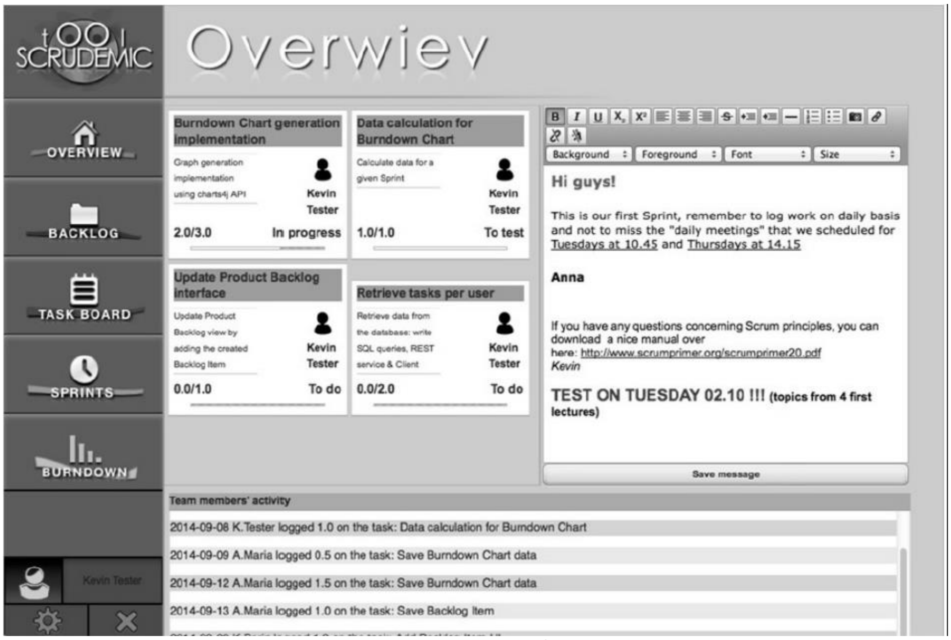
Each Sprint is ensued by a *Scrum Review*, which is present in both types of courses being conducted. It is a great moment for the professor to evaluate what students have achieved in the last Sprint. Furthermore it is a perfect time to officially grade the achievement of four weeks work – a demonstrable demo of the realized functionality.

Sprint Retrospective, for the most part scheduled right after Sprint Review, plays utterly different role in the Scrum methodology. This meeting is a perfect time for the team to reflect on the process they have been applying and ways of improving it. These might include scheduling, communication channels, and division of tasks; in essence any aspect that team finds questionable or burdensome. It is also suggested for the team to use the feedback from burndown charts to tune the task identification and estimation technique for future Sprints.

3.5. Scrudemic Tool – Scrum Supporting Tool

The project management tool was developed and used to support the academic version of Scrum. Owing to the Scrudemic Tool, students can manage the project they develop as a part of laboratory classes according to the "academic" Scrum rules. The application is intended to serve students in the management of the project throughout its entire lifecycle – from the early phases of development (requirements gathering in the Backlog), actual development (Task Board with assignments that are currently in progress) and evaluation of results (Burndown charts that show the overall performance over an interval of time). The functional requirements for the Scrum supporting tool are:

- user can register an account,
- user can be identified and matched with his account,
- user should see activity of his own team,
- user can manage tasks and log his progress,
- user can manage Sprints within his project,
- user can maintain a Backlog for the project,
- user can track progress of work using Burndown charts.



Overwiev

Burndown Chart generation implementation
Graph generation implementation using charts4j API
Kevin Tester
2.0/3.0 In progress

Data calculation for Burndown Chart
Calculate data for a given Sprint
Kevin Tester
1.0/1.0 To test

Update Product Backlog interface
Update Product Backlog view by adding the created Backlog Item
Kevin Tester
0.0/1.0 To do

Retrieve tasks per user
Retrieve data from the database: write SQL queries, REST service & Client
Kevin Tester
0.0/2.0 To do

Hi guys!

This is our first Sprint, remember to log work on daily basis and not to miss the "daily meetings" that we scheduled for Tuesdays at 10.45 and Thursdays at 14.15

Anna

If you have any questions concerning Scrum principles, you can download a nice manual over here: http://www.scrumprimer.org/scrumprimer20.pdf
Kevin

TEST ON TUESDAY 02.10 !!! (topics from 4 first lectures)

Team members' activity
2014-09-08 K.Tester logged 1.0 on the task: Data calculation for Burndown Chart
2014-09-09 A.Maria logged 0.5 on the task: Save Burndown Chart data
2014-09-12 A.Maria logged 1.5 on the task: Save Burndown Chart data
2014-09-13 A.Maria logged 1.0 on the task: Save Backlog Item

Figure 6. Dashboard – main window of Scrudemic application

Once the user successfully logged in, a Dashboard/Overview interface is rendered. The window consists of the menu bar on the left, which allows the user to browse through 5 principle views, namely: Overview, Backlog, Task board, Sprints, Burndown as well as access the settings of the application (left) and log out button (right) (Fig. 6).

4. Conclusions

This paper presented the outline of Scrum project management and its practices, rules and artifacts enabled to see how Scrum works in commercial project development and what benefits it brings. It proposed a course design that would allow importing Scrum into the classroom and developing a student's project in accordance with its practices. Introduction of a project management framework in academia classes gives both challenges and opportunities, yet the potential benefits surpass the downsides. From the course professor perspective it is primarily an improvement in the class organization dimension. Introducing Scrum yields simple yet powerful means to get insight into students' work, track it effectively and evaluate it on regular basis.

As from the student's angle it is a great opportunity to engage in Agile practices within the framework of Scrum that supports the fundamental software engineering skills every IT specialist needs to have. Integrating a project management methodology into daily work of students puts in perspective how effectively crucial aspects of development can be carried out. Requirements management, project planning, tracking and testing become structured tools that students can use to fight time constraints and deliver high quality project on time. Finally owing to Scrum practices students benefit from effective team collaboration and can develop their communication skills.

The presented approach is realized each year at the authors' institution, involving the group of 30-50 students, performing different projects, both for desktop and mobile platforms, creating Internet applications, computer games and management information systems.

5. References

[1] R. C. Martin, Agile Software Development - Principles, Patterns, and Practices, Prentice Hall, 2002
[2] P. Abrahamsson and O. Salo, Agile software development methods - review and analysis, VTT Publications, 2002
[3] K. Schwaber and M. Beedle, Agile Software Development with Scrum, Wiley, 2001
[4] R. Ramsin, Software Development Methodologies, http://sharif.edu/
[5] L. Werner and D. Arcamone and B. Ross, Using Scrum in quarter-length undergraduate software engineering course, University of California, 2012
[6] J. Schild and R.Walter, ABC-Sprints: Adapting Scrum to Academic Game Development Courses, University of Duisburg-Essen, 2010
[7] M. Hicks and J. S. Foster, Adapting Scrum to Managing a Research Group, University of Maryland, 2010
[8] L. Pinto and R. Rosa, On the Use of Scrum for the Management of Practical Projects in Graduate Courses, Frontiers in Education conference, 2010
[9] Using Scrum for Software Engineering Class Projects Ramrao Wagh, 2012
[10] O. Hazzan, The reflective practitioner perspective in software engineering education, Journal of Systems and Software, Vol. 63, Issue 3, pp. 161–171, 2002
[11] T. Reichlmayr,Working Towards the Student Scrum - Developing Agile Android Application, Rochester Institute of Technology, 2011
[12] Agile and Extreme Programming for Product Development and Consulting, http://www.goodworklabs.com/process/
[13] J. Sutherland, Scrum: The Art of Doing Twice the Work in Half the Time, Crown Business, 2014

Chapter 3

Agile Management of Research Projects in the Contract Context

1. Introduction

This study is based on the 30+-year experience gained while managing innovative process control and business management projects [1][2][3][4][5]. For these and similar projects, their scope definition and budget estimation in advance was always the most challenging task. Typically, if the estimated budget of any project is higher than the other ones, the solution provider is recognized as inefficient in one way or another. But there might be another reason if innovative projects are concerned, i.e. the provider's know-how and extraordinary experience make a better assessment possible. Better assessment always means higher budget in this context and, in a typical bid where budget is the most important factor, it puts the solution provider in an underprivileged position and leads to the "more stupid the better" syndrome.

For an innovative project, the main reason why its critical parameters are hardly predictable is its innovative nature. From the definition, an innovation as a translation of an idea or invention into a product or service that creates value is an exploration into unexplored areas. The leader of the team must, therefore, face a high level of uncertainty.

The main aim of any invention application outcome is to further satisfy the needs and improve selected processes. But in all cases it is a business process involving at least two organizations: a customer and a solution provider that must cooperate under a contractual relationship, i.e. a voluntary, deliberate and legally binding agreement between them. The contractual relationship is evidenced by an offer, an acceptance thereof, and a valid (legal and valuable) consideration.

To make the procurement process transparent, fixed-price and fixed-term offers are usually expected to simplify the comparison and selection of a bid for contract award. As a consequence, the quantitative nature of the comparison relaxes the responsibility of the target company (customer) management involved in the selection process, which makes the selection process offer-centric and neglects the uncertainty of the proposed terms. In some circumstances it may cause an assessment of just a "wish list", but not a realistic proposal and lead to circular impossibilities:

1. It is impossible for the customer to prepare the specification because the customer is unaware of the necessity of exploration.

2. It is impossible for the solution provider to prepare the offer as the specification is inadequate and the unanswered questions can be addressed and worked out as project goals only.

The procurement issues described above could be partially solved using direct negotiations or the single-source acquisition method. Unfortunately, both suffer from the non-quantitative nature of the selection process and usually are an exception to the typical procedure. Nevertheless, as the quantitative assessment is difficult or even impossible, they might be a better choice.

The discussion about the procurement process is out of this paper scope. However, in spite of the selected procurement method, the question how to limit the budget, determine the time frame and define the expected scope and quality in the contract is still open.

There are a variety of methods supporting budget assessment, but it is always only assessment as a result of the fact that the project scope is unpredictable. In most cases unpredictability means that some aspects of the research and development work are not calculated (are omitted) in the cost estimation, which usually leads to budged underestimation. Appling an arrangement under which a solution provider (contractor) is paid on the basis of actual cost (time and material contracts) alone puts the customer into an unprivileged position because there are no measures that can be used to asses if the paid workload is really required, i.e. it is relevant and within the scope of the project. It could easily lead to endless projects, as there is always something to discover as a part of the research and development activity. Especially in case of direct negotiations or single-source acquisition method for fixed-price contracts the budget overestimation could be enforced as a method to address the inevitable project unpredictability. For the customer, both situations are usually inacceptable and a compromise must be worked out.

Detailed definition of the contract rules is far beyond of this paper scope, but it is only proposed to assume that the contract imposes a budget limit and is paid on the basis of actual cost. Additionally the solution provider is tasked with making the final cost estimation on continuous basis to recognize the situation when the project budget is put in danger. In this case the following options could be applied:

- the budget is renegotiated;
- the scope of the work is limited;
- the project is canceled.

For the problem described above, the article proposes a methodology framework that tightly couples:

- *Agile approach* to dynamically control the work scope and time framework
- *Workload tracking* to precisely control the value for money

Agile management is recognized as a methodology that helps us to guide software development projects towards the most valuable outcome possible [6] [7]. To address this question the existing agile approach (sometimes called philosophy [6]) must be implemented in the context of contract rules (section 2). It should be non-invasive and effortless, but implemented as strictly observed rules described in a formal way by the contract to control the development process with the goal to assure customer satisfaction under the contract limitations. It is proposed to deploy it as a consistent solution (section 4) using supporting tools developed on the business process modeling basis (section 3).

2. Agile Based Contract Rules

2.1. **Objectives**

Generally, the innovative projects are aimed at improving the selected key performance indicators (KPI) as a result of process control or business automation using novel architecture, algorithms, business process patterns, software, hardware, etc. It requires providing a variety of:

- out of the box products,
- custom services including designing specifications, developing, integrating, testing, training, commissioning and supporting.

Usually the appropriate selection of products is an outcome of services and depends on their quality and performance (the purchase procedure is beyond the scope of this paper).

One of prerequisites for the achievement of the goal, i.e. real improvement in the selected KPI, is the use of such solutions and implementation of such target solution functionalities that the expectations of the parties concerned are met to the greatest possible extent within the agreed time frame and cost limits.

Here the optimization of:

- the scope of work that will enable the customer to achieve the business goal, and
- the implementation team's efforts to maintain maximum work efficiency and quality of the provided deliverables, which, consequently, significantly reduces costs

is of crucial importance.

It is, therefore, proposed that well-proven practices derived from Scrum [6] [7] [8] [9] [10] methodology related to management of development and implementation work within the project should be used. Scrum is proposed because it concentrates on the management aspects, which makes the adoption process straightforward. The *Product Owner* role defined by the Scrum methodology is also the best candidate to be tasked with representation of the business objectives on the continuous basis during the whole project lifecycle. A well-defined roles set proposed by Scrum could be also expanded non-invasively by adding roles in charge of taking care about the formal and legal course of the agreement and controlling of the work environment.

On the basis of agile philosophy it is assumed that optimization can be achieved if cooperation is based on the following assumptions:

- *Cyclical nature* - work is carried out in pre-scheduled time cycles (milestones), which allows the customer to control the scope and progress of work and value for money.
- *Mutual trust* – the application of methods and tools to ensure great transparency in the process of developing and implementing new findings and deliverables, which allows the customer to fully monitor work.

2.2. **Methodology Rules**

It is proposed that the following business model should be used to carry out tasks in the area of cooperation:

- Under an agreement of business cooperation (framework contract) concluded for an indefinite or long period of time, the solution provider will implement and maintain solutions in the area of collaboration stipulated in the agreement.
- Further development of the implemented solutions will be effected as an extension to the basic scope of the agreement.
- To ensure appropriate competitiveness conditions, the rules of Scrum project management will be implemented.

The contract as a formal set of statements provides business environment for any activities related to the development and deployment of project deliverables. The cooperation agreement will define the organizational framework and basic accounting principles of the further work.

In all projects referenced in Section 1 as the experience source the target was reached as a series of contracts. For the innovative research projects it is a typical behavioral pattern as exploration into not well-known areas usually yields discovery of new ones also worth exploring next time. To limit this naturally infinite process there must be defined performance key indicators to assess quantitative and qualitative objectives before making decision in this respect.

It is, therefore, proposed to define the contract as a foundation for carrying out a series of loosely coupled projects, instead of a series of contracts. The main aim of the contract is to define basic rules related to new project scoping, budgeting, scheduling and progress controlling.

The proposed approach is mutually beneficial. One of the benefits for the customer is an obligation of the supplier to provide appropriate resources to address any problems in the area of cooperation. This model is also beneficial for the supplier because it offers prospects for a continuous fixed-term solution delivery process.

It is proposed that the contract has to define procedures compatible with Scrum project management and consequently the following roles are featured as a part of collaboration:

- *Managers* (one for each party) - are responsible for the formal and legal course of the agreement and controlling of the work environment.
- *Business Owner* (customer's employee: like Product Owner in Scrum methodology) - is responsible for defining and prioritizing requirements in the initial phase of each milestone (iteration in Scrum methodology) to ensure maximum improvement in the efficiency of business processes.
- *Team Leader* (provider's employee: like Scrum Master in Scrum methodology) - is responsible for cooperation between the *Development Team* and the *Business Owner* and for compliance with the contract rules governing management of the project.
- *Development Team* (provider's employees) - is responsible for selecting requirements (scope of work) to be fulfilled within a milestone from the *Basic Product Backlog* (equivalent to Product Backlog in Scrum methodology) and

the milestone length at the beginning of each milestone and for the transfer of the developed deliverables to the *Business Owner* before the milestone end.

The names defined by Scrum are changed to emphasize the responsibility modification. For example the *Team Leader* must not only follow the Scrum rules but also take care about contract limitations and obligations. It requires some familiarity with the local legal system. The *Business Owner* is additionally responsible for validating of the workload settlement in the context of the selected milestone scope.

Work scope selection, designing, preparation of deliverables (software, documentation, graphics etc.) and deployment (installation, configuration, testing, and documentation) will be effected in specific time cycles called milestones. The milestone length should be fixed each time and should not exceed 2 months. It is the upper limit that should be a compromise between the development needs and business bureaucracy inertia.

Each project begins with defining the direct target, initial functional requirements (a set of required functions) and non-functional requirements (a set of required solution features) that make the *Basic Product Backlog*; that backlog gives grounds for selection to determine the scope of further work in the next milestone.

The *Development Team* selects requirements at the beginning of each milestone, on the grounds of their priorities agreed with the *Business Owner* and the scope of work selected to be carried out in that milestone, which (according to the Scrum rules) remains unchanged for that milestone. The milestone scope selection is a negotiation process based on the business objectives represented as requirements priorities and technical limitation analysis. Unfortunately, it usually causes a longer iteration cycle because it is not enough to provide something that works, as it has to work also for business. Good examples are code refactoring in software development or design documentation preparation. For business, this work provides no value at all so if required it must be included as negligible part of the whole milestone scope of the work.

The *Development Team* hands the products of the completed milestone (including deliverables such as software code, documentation etc.) over to the *Business Owner* at the milestone end.

Any work that has not been completed during the milestone for any reason returns to the *Basic Product Backlog* and is subject to priority analysis and technical dependencies for the next milestone. All new requirements defined in course of work are entered into the *Basic Product Backlog* as well.

The reported team workload and the value of granted licenses and copyright form the basis for settlement. The workload is settled on the grounds of monthly reports submitted to the *Business Owner* and the *Managers*. It is calculated at the common basic rate per man-hour. It is important to apply the common basic rate with the goal of mitigating customer's influence on association of the team members and tasks.

On completion of the project, the customer gets a license granting him a right to use software and supplied deliverables.

In order to minimize the risk of underestimation or overestimation of the project budget in case there is a great deal of uncertainty about the research field, a feasibility study, design work or a pilot application may be executed in order to determine:

- qualitative (direct and indirect) business goals,
- technical solutions optimal against selected KPI,

- necessary investment outlay on the purchase of third party products,
- own and third party workload,
- maintenance costs of the proposed solution,
- selected economic efficiency indicators,
- risk analysis as a component of risk management; it consists of identification of possible negative external and internal conditions, events or situations.

Additionally, the risk of budget estimate incorrectness can be minimized thanks to:

- the work cyclical nature, i.e. the possibility of finding an error at an early stage of project execution,
- billing work intensity, which balances risks of both the partners and makes overpayment in case of overestimate impossible,
- maintenance work that ensures the continuity of system operation through the quality of the rendered services instead of a guarantee; it does not exclude guarantee commitments for certain products.

3. Implementation

3.1. **Prerequisites**

Successful implementation of the proposed methodology requires appropriate measures to:

- Implement billing principles.
- Support defined roles and management rules in a consistent way.
- Promote mutual trust.

All are closely related to rules that must be concluded by the contract. There is no doubt that it requires appropriate tool selection generating reports formally acceptable to the customer Account Department to effect the settlement and effectively support management rules at the same time. The main purpose of this section is the definition of requirements that have to be met by the tool.

A variety of out of the box products supporting implementation of the Scrum management rules are available on the market. A set of primary prerequisites for the proposed approach must be defined to select one of them or decide to develop a new one. All of them can be grouped into the following categories:

- *Business processes*: to organize the work.
- *Data model*: to represent vital information.
- *Functionality*: to process the data according to the business process rules.

Even though the detailed description of all the prerequisites is beyond this paper scope, the topics most important to the proposed methodology implementation will be discussed in the following subsections in hopes of making the decision process easier. The discussion could also be used as a starting point to expand the existing products by the end user or vendor with the aim of implementing the contract rules proposed in Section 2. The requirements defined in this section are vital for the decision as to how the presented methodology should be deployed in the case study presented in Section 4.

3.2. **Business Processes**

Any business process is a series of logically related activities or tasks (such as planning, research, development, design, testing, documentation, etc.) performed together to produce a defined set of results. According to the Business Process Model and Notation specification [11] a business process can be formally modeled as:

- *Roles*: a set of actors engaged in activities
- *Activities*: a set of actions and events
- *Relationships*: a set of workflows and roles communications
- *Artefacts*: a set of data objects, groups and annotations

For the methodology proposed in this paper, the roles together with their responsibilities and expected behavior as defined in Section 2.2 must be transformed to actions and workflows.

Communication of roles is crucial to the performance of actions and final results quality for the kind of business processes discussed in this paper. The main feature of this communication is unpredictability of forms and formats selected for this purpose by the *Development Team*. For example, brainstorm results may be documented as a thread on a discussion board, picture of notes/diagrams on a white-board or meeting minutes. Unfortunately, the results remain often undocumented at all and the supporting tools must, therefore, engage the *Development Team* to select one and provide documentation for the most important collective actions.

On the other hand, the selection of forms and the necessity of preparing documentation must follow the general agile management principles, for example documentation of a *Development Team* daily meeting can be recognized as time wasting.

Business processes aimed at accomplishing an innovative project may produce a variety of results but intellectual properties are usually the most important ones, i.e. discoveries, formulas, inventions, knowledge, registered designs, software, know how, etc. In spite of their intangible nature they must be documented to be useful as an outcome of the project governed by contract rules. In this case documentation is a representation of something that has no physical existence otherwise. Documentation or other forms of intellectual property representation are of special importance as the contract must not be used to describe any form of abstraction.

Business processes are usually expected to produce results of the highest possible quality level. Because the *Development Team's* outcome is intellectual property (some kind of abstraction) the question how to improve the quality arises. To address this necessity it is proposed to combine the following tracking functionalities into one consistent mechanism:

- Deliverables including code
- Tasks and issues
- Workload
- Entities representing the process state

Deliverables tracking is used to maintain current and historical versions of files such as the source code, web pages and documentation. The tasks and issues tracking mechanism allows the team members to follow the sequence of actions undertaken to fix the problem or obtain the requested result. An association of each reported workload with a task and team member should be a good motivation to improve individual performance and engagement [13] [14]. It is worth noting that during the task lifecycle it typically changes the current owner many times. Any record describing a workload must, therefore, preserve information about the associated team member's activity. The tracking mechanism should also facilitate diagnostics and finding workaround solutions. An entity is a collection of properties representing selected data – a class - of the current process state.

3.3. **Data Model**

Contract settlement and, finally, billing requires accuracy. To put trust on the accuracy, the settlement mechanism must be easily verifiable on a continuous basis, and unambiguously associated with the related workload.

In spite of the management rules governing the project, the team work is organized on the task basis. Tasks are defined to describe work needed to fulfill requirements planned for the selected milestone. This relationship between *Task* and *Resources* is dynamic and can be changed several times during the task lifecycle. Each reported workload must be associated with a relevant task. To promote auditability and team members engagement, the reported workload must be also accumulated for each member individually and associated with the task. Both associations are permanent for the whole task lifecycle. A follow-up class diagram is shown in Figure 1.

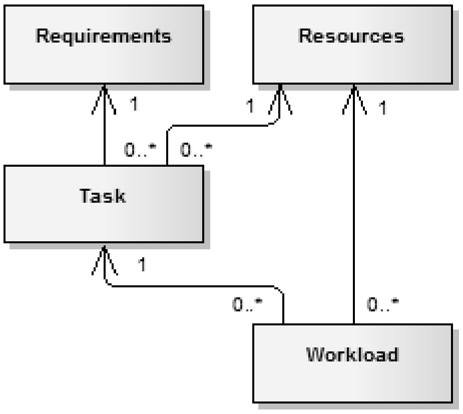


Figure 1 Task relationship diagram.

To successfully finish any milestone, all associated requirements must be accomplished. Before being selected for a milestone all requirements make up the *Basic Product Backlog*. The *Business Owner* should be able to add, update, prioritize, split, merge and categorize them.

The contract billing rules use *Project* class entities to represent:

- Budget limit
- Scope
- Timeframe
- Quality

The *Project* is, therefore, a collection of *Milestones* (Figure 2). This way functionality counting the reported workload for the project can be easily implemented.

The requirements planned for any milestone and related tasks can be used to manage and monitor the project scope. To facilitate this process and allow all participants to monitor the work progress, the entity of the *Task* class has to expose status information. Each task is a collection of actions reported as workload items and should be defined in terms of the baseline start and end times planned to realize it and actual timing information collected from the underlying *Workload* records.

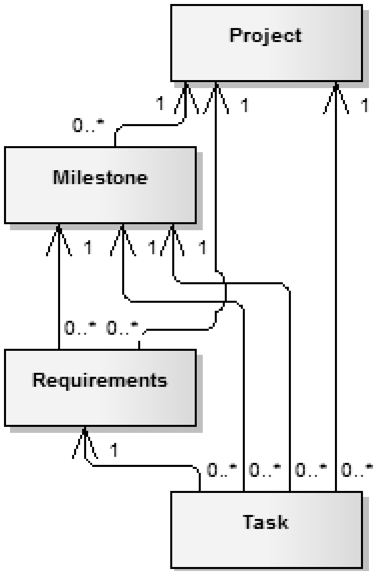


Figure 2 Project relationship diagram

To support diagnostics and maintain quality, the model shown in Figure 2 has an additional relationship between the *Task* and *Milestone* classes to provide supplementary tracking information that allows the current owner to recollect situation at the point in time it was created. Registering all modifications of entities shown in Figure 1 and Figure 2 should be required for the same purpose.

According to Section 2.2 the *Contract* class is a collection of *Projects* (Figure 3). Its role is to provide a set of rules governing the engaged parties cooperation. Therefore, together with the surrounding entities, it should provide information allowing mangers to:

- Synchronize and schedule work realized as separate projects
- Optimally use resources

Sometimes splitting even closely related work is necessary in case a different *Business Owner* must be assigned. Engaging the same team into many projects must be limited by the working hours – team capacity. Theoretically the optimal load of the team as a whole is near 100% potential capacity limited by the working hours. To fulfill this requirement the *Estimation* class is added to the model (Figure 3). At the planning stage its role is to establish the *Development Team* of a project and assign estimated workload to the team members. It is worth noting that resources usage must be monitored for each individual separately irrespective of its teams association.

3.4. **Functionality**

Generally speaking, implementation of the methodology proposed in this paper requires functionalities that may be grouped as follows:

- *Contract management* centric
- *Project management* centric
- *Product management* centric

People hate to track time, but it has to be done to provide the project settlement mechanism. Time tracking should be fast and easy, however, since progress reports and contract billing are based on accurate time records, it must prevent team members from reporting workload overlapping in time and associated to more than one task. To facilitate workload reporting, the user interface may offer functions like stop-watch, workload entities snapping, common activity reporting, etc.

The workload tracking result may also provide a very good feedback that can be used for further improvements of the project scope planning, budgeting and personal improvements. A good source of feedback making a common experience foundation for planning is a well-suited periodical report generation mechanism. A different solution is required for the analysis of the individual engagement. It could be obtained by using a personalized dashboard that integrates workload reporting and monitoring of reported workload.

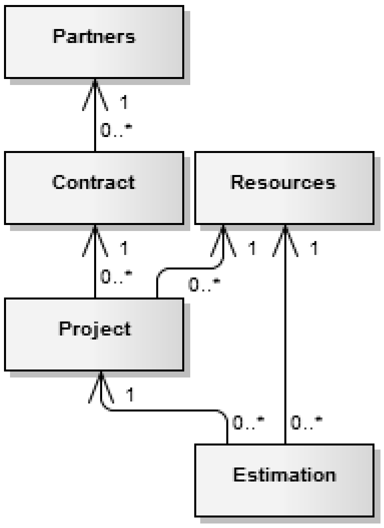


Figure 3 Contract relationships diagram

The project management has to be supported at least by:

- Requirements/tasks management
- Work organization and planning
- Team member communication and documentation
- Deliverables management

The main challenge faced up by innovation deployment projects is commercialization of the results. Even if a result is launched as a prototype, a release procedure and lifecycle management support must be considered. It requires documents versioning, i.e. assigning either unique version names or unique version numbers to unique states of document sets (e.g. computer software).

4. Case Study

4.1. **Deployment**

For a successful deployment of the contract aware project management methodology presented in this paper, the following nonfunctional requirements are of great importance:

- Remote access to the main functions for all parties.
- Possibility of offering the application as a service (cloud computing).

Typically, parties to a contract are not sited nearby and, hence, are not able to use common IT infrastructure. They have to use remote access to common resources instead. Moreover, it must be assumed that contract parties are independent organizations and, therefore, the application used to support management must be offered as a service to at least one of them. Consequently, the contract rules must also cover a Service Level Agreement and infrastructure must meet the cloud computing requirements.

Agile Workload Tracker [15] software package is a solution meeting the requirements presented in this paper. It was designed as a customization and extension of SharePoint 2010 (Microsoft's widely used collaboration software) instead of developing the application from scratch. It is an Internet-based and cloud computing ready platform providing a variety of out of the box functions dedicated to support team work, including documents management and workflows.

Additional functionality dedicated to supporting the proposed methodology has been implemented using a SharePoint server and client site application program interface (API). Both are very powerful foundations for the deployment of custom functionality. Using reusable software mitigates development costs and locating the functionality in a cloud mitigates deployment and maintenance costs.

Using SharePoint only the software code versioning and modification tracking are not satisfactorily supported. Fortunately, API and email processing service offered by SharePoint make the integration with existing versioning software straightforward.

4.2. **Usage**

Since 2006 an independent workload tracker tool has been developed and successfully used to improve research team performance in a number of projects, e.g. [3] [4] [5] as the result of:

- Controlling the engagement of team members in particular contracts.
- Measuring member's effort related directly to contracts in comparison with time spent on general operational activities.

The idea was to create and apply a team motivation mechanism.

Since 2011 a limited version of:

- Agile project management rules based on the Scrum principles described in Section 2;
- Tools supporting project management and workload reporting described in Section 3;

has been used internally as a basic management methodology in a projects aimed at deploying a family of solutions [5] [16] (contracted by a company from the fast moving customer goods industry) with the main goal of limiting the scope of the research and development and gaining the prospective customer. Because the solution was limited to internal use only, to mitigate the project uncertainty the first stage was dedicated to prepare design documentation to precisely define the project scope and deployment environment. Starting form a few pages of initial specification provided by the customer to process the procurement procedure the 600 pages professional design documentation was prepared providing:

- Target domain-centric business process model to define roles, activities and main objectives of the proposed solution.
- Functional and non-functional requirements to define the scope of the project.
- Use cases models to figure out how the requirements are to be fulfilled.
- Coverage matrix to make sure that all requirements are addressed by at least one use case.
- Implementation and deployment models to make sure that the final solution is applicable in an existing environment.

Unfortunately up to now no one have read the documentation in details. The specification was almost neglected and recognized as prepared by the developers for developers only. Fortunately the documentation was used partially by the team as a road map helping to avoid decisions leading to blind ends. Finally, the documentation usefulness and its positive impact on the project performance are hard to be proved but there are no doubts that this approach was unpractical for this project at all.

Table 1 Team projects comparison.

Project	Year	Workload (a)	Budget (b)	Overdue (a/b)
Project 1	2011	1973,25	500,00	294,65%
Project 2	2012	2481,04	1000,00	248,10%
Project 3	2013	636,50	624,00	102,00%
Project 4	2013	967,52	850,00	113,83%
Project 5	2013	509,96	500,00	101,99%
Project 6	2013	525,33	450,00	116,74%
Project 7	2014	2061,62	2212,50	93,18%

The key performance indicators for the team involved and its projects aiming at deploying the mentioned above products family are presented in Table 1. In the table the budget limit and reported workload are expressed by normalized values to minimize the influence of condition fluctuation over years and does neither represent directly real budget nor workload (non-disclosure limitations). All projects are contracted and realized on a fixed-price basis in spite of the fact that the contract allowed adaptation of the budget in case of any need to change requirements significantly. The team members are involved in other activities at the same time. To make auditing possible, i.e. reporting daily workload of individual members, all working hours are reported and allocated to separate projects.

From this table we can learn that the financial performance for Project 1 and Project 2 was not satisfactory. At the same time the customer satisfaction was also far from expectations in spite of efforts spent to obtain a better result. Because the overall conclusion of the survey was: “Some improvements have to take place to continue cooperation” at the end of 2012 a new agreement was made under the assumption that new projects should be contracted applying the rules described in this article, i.e. agile management embedded in the contract rules.

After applying new contract rules in the early 2013 an extraordinary improvement of the projects key performance indicators proves that using the agile approach alone for fixed-price contracts does not guarantee a satisfactory solution. It must be also noted that the customer satisfaction rated as an index 0-100 reached the max value 100, i.e. overall conclusion of the survey was “cannot be better”.

5. Conclusion

The innovative process control and business management research projects aimed at providing solutions that cannot be just a copy of a well-known „technical pattern” suffer from uncertainty and are recognized as a high risky business activity. It is caused by the work scope that is hard to be predicted and described as it must embrace exploration of critical areas not jet discovered.

The paper presents the methodology and tools that can be used for calming down as a result of a tight coupling of:

- The Scrum project management methodology.
- Workload, tasks, and source code tracking.
- Appropriate legal instrumentation.
- A set of software tools and dedicated workspace to improve trust and support seamless control of defined principles, operational coordination, and make the cooperation transparent for all parties.

The selected practical results are also presented as a use case analysis. The case study shows that using the agile approach alone does not improve the business efficiency and customer satisfaction. It can be seen that the real improvement is achieved after converging selected agile management and contract legal rules including but not limited to the workload settlement.

The presented result leads to a very interesting question of what is the cause that has a major impact on the business performance and customer satisfaction improvement. To answer this question independent research must be undertaken, but it is worth noting that applying the new approach changed the role of customer representatives significantly.

Before applying the proposed methodology they are responsible for auditing of the final output for conformance to technical, reliability, maintainability, and performance requirements. It leads to arguing and arguing the question “why a …..

functionality/feature is not supported" with the *Team Leader* gives always the same answer "because it is not included in the specification, which has been already approved by the contract parties". From the discussion in Section 1 and 4.2 we know that authoring the specification and making it comprehensive both for the end user and development team is an unrealistic goal and unpractical approach for many reasons. This limitation is usually taken as an opportunity to assume that there are "obvious" requirements that must be fulfilled by any professional team regardless of whether they are in the specification or not. Main concern is that we have not adequate measures to distinguish obvious and not obvious requirements at the commissioning stage of any project. Any discussion with the customer then becomes slightly difficult.

After applying the methodology proposed in this article the contract representatives become active members of the development team contributing to the selection of the research directions and scope of the work. At the same time they must share responsibility for an overall project success. It changes seamlessly the project into an adaptive process relaying on the customer involvement on the continuous basis. If that is the case new requirements are also formulated by the development team. The team's proposals like "maybe this … functionality/feature will be useful" are replayed by "yes, provided we have budged and time to accomplish it".

Finally the above discussion can be concluded that the methodology presented in this article is a proposal of a practical implementation of adaptive processes [6] in the context of real contract limitations like a fixed price.

References

[1] CAS. Cfis-1 -flight inspection system of the radio navigation aids. *http://www.cas.eu/Projects/FlightInspectionSystemoftheradionavigation.aspx*, 2015.

[2] D. Arendt and M. Postol. Real-time multiprogramming system for mine control centre. *Microprocessors and Microsystems*, 14(1):39–46, 1990.

[3] M. Postol. Large scale distributed process and business management integration. *In 14th International Congress of Cybernetics and Systems of World Organisation of Systems and Cybernetics*, Wroclaw, 2008. Politechnika Wrocawska.

[4] M. Postol. Real-time communication for large scale distributed control systems. In International Multiconference on Computer Science and Information Technology, 2007.

[5] CAS. Shepherd application; optimal management of inbound and outbound deliveries. http://www.cas.eu/Products/YARD/Shepherd.aspx, 2015.

[6] M. Fowler. The New Methodology. http://martinfowler.com/articles/newMethodology.html, 2013

[7] M. Rizwan J. Qureshi. Agile software development methodology for medium and large projects. Software, IET, 6(4):358–363, 2012.

[8] K. Beck. The agile manifesto. www.agilemanifesto.org/principles.html, 2001.

[9] D. Jamieson, K. Vinsen, and G. Callender. Agile procurement and dynamic value for money to facilitate agile software projects. In 32nd Euromicro Conference on Software Engineering and Advanced Applications, SEAA, pages 248–255, 29 August 2006 through 1 September 2006.

[10] J. Shore and S.Warden. The Art of Agile Development. O'Reilly Media, Inc.,, Sebastopol, 2007.

[11] Object Management Group. Business process model and notation. http://www.bpmn.org/, 2013.

[12] K. Schwaber. Agile project management with Scrum. Microsoft Press, Redmond, Wash., 2004.

[13] A. Cockburn and J. Highsmith. Agile software development, the people factor. Computer, 34(11):131–133, 2001.

[14] C. De O. Melo, C. Santana, and F. Kon. Developers motivation in agile teams. In 38th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2012, pages 376–383, 2012.

[15] CAS. Agile workload tracker software. http://www.cas.eu/Products/AWT.aspx, 2015.

[16] CAS. IPR application; electronic inward processing relief. http://www.cas.eu/Products/IPR.aspx, 2015.

Chapter 4

The Project Management Perspective on Software Value: a Literature Review

1. Introduction

In today's cutthroat product and services industries, software has become the main driver for competitive advantage, enabling faster and cheaper innovation and product differentiation. As the size and complexity of software-intensive solutions increase, so does the impact of software development decisions on the overall product offering. That is, any decision taken regarding software product/project management and development (e.g. what features to design, what quality to offer, or what technology to choose) could impact the entire product's/project's life cycle and value, not to mention that it could limit future possibilities and direction of both the software and business 7[2][3]. To remain competitive, innovative and to grow, companies must change from cost-based decision-making to value-based decision-making where the decisions taken are optimal for that company's overall value creation [4][5].

Previous studies have proposed value considerations and corresponding measurement solutions needed for making decisions about software product management and development [6][7][5][8][9]. Moreover the two recent literature reviews in the subject leave the software project aspect unaddressed [4][5]. However, these contributions were often isolated and with a limited perspective; for example, focusing only on cost, or on product characteristics such as usability. Consequently, a complete picture of value considerations relevant from different perspectives, and required for taking software product management and development decisions, was missing.

Khurum et al. [4] proposed a consolidated view of value – the Software Value Map (SVM). SVM characterizes software value according to the four major value perspectives - customer, internal business, financial and innovation & learning. Each value perspective is detailed further using a nested structure in which a perspective is a root node, and split into Value Aspects (VAs); these in turn can be split further into Sub-Value Aspects (SVAs) and Value Components (VCs), which are the leaf nodes.

The VCs are used by decision makers as decision support when identifying and discussing important value components. VCs can be measured on a five-point ordinal scale (very negative impact, negative impact, neutral, high impact, very high impact) representing the impact that the implementation of a given decision scenario may have on a VC. The SVM currently has 32 different VCs; however in practice professionals only use a subset of these within the context of a decision scenario being discussed - the most important VCs. Such subsets are called Impact Evaluation Patterns [10].

The SVM contains several value aspects that are related to software quality aspects outlined in e.g. ISO/IEC 25000. SVM was developed and validated in collaboration with Ericsson, and presented promising results; however, it still needed to be evaluated in software project intensive contexts that represent a view of value within the scope of project management and development. The current focus of SVM is on product management and its inherited characteristics, leaving the project managers without quality support for identifying and managing value on the project level.

The goal of this paper is therefore to identify additional value components from the project management perspective in order to expand the current body of knowledge regarding value considerations within the context of both product and project management and development. In order to accomplish our goal we carried out a snowball literature review.

Recent master thesis effort [5] carried out a systematic mapping study on value-based software engineering relating to product and project management; this study is grey literature and did not provide the level of detail we are focusing upon herein. The work of Khurum et al. [4] focuses on product management view on software value and therefore provides limited usefulness for this work.

This work provides a comprehensive view on software value in software project management coming from a snowballing literature review that was grounded in previous work. We identified nine papers that focus on project management perspective on software value, grouped into three themes: financial analysis of a project, risk analysis within a project and process improvement based on project assessment. From these studies, we suggest seven extensions to the current Software Value Map.

The remaining of this paper is organised as follows: Section 2 provides related work, followed by a description of the research methodology in Section 3. Results are presented and discussed in Section 4, followed by suggested extensions to the SVM in Section 5. Finally, a discussion on our experiences carrying out a snowballing literature review and conclusions are given in Sections 6 and 7, respectively.

2. Related Work

Two literature reviews are relevant for this study. In the first study, Khurum et al. [4] survey related literature using a systematic mapping methodology, followed by snowballing. The authors identified 30 value aspects classified into the customer, financial, innovation and learning and internal business perspectives. The work focuses on product management perspective and thus does not cover project management perspective on software value.

The second secondary study we are aware of related to the topic of our snowballing literature review is the master thesis work by Nasser and Ibrar [5], where they have conducted a mapping study of value-based software engineering in order to identify what sub-areas of software engineering were represented by the primary studies that were identified. They also identified the types of empirical studies associated with each sub-area. The sub-areas covered requirements engineering, architecture, design, development, verification and validation, quality management, project management, risk management and people management.

27 studies were included within the project management sub-area of the study by Nasser and Ibrar [5], and the results showed that studies focused on the project planning and monitoring, as well as project enactment. Although no evidence was presented on value aspects, we have employed the list of primary studies that were selected within the project management sub-area in order to select an initial seed for our snowballing literature review.

The project management literature often focus on the financial aspect of value that is operationalized using the Net Present Value (NPV) or Earned Value (EV) concepts. For example, Jordanger and Klakegg [10] pointed out that the value concept in a project is often limited to the earned value gained by fulfilling the project's scope. The authors suggest using a utility function that combines monetary utility (NPV) and qualitative utilities. The decision support is realized with the help of Multi Criteria Decision Analysis (MCDA). The paper lacks detailed view on how to break down the NPV and qualitative utilities into detailed characteristics that can be attributed to features or their quality attributes. The analysis remains on high level with functionality, flexibility and community development as value components.

3. Research Methodology

We used snowball sampling to search for relevant literature [11]. There were two reasons why we decided to use snowballing: 1) the two systematic mappings associated with the studied phenomena were performed recently [4][5] and 2) performing the entire systematic literature review would create a risk of a significant number of duplicates as the search string will have to be greatly reused from the previous studies. Snowballing is also recommended as a method for updating the existing literature reviews with new findings [11]. The subsections that follow provide details of the review methodology process.

3.1. **Research design**

We used guidelines provided by Wohlin [11] for performing the snowballing. The process was divided into three steps detailed below: 1) identification of the start set, 2) performing the snowball iterations, 3) data analysis and synthesis.

3.2. **Step 1 deriving the start set**

In this step, we reused the previous literature reviews [4][5] as a starting point towards the start set identification. The first review is published in an international journal [4] but does not primarily focus on project management aspect of software value. The second review focuses on the project management aspect of software value but was not published in a peer-reviewed venue. In addition, we reviewed all chapters from the Therefore, we decided to screen the final sets of papers derived from both studies in order to identify the start set for our study.

Both search strings were examined by the research team and a decision has been made that they were appropriate also for this study, however too broad. As both literature reviews are recent (conducted in 2010 and 2014), the research team decided to re-use these results instead of re-execution of these search strings. Another reason was that citation analysis during snowballing is going to identify potential new papers from both literature reviews.

The results from both literature reviews in terms of the final sets of the papers were carefully examined with the following inclusion criteria: 1) is the candidate paper relevant for the new study (has project management perspective on value), 2) was it published in a peer-reviewed venue and 3) was it written in English. The order of screening was title, abstract and (if necessary) full paper read.

To minimize wrong selection and the selection bias, the first and second authors independently screened the candidate papers and compared the results. Disagreements were iteratively discussed and resolved in several meetings. No studies from the literature review by Khurum et al. [4] were included in the start set. Six studies out of 27 that focused on the project management sub-area in the literature review by Nasser and Ibrar [5] were selected as the start set [12][13][14][15][16][17][1]. Finally, we also reviewed the book about value-based software engineering edited by Biff et al. [27], but did not find any chapter meeting the inclusion criteria.

3.3. **Performing snowball iterations**

We performed two iterations, both initiated by looking at the references. The references were extracted into an Excel file, organized and independently judged by both researchers. Next, citations were extracted with the help of Google scholar. Google Scholar provides a comprehensive database of citations that often include peer and non-peer reviewed work. Despite increased screening effort, this brings the confidence that no citations are missed in the analysis. The citations were also independently judged by the two authors. After performing independent judgments, discussion meetings were held where independent judgments were compared, analyzed and conflicts were resolved. This was a pre-requisite to consider each iteration done and move on to the next one. The inclusion criteria remained stable between the start set selection and the snowballing iterations. We did not screen citations of citations for each considered candidate, this was done in case an article was accepted to the next snowballing iteration.

3.4. **Data extraction and synthesis**

We performed several analysis steps here. Firstly, we performed thematic analysis with the help of suggestions from the work of Cruzes et al. [18][19]. The thematic analysis enabled us to perform metrics identification as a next step. We extracted the relevant metrics by classifying similar value constructs with different names under the same value metric. Next, we performed Rigor and Relevance analysis, supported by the guidelines suggested by Ivarsson and Gorschek [26]. We graded low rigor studies as studies that received the score from 0-1.5. Similarly, we consider studies as low relevant if their relevance score was below 2. The results of the rigor and relevance analysis are visualized in Fig 3, positioned into four quadrants (A, B, C and D).

[1] The papers included in this review are marked with **C1, C2, C3** etc…, please see the end of the reference record for their identification.

3.5. **Validity**

Internal validity. We made an attempt to minimize the researcher bias by following the guidelines for conducting snowballing suggested by Wohlin [25] and the quality assessment criteria suggested by Ivarsson and Gorschek [26]. Moreover, the selection criteria were extensively discussed among the researchers and written down to enable consistency analysis and disagreement discussion.

Construct validity is concerned with the presence of potential confounding factors that hinder achieving the planned study aims and objectives. In this case, the multiple definition of software project management can be considered as one of the confounding factors. In order to minimize this threat, we based our understanding of software project management on the definition provided by the PMBOK[2].

Conclusion validity is concerned with possible factors that made the data collection and analysis dependent on a specific researcher. We minimized the risk of missing important papers by studying two previous systematic literature reviews and deriving our start set from them. Furthermore, to objectively measure the quality of the identified studies, we applied rigor and relevance criteria introduced by Ivarsson and Gorschek [26]. Finally, both study selection and data extractions were extensively discussed and reviewed among the authors of this study where disagreements were resolved and procedures were unified.

External validity is related to the ability to generalize the study findings. In this case, the fact that the snowball start set is based on two systematic literature studies increases the confidence of the start set objectivity. However, what remains to be further investigated is the relevance of the identified value aspects since the identified studies were conducted with low rigor and therefore provide weak empirical evidence. Moreover, there exists literature about agile estimating and planning, e.g. [28], however it is in most cases not peer-reviewed and therefore was excluded from this study. Finally, some authors, e.g. [29] or [30] address aspects associated with the main focus of this study, but due to the usage of different vocabulary, terms and definitions, these studies and probably some others are removed during the initial database search as they are strongly related to software value without using the term neither in the title nor in the abstract.

4. Results

This section presents the results of both snowballing iterations, visualized in Figures 1 and 2. We provide here also the results analysis and synthesis.

4.1. **First and second iteration results**

First iteration. The results from performing the first forward and backward snowballing iterations are presented in Figure 1. During backward snowballing 93 references and 98 citations were screened. Paper C3 [14] received most citations (72) while paper C4

[2] http://www.pmi.org/PMBOK-Guide-and-Standards.aspx

[15] has most references. 50 references and 11 citations were removed because they were not peer-reviewed or they were books. 10 references in this iteration were not in English and therefore removed. From the remaining candidates, 6 were removed based on the abstract and 16 were removed after the full read of the paper and discussions between the authors. The uncertainties based on the abstract screening were resolved after reading the full papers. Three papers 2C7, 2C8 and 2C9 [22][21][23] were selected to the next phase.

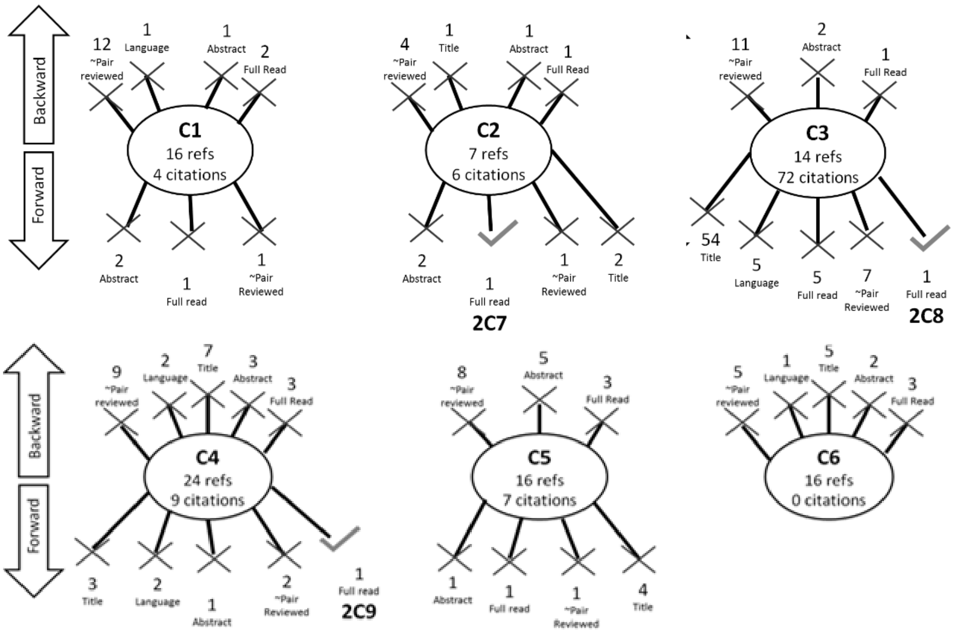


Figure 1. The first iteration results.

Second iteration. Three papers were included to the second snowball iteration [21][22][23]. 51 references and 16 citations were examined in this iteration. 20 references were not pair reviewed and 12 were removed based on the abstract screening, 18 were removed based on full read and 2 were not in English. The screening details are depicted in Figure 2.

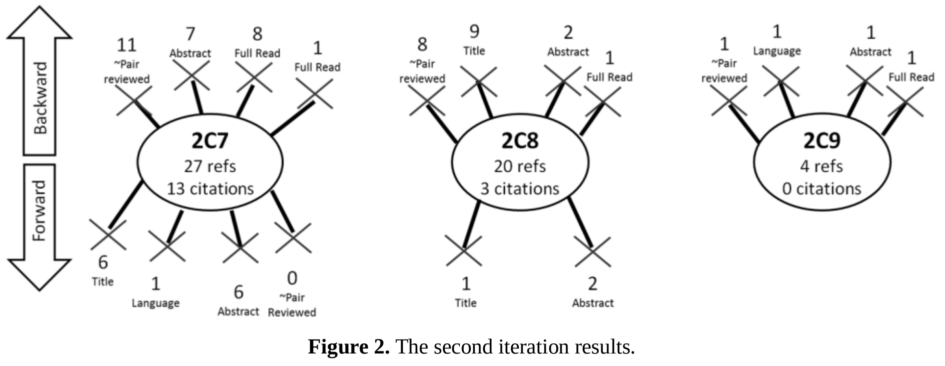


Figure 2. The second iteration results.

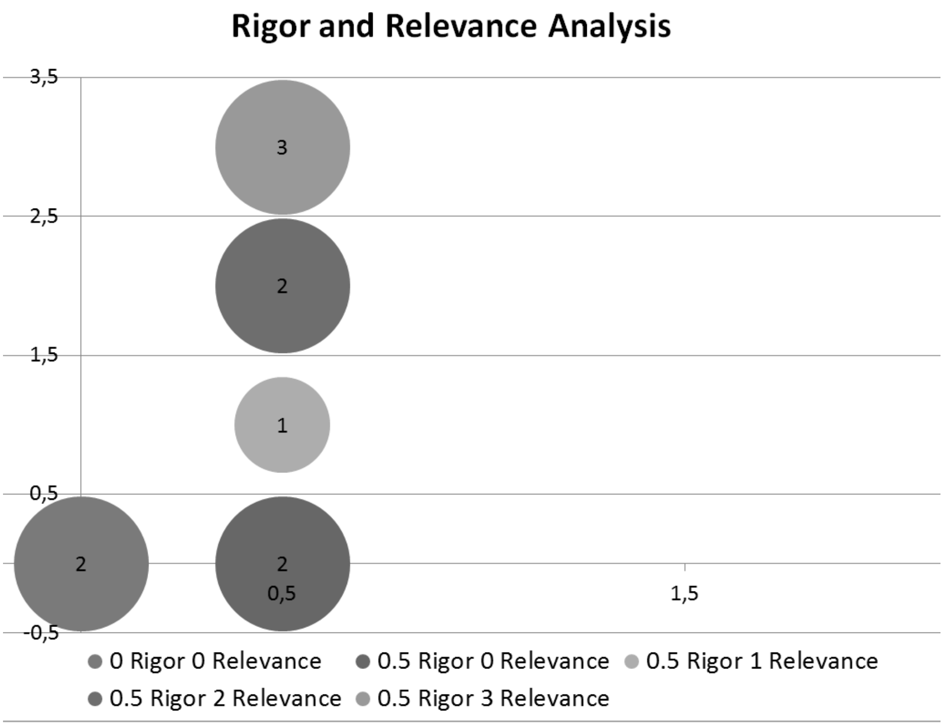


Fig. 3. The results of the rigor and relevance analysis.

4.2. **Quality assessment**

We have performed quality assessment of the identified nine papers using rigor and relevance scores [26]. We used the three perspectives on rigor and four perspectives on relevance, including the scales suggested in [26]. The results are depicted in Figure 3. Two papers scored 3 on relevance score (C5 [16] and C6 [17]) and 2 scored 2 on relevance score (C1 [12] and 2C7 [21]). However, none of the above provided high rigor since all scored 0.5 on rigor. Two papers (C4 [15] and 2C9 [23]) scored 0 on both rigor and relevance. This is mostly due to the fact that these are position papers or poster with very limited amount of provided information. Paper C2 scored 0.5 on rigor and 1 on relevance while papers C3 and 2C8 scored 0.5 on rigor and 0 on relevance.

4.3. **Thematic analysis**

We synthesized the primary studies utilizing thematic analysis technique suggested by Cruzes and Dybå [18][19]. In order to perform the synthesis, we first extracted the necessary data from primary studies. Next, we identified the interesting themes from the extracted data and grouped them into distinct categories. Finally, we evaluated the trustworthiness of the themes by revisiting the rigor and relevance scored for each theme. The resulting three themes of project management perspective on software value are listed below.

4.4. **Financial analysis of a project**

Paper C3 [14] is similar to C2 [13] in regard that it focuses on delivering value rather than exploring various value components. C3 focuses on applying the Incremental Funding Method (IFM) that delivers "chunks" of value to the customers and in this way optimizes the net present value (NPV). It introduces and interesting aspect on value delivery which is time to value and allows better control over the cash flow. This rate at which value is added to the customer is particularly applicable for fast-changing environments. Paper C3 criticizes net cash and ROI as example approaches to value delivery as they may contribute to the discounted cash flows. IFM is based on *minimum marketable features* that are self-contained and can be delivered quickly to the customer. The paper mentioned decreasing architecture value (similar to architecture related internal business perspective value components in the SVM). Apart from that, no value components are suggested by this study. Both C2 and C3 received low rigor and relevance scores which limit the ability to draw strong conclusions from these studies.

Halling et al. [15] (C4) advocate that the current value-based software engineering strategies are cost-oriented or focused on reducing project-level risks and strive for providing a broader model of value creation in software engineering. They adapt the value chain model from Porter [20] to software engineering specifics as well as perform a cost benefit analysis of software products and processes. Halling et al. also mention intangible benefits (like paper C1 by Wagner and Dürr [12]) in terms of increased flexibility and information gain and suggest using known existing corporate finance valuation techniques, e.g. discounted cash-flow and real option theory for supporting development projects.

Halling et al. also discuss the issue of asymmetric information in software engineering project management, e.g. developers bring better feature effort estimates than customers. Finally, the paper brings two interesting perspectives on value-management: 1) modeling uncertainty during the risks analysis phase should also consider unexpected positive outcomes and 2) the project's environment has an influence on the value delivery process. The main issue with this work is that it mixes up project with process and product perspectives. The paper scored 0 on both rigor and relevance which is an indication that the suggestions, although interesting, would have to be empirically evaluated before putting in practice.

Alencar et al. [22] focus on value from the nonessential modules that may positively impact the value of the essential modules. These features are also called enablers [24]. The nonessential features are characterized as minimum marketable features modules (NMMFs), introduced in paper C3 [14]. The authors suggest that early identification of nonessential elements may affect the final value of a project, however what value components they may affect is not provided. The work provides some example of incremental value delivery via a chain of software units that are estimated using net present value and cash flow elements. Different scheduling options and their impact on the final value are discussed. The concluding advice is that a project team should examine the effect of non-essential software units before selecting any scheduling option.

Ereno and Cortazar [23] propose applying the impact of relationships (IOR) measure defined as "the amount of interactions that the social media participants have with their community and how that relationship has impacted the totality of the product". The authors list the following measures associated with IOR:

- Customer expected value (it corresponds to the customer perceived value in SVM [4]),
- Product line value - corresponds to the Internal Business perspective and Physical value in relation to quality and Product Architecture[3]
- Real value (the amount of features that the software product line offers for a specific stakeholder) – corresponds to the Physical Value in relation to Product architecture Functionality value component and also include all implemented features that may not be noticed by the customer.
- Potential Value – this aspect is similar to Customer Perspective perceived value and pragmatic value components if we think about implemented or delivered features that may not be used by the users. If we take another perspective; potential value of the features that exist in a product line but were not delivered in a product than the most similar SVM component is the Physical Value in relation to Product architecture Functionality from the Internal Business Perspective.
- Not Covered Value – the features that the SLP do not cover. The rap between customer expectations and the Real Value (RV). This is little related to Customer Perspective pragmatic value but greatly uncovered in the SVM.

The paper, although interesting, scored zero on both rigor and relevance because it does not go into details and presents tangible value components that can be applied in product or project management.

To summarize, the financial analysis of a project category brings some interesting discussions and suggestions in relation to how to connect value with the financial aspects. However, the focus of these papers is mainly on the exploring the financial part of the equation while the value concept is left with little consideration. Moreover, the papers in this category have low rigor and relevance.

4.5. **Risk analysis within a project**

Huang et al. [13] present the VBSQA process framework for value propositions identification with respect to quality attributes, understood as prioritization of expected and/or desired values. The potential conflicts between different stakeholders' views on quality attributes are evaluated with the help of risk analysis methods. This helps to avoid problems associated with delays or low quality deliveries. The high-level question that the method attempts to answer is: how much investment in quality is enough?

The authors provide three example strategies for delivery quality to the stakeholders: 1) schedule-driven, 2) market trend-driven and 3) product driven. For each of them, the authors discuss the quality and schedule risks, the suggested architecture and refac-

[3] The Software Value Map is available at http://softwarevaluemap.org/

toring approaches as well as the nature of requirements. The schedule-driven business case delivers value by rapidly accommodating small product update requirements. For the product-driven business case, the quality of the product is the main concern rather than the functionality. Finally, for the market trend-driven business case the products' upgrades are based on competitors' activity. Albeit the study discusses how to deliver value to the customers, it does not provide any value components that can be used in SVM. This study scored 0.5 on rigor and 1 on relevance.

Huang [13] proposes a value-based dependability analysis framework for software projects. The method measures the software dependability achievement based on a scenario-based approach to identify stakeholders' value propositions with respect to dependability. The project requirements are derived from the dependability attributes. One of the steps in this method is to identify high level value propositions. Based on the example from NASA, the top level dependability attributes include availability, accuracy, performance, usability, cost and schedule. From the identified scenarios, risk analysis is performed with potential value losses if scenario execution fails. Unfortunately, no detailed value components are identified by this study. The paper scored 2 on rigor and 2 on relevance making it one of the papers in this study that can be trusted the most.

To summarize, this category provides two studies that focus on expanding the risk identification and analysis techniques for providing improved value to the customers. Unfortunately, equal focus is not put on the exploration of the value components that should be taken into consideration when performing project risk analysis. Therefore, we cannot suggest any additions to SVM based on the work presented in this category.

4.5.1. Process improvement based on project assessment

Paper C1 by Wagner and Dürr [C1] proposed a five step method that enriches the "earned value" concept with other important value aspects and supports project managers in value planning. The process involves defining and monitoring process values during the project. Wagner and Dürr [C1] also concluded that value brings important benefits to the stakeholders both tangible and intangible (economic, social, monetary or utilitarian).

During the process value definition phase, the value-based project measures (gathered as expectations during the brainstorming sessions) are identified and mapped to the organization's KPIs. The identified expectations are mapped on four perspectives: process perspective, customer perspective, financial perspective and potential perspective. These perspectives overlap to a large degree with the SVM perspectives.

The provided examples are mostly cost driven and represented by variations of effort put in various phases. Two measures are interesting for this study: the Project Manager's team management capability and Systems Engineer's coaching capability. The paper seems to focus on requirements engineering and system design perspectives within a project.

Ojala [16] applied value engineering principles for collecting experiences in using value assessment of project management tasks. The focus here is also on finding the differences between project management tasks valuation. According to value engineering principles, value is associated with some object, product, service of process and is the ratio between the worth (the least cost to perform the required function or its functional equivalent) and the cost. Another definition of value include (function + quality) divided by cost.

Ojala provides four ways of assessing value in software projects: 1) an addition of defined VE process into the existing process models, 2) value assessment for processes defined in used process models, 3) value assessment for processes without a process model and 4) value assessment of a product.

From the interviews at a large company, it appears that the valuation is performed by assessing the worth of project management asks. These seem quite familiar to requirements or features and are later prioritized by customers and vendors. No additional value attributes are introduced in this paper.

Itaborahy et al. [17] propose a method for project value factors identification and how they can be monitored throughout the project. The suggested approach is designed to help project management in selecting and executing projects that will bring value and generate business returns. The paper associated with the IT investment perspective on project management and how IT leads to value generation by: 1) aligning the company's strategic goals with software project's goals and 2) generating IT assets from these goals.

Itaborahy et al. [17] list ten determinant factors of value in a software project: 1) strategic objective, 2) business process, 3) business transformation, 4) benefits, 5) conversion process, 6) integration process, 7) competition process, 8) time, 9) costs and 10) risks. These factors only focus on generating IT assets rather than aligning the company's and project's goals.

The suggested approach for software project management based on value includes the value model as one of the key components. This value model includes the organizations' expectations of project value in terms of: strategic objectives, the business process to be modified and the required business transformation that enables this success. The value model is accompanied by complementation initiatives, market and project scenarios. No additional or details value components are provided by this paper.

5. Suggested extensions to the software value map

Based on the literature survey results, we suggest the following additions to the SVM:

1. Intangible learning benefits [15] – should be added to the Internal Business Perspective of SVM as a value component that enables increased flexibility
2. Social or utilitarian benefits for stakeholders should be added to the Customer Perspective of SVM [12]
3. Intangible learning benefits [15] should also be added to the Innovation and Learning perspective as we believe some benefits are much more long term and can enable innovation and facilitate learning. The reason is that there could be intangible learning benefits valid within a single project and universal for the entire organization.
4. Value from nonessential modules or features [22] (enablers) should be added to: Internal Business Perspective, Production Value, Physical Value wrt. Quali-

ty, PVq Product Architecture, Functionality. Moreover, the same value branch should be expanded with the notion of Real value identified in [4].

5. The amount of interactions that a feature creates [23] (in the social context) – this aspect should be added next to the Network Externalities in the Customer Lifetime value and Value for Customer part of SVM. Here the difference and novelty is that the interactions are created between the features not the users of a feature as so far suggested by the SVM.
6. The Project Manager's team management capability – this could be added to the Internal Business Perspective, Production value, Physical value, Physical value wrt. Quality and PVq Organization. We believe that the management capability is an additional value of the organization that can be monetized by the software companies. Team management capability is also highly relevant for project management as projects deliver the production engine for product management and actually realize features. Managing teams in this case is highly relevant and critical for success.
7. The Systems Engineer's coaching capability - this could be added to the Internal Business Perspective, Production value, Physical value, Physical value wrt. Quality and PVq Organization. The ability to coach on system engineering concepts is highly relevant and required for projects where knowledge about new technologies needs to be quickly grasped and extended.
8. The value from the project's environment – this should be added to the Internal Business Perspective, Production Value, Physical value wrt. Quality, PVq organization. Each organization operates in a given environment and the impact of this environment should be detailed and estimated in this value component.

6. Conclusions and future work

In this paper, we present the results from a literature review study with an aim to complement the current view on software value with the project management perspective. We have based our literature review on two previous literature surveys [4][5] and performed snowballing in two iterations. We identified nine publications that were analyzed and summarized into three categories: financial aspect of software value, risk analysis within a project and process improvement based on project assessment. From the identified publications, we extracted eight additional value components for the previously published Software Value Map (SVM) concept.

In future work, we plan to empirically evaluate the new value aspects suggested from the surveyed literature. In particular, we are interested in expanding the financial aspect of software value in the project and confront the additional aspects that we suggested with industry view on the financial aspects of software projects.

7. References

[1] Aurum, A., Wohlin, C. and Porter, A., "Aligning Software Project Decisions: A Case Study," International Journal of Software Engineering and Knowledge Engineering, vol. 16, pp. 795-818, 2006.

[2] Harmon, R. R., and Laird, G. "Linking marketing strategy to customer value: implications for technology marketers," in Innovation in Technology Management - The Key to Global Leadership. PICMET '97: Portland International Conference on Management and Technology, 1997, pp. 896-900.

[3] Jeffery, R. Value-Based Software Engineering. Germany: Springer, 2006.

[4] Khurum, M., Gorschek, T., and Wilson, M., "The software value map - an exhaustive collection of value aspects for the development of software intensive products," Journal of Software: Evolution and Process, 2012.

[5] Naseer, J., and Ibrar, M., "Systematic mapping of value-based software engineering – a systematic review of value-based requirements engineering", Masters thesis Software Engineering, Thesis no: MSE-2010:40, Blekinge Institute of Technology, Karlskrona, December 2010.

[6] Gorschek, T., Fricker, S., and Palm, K., "A Lightweight Innovation Process for Software-Intensive Product Development," IEEE Software, vol. 27, 2010.

[7] Kontio, J., Ahokas, M., Poyry, P., Warsta, J., Makela, M.M., and Tyrvainen, P., "Software Business Education for Software Engineers: Towards an Integrated Curriculum," in Software Engineering Education and Training Workshops, 2006. CSEETW '06. 19th Conference on, 2006, pp. 5-15.

[8] Lei, Z., Shouju, R., Procopio Garcia, F., and Zuzhao, L., "Multiple-value decision supporting application in software production facing global market," in Systems, Man, and Cybernetics, 2000 IEEE International Conference on, 2000, pp. 346-351.

[9] Rönkkö, M., Frühwirth, C., and Biffl, S., "Integrating Value and Utility Concepts into a Value Decomposition Model for Value-Based Software Engineering," in Product-Focused Software Process Improvement. vol. 32, F. Bomarius, M. Oivo, P. Jaring, and P. Abrahamsson, Eds., ed: Springer Berlin Heidelberg, 2009, pp. 362-374.

[10] Jordanger, I. Klakegg, O. J., "Value Management Beyond Earned Value", PM World Journal, vol. 2(2), Feb 2013, pp. 2-12.

[11] Wohlin, C., "Guidelines for Snowballing in Systematic Literature Studies and a Replication", Proceedings 18th International Conference on Evaluation and Assessment in Software Engineering (EASE 2014), pp. 321-330, London, UK, May 2014.

[12] Wagner, K. W., Durr, W., "A Five-Step Method for Value-Based Planning and Monitoring of Systems Engineering Projects," 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA '06, pp.282 - 290, 2006 (**C1**).

[13] Huang, L., Hu, H., Ge, J., Boehm, B., Lü, J., "Tailor the Value-Based Software Quality Achievement Process to Project Business Cases", Software Process Change, Lecture Notes in Computer Science, vol 3966, pp. 56-63 **(C2)**.

[14] Denne, M., Cleland-Huang, J., "The incremental funding method: data-driven software development," Software, IEEE , vol.21, no.3, pp.39,47, May-June 2004 **(C3)**.

[15] Halling, M., Biffl, S., Grünbacher, P. "The Role of Valuation in Value-Based Software Engineering", 6th International Workshop on Economics-Driven Software Engineering Research Proceedings", IEE, 2004, pp. 7 – 10 **(C4)**.

[16] Ojala, P. "Value of project management: a case study". WSEAS Trans. Info. Sci. and App. 6, 3 (March 2009), 2009, pp. 510-519 **(C5)**.

[17] Itaborahy, A., De Oliveira, K., and Santos, R. "Value-based software project management - A business perspective on software projects", International Conference on Enterprise Information Systems - ICEIS , pp. 218-225, 2008 **(C6).**

[18] Cruzes, D. S., and Dyba, T. "Research synthesis in software engineering: A tertiary study." Information and Software Technology, 53(5):440--455, 2011.

[19] Cruzes, D. S., Dyba, T., Runeson, P., and Host, M., "Case studies synthesis: A thematic, cross-case, and narrative synthesis worked example. Empirical Software Engineering, Accepted for publication, 2014.

[20] Porter, M.E., "Competitive Advantage: Creating and Sustaining Superior Performance". Free Press, New York, 1985.

[21] Huang, L., "A Value-Based Process for Achieving Software Dependability", Proceedings of International Software Process Workshop (2005), Beijing, China. LNCS, Springer Verlag (**2C7)**

[22] Alencar, A. J., “Unleashing the potential impact of nonessential self-contained software units and flexible precedence relations upon the value of software”, Journal of Software, Vol 6, No 12 (2011), 2500-2507, Dec 2011 (**2C8)**

[23] Ereño, M. and Cortázar, R. 2010. Getting the product value with IOR. In Proceedings of the 11th International Conference on Product Focused Software (PROFES '10). ACM, New York, NY, USA, 118-119. (**2C9**)

[24] Wnuk, K., "Visualizing, Analyzing and Managing the Scope of Software Releases in Large-Scale Requirements Engineering" , Doctoral Thesis, October 2013

[25] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., ”Experimentation in Software Engineering, an Introduction”, Springer 2012

[26] Ivarsson, M., Gorschek, T., “A method for evaluating rigor and industrial relevance of technology evaluations”, Empirical Software Engineering, vol. 16 June 2011, pp. 365-395.

[27] Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., & Grünbacher, P. (Eds.). (2006). Value-based software engineering. Springer Science & Business Media.

[28] Cohn, M. (2005). Agile estimating and planning. Pearson Education.

[29] Kazman, R., Cai, Y., Mo, R., Feng, Q., Xiao, L., Haziyev, S., ... & Shapochka, A. A Case Study in Locating the Architectural Roots of Technical Debt. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2015

[30] Racheva, Z., Daneva, M., & Sikkel, K. Value creation by agile projects: methodology or mystery?. In Product-Focused Software Process Improvement (pp. 141-155). Springer Berlin Heidelberg, 2009.

II. Modelling and Code Generation

Chapter 5

Towards creating complete business process models

1. Introduction

Development of the Business Process Model and Notation (BPMN) [18] lies on one of the branches of the Unified Modeling Language (UML) [20] evolution. Similarly as the UML, the BPMN is a semi-formal language but unlike the UML, it is concerned only on the initial stage of software systems development. Its main goal is to present a model of an organization or a company being the domain of application of a future information system. Such models are called business or domain models. In modeling, emphasis is put on processes related to this domain. The notion of a process is the most important. The term “business process” refers to the function (service) performed within the organization. The process is defined as a sequence or flow of activities in an organization with the objective of carrying out work, and is depicted as a graph of flow elements, which are a set of activities, events, gateways, and sequence flows that adhere to finite execution semantics [18].

BPMN models describe private (internal) business processes in an organization (e.g. a company, a company division), and their collaboration with public (external) business processes in the environment of the organization (e.g. a consumer, a seller). The models are presented in a graphical notation, easily understandable by all business stakeholders – business analysts, IT specialists, and organization/company managers [18]. The notation is based on a flowcharting technique similar to the activity diagrams from the UML. A process determines a partially ordered set of business activities that represent the steps required to achieve a business objective. The order results from the flow of control and the flow of data among the activities.

Although BPMN is not declared as a data flow language, in fact, there are two forms of data exchanged between processes and activities: a message flow that depicts the contents of communication and an object flow that depicts a data object reference with its state. BPMN does not itself provide a built-in model for describing the structure of data or a querying language for that data but allows for the co-existence of multiple data structure and querying languages within the same model. Additionally, tool vendors are encouraged to include such languages to their products with commitment to keep compliance with the data modeling defined in the BPMN specification.

BPMN is constrained to support only the concepts of modeling that are applicable to business processes. Therefore, the following aspects are out of the scope of the BPMN specification [18]:

- definition of organizational models and resources,
- modeling of functional breakdowns,
- data and information models,
- modeling of strategy,
- business rules models.

It should be noted that a very important aspect concerning data and its structure is omitted from BPMN specification. In spite of BPMN transition from BPMN 1.0 to 2.0, this claim is still valid [7]. For example, elaboration of the conceptual database model requires information about data types and their relationships. This observation gives rise to the natural idea of integration of BPMN diagrams with these UML diagrams that describe the data structures and methods of their processing. The precise and complete business model plays the fundamental role for the further system development. Especially, it strongly influences on a quality of the final software product. We propose a novel approach to business modeling basing on the compound models, i.e. the BPMN diagrams integrated with the UML class diagrams and UML state machine diagrams. Even though both BPMN and UML models are widely described and researched, the added value of the proposed approach is a result of linking these well-known elements.

Chapter is structured as follows. The next Section gives a review of related works. In Section 3, the definition of the compound BPMN model is presented. Section 4 provides a simple example illustrating the application of the compound BPMN model, and the last Section completes the Chapter with concluding remarks.

2. Related works

The question: "How to build a good model of a business process?" can be used to properly define the context of our consideration (similar questions were stated in [15] and [23]). This question entails two more detailed questions: "What is a good model?" and "Which methodology would be recommended for effective model construction?". Unfortunately, up to now, there have been no satisfying answers to these questions. The conclusion of the paper [27] from 2006 is still valid: there is no well-established modeling standard in this area. A similar conclusion emerges from the current comprehensive overview of the literature on the quality of business modeling [16]: there is a lack of an encompassing and generally accepted definition of business process modeling quality. Having these findings in the background, we concentrate on BPMN only. There may be many issues relating to BPMN, for example: "Whether BPMN is a good enough modeling language?" and "Do the existing tools provide an adequate support for the modeling using BPMN?", etc. In further, we consider some aspects regarding the first issue, however, it should be noted that the assessment of BPMN was considered in some publications, e.g. [1], [3], [21].

BPMN is one of numerous modeling standards (e.g. XPDL, BPEL, BPMN, EPC, and UML Activity Diagrams) developed in last two decades. BPMN seems to be one of the most popular business modeling languages, which does not mean that it is not the object of numerous critics and polemics [3], [9], [21]. It seems that the primary cause of disputes is the lack of a common or, at least, a widely accepted approach for modeling business processes. There are some currently prepared proposals, e.g. [2], [4],

[10], [11], [17], [22], but they all base on specific assumptions regarding a field of application or modeling languages.

Let us remind the reader that a business model is supposed to express intuitive ideas, thus supporting communication among users, and thus delivering information necessary to specify the requirements for the future software system. Therefore, a modeling language should have sufficient expression power enabling the presentation of all interesting structural and behavioral features from the domain of interest. Additionally, the language should have a satisfactory level of formality that will allow to check consistency and completeness of a model expressed in this language.

Has the BPMN enough expression power? At the beginning, it should be noted that BPMN enables only partial description of the domain of interest. Namely, BPMN concentrates on a specification of business participants and the types of processes performed, i.e. the types of mutually offered services. But even in this scope we observe that BPMN models seem to suffer from incompleteness. It results from the fact that the BPMN puts stress on the description the structures of processes with skipping details of the processed data objects.

3. A compound model of a process

BPMN 2.0 specification [18] defines the so called item-aware elements that may represent both physical or information items, and which are stored and conveyed during process execution. Data objects, representing information items, are specialization of item-aware elements. There are four elements that represent data: data objects, data inputs, data outputs and data stores. Data objects are identified by names but the specification does not define their internal structure. Data inputs and outputs are used to show data that are produced during or as a result of execution of processes. Data stores provide mechanisms to retrieve or update stored information.

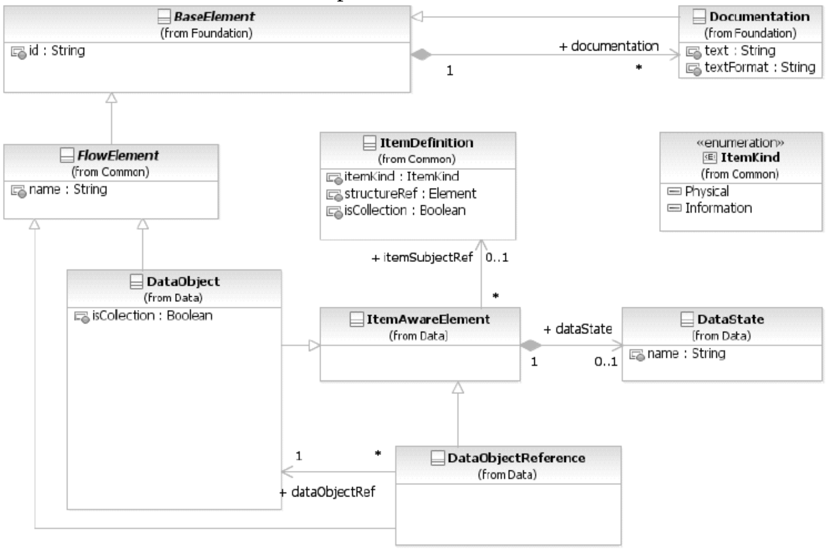


Figure 1. DataObject class diagram according to [18]

Data objects in BPMN are defined by *DataObject* class from the specification presented in fig. 1 [18]. The *DataObject* inherits the attributes and model associations of *FlowElement* and *ItemAwareElement* classes. *DataObjectReference* class is used to reuse data objects in the same model. The *DataObject* can optionally reference *DataState* class, which is used to specify different states of the same *DataObject* at different points in a process. BPMN 2.0 specification does not define the states, e.g. the possible values and semantic, but it introduces an extensibility mechanism that allows extending the standard. Therefore using the state element and the BPMN extensibility capabilities one can define the object states more precisely.

Following [5], modeling business processes without modeling the processed objects would be rather poor. Therefore, it seems to be beneficial to create compound models of processes that would take into account all the details regarding processed data. To fulfill this postulate, UML class diagrams can be incorporated into the compound model. In this way some data objects represented on a process diagram will have references in the class diagram. More precisely, more information is carried if a data object on the process diagram has an instance of a respective class on the class diagram. Moreover, data objects may change their states during the execution of a process. Usually, these changes are subjected to some constraints. These constraints can be clearly presented by UML state machine diagrams.

Finally, the proposed compound model of a BPMN process $\mathbf{CM}_{\mathrm{BPMN}}$ consists of a set of three types of diagrams: a process diagram, a class diagram, and a state machine diagram:

$\mathbf{CM}_{\mathrm{BPMN}} = <\mathbf{PD}_{\mathrm{BPMN}}, \mathbf{CD}_{\mathrm{UML}}, \mathbf{SMD}_{\mathrm{UML}}>$

where:

- $\mathbf{PD}_{\mathrm{BPMN}}$ is a set of BPMN 2.0 Process Diagrams which illustrate a needed business process,
- $\mathbf{CD}_{\mathrm{UML}}$ is a set of UML Class Diagrams whose role is to describe the structure of a system by showing the classes with attributes and operations, and the relationships between the classes,
- $\mathbf{SMD}_{\mathrm{UML}}$ is a set of UML State Machine Diagrams which are aimed for a given class to describe transitions between the states of its objects together with the events that trigger transitions between the states.

The figure 2 presenting relationships between process diagrams, class and state machine diagrams, components of the compound model, looks like a metamodel of the compound model. However, formally it cannot be treated as a metamodel because the metaclasses: *BPMNProcessDiagram, UMLClassDiagram* and *UMLStateMachine-Diagram* are not formally defined in BPMN or UML specifications. These specifications define only components of diagrams. For example, structural constructs (e.g. classes, components) used in the $\mathbf{CD}_{\mathrm{UML}}$ are defined in the *Classes* package in “Subpart I - Structure” section of the UML Superstructure specification [20]. Similarly, “Subpart II - Behavior” section in [20] specifies the dynamic behavioral constructs, e.g. state machines used in $\mathbf{SMD}_{\mathrm{UML}}$.

The compound model $\mathbf{CM}_{\mathrm{BPMN}}$ consists of $\mathbf{PD}_{\mathrm{BPMN}}$, $\mathbf{CD}_{\mathrm{UML}}$ and $\mathbf{SMD}_{\mathrm{UML}}$ diagrams that are interrelated in a way shown in fig. 2. The compound model **CM** should always have at least one $\mathbf{PD}_{\mathrm{BPMN}}$ diagram and any number of related $\mathbf{CD}_{\mathrm{UML}}$ diagrams. In practice, only in trivial cases will the compound model not contain any class diagrams or state machine diagrams. The metaclass *CompoundModel* is not navigable from metaclasses *BPMNProcessDiagram*, *UMLClassDiagram* and *UMLStateMachine-Diagram*. *BPMNProcessDiagram* and *UMLClassDiagram* metaclasses are associated with each other as they together complement the description of the process. *BPMNProcessDiagram* metaclass derives states of data objects so it is associated with *UMLStateMachineDiagram*. *UMLStateMachineDiagram* metaclass is associated with *BPMNProcessDiagram* as it shows more information about some of its data objects. Finally, *UMLClassDiagram* is associated with zero or more *UMLStateMachineDiagram* metaclasses by providing more details about the described objects. Similarly, each *UMLStateMachineDiagram* metaclass is associated with zero or more *UMLClass-Diagram* metaclasses by showing states and transitions of the objects.

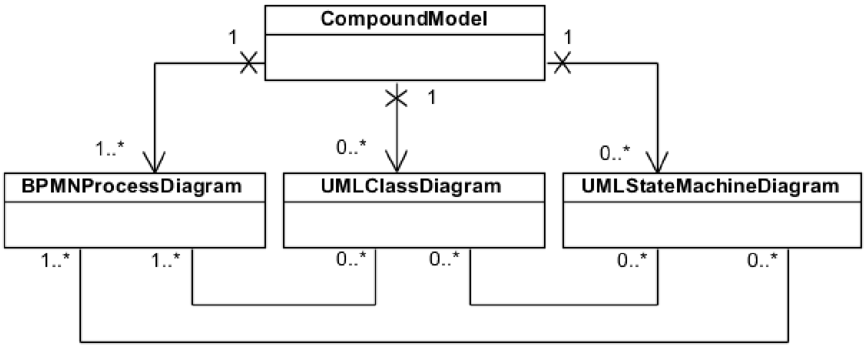


Figure 2. The structure of the compound model.

4. Example of a compound model

The following example is aimed to depict the idea of a compound model. It presents a business process of ordering and manufacturing windows (please note, “windows” here are understood as “window products”, the openings in walls). More specifically, the model illustrates all important stages of the production process, from the acceptance of an offer for windows, through the completion of components and raw materials and finally, shipment of the windows to the customer. An example BPMN process diagram (fig. 3) has many data objects. Three of them − an offer, an order and a contract − have also states that depict how the objects are updated within the process. The presented compound model is composed of the following diagrams: a process diagram (fig. 3), a complementary class diagram (fig. 4) and three derived state machine diagrams (fig. 5).

The BPMN process diagram (fig. 3) has three lanes for three parties involved in the process: a customer, a point of sale and a manufacturer. Due to the fact that we want to model the interaction between the parts explicitly, they have been classified as participants. In order to make the example easily readable, some simplifications of the real process have been adopted. First of all, the process describes an internal point of sale of the manufacturer and its interaction with the customer. It often happens in reality that manufactures apart from the internal points of sale also have a distributed network of the external dealers, who sell final products to customers. To support this situation, the process should be extended by additional communication and the accounting of payments using, for example, trade credits and individual discounts for different dealers. This would require introducing additional data objects and additional state machine diagrams related to the BPMN process model. Another simplified element of the diagram is a process of gathering information about all windows that should be produced, e.g. their dimensions, features and additional requirements. In many cases, dealers measure windows by themselves and the role of the customer is reduced to only making it possible to conduct all the necessary measurements on a construction site. A simplification was applied also in the process of gathering and selecting the production orders. The diagram presents only a general view on the production orders and the further steps leading to a final product. The production of windows is presented as a sub-process. The decision how to organize, optimize and select production orders before sending them to the production and the organization of the production are beyond the scope of the diagram.

These and other simplifications were adopted in order to improve the overall understanding of the BPMN model with the aim of not changing the real process much. In practice, every window manufacturer produces and organizes the production and sale differently because there are many possible solutions depending on many internal and external factors. The main reasons for different organizations are: the available machinery park, the organization of the sales, choice of the profile materials (PVC, aluminum or wood), historical decisions, the region and habits of employees.

Some data objects introduced in the BPMN process diagram also appear in the UML class diagram, which provides the additional information. Let us consider an offer. The data object *Offer* from fig. 3 has its counterpart class named *Offer* in fig. 4. The class has not only the internal attributes and operations but also the associations to other classes in the diagram. Class associations, which can be adorned with role names, ownership indicators, multiplicity, visibility, and other properties, significantly increase the amount of information available for the data objects which are specified just by a name with an optional status on BPMN process diagrams. Moreover, BPMN process diagrams describe services in terms of sequence of processes and messages that flow between the participants in different activities. In the example, a customer and a point of sale pools exchange message flows and the *Offer* data object is associated with tasks and sub-process within the pools. More details are provided on a class diagram, where we can see that many sellers may work in a concrete point of sale (*Seller* class has an attribute *worksInPointOfSale*), but a concrete *Offer* will be prepared by one specific *Seller*. These additional information may later be used to design better user interfaces. It also has a pragmatic aspect as the compound model can be easier understood than the standalone BPMN process diagram.

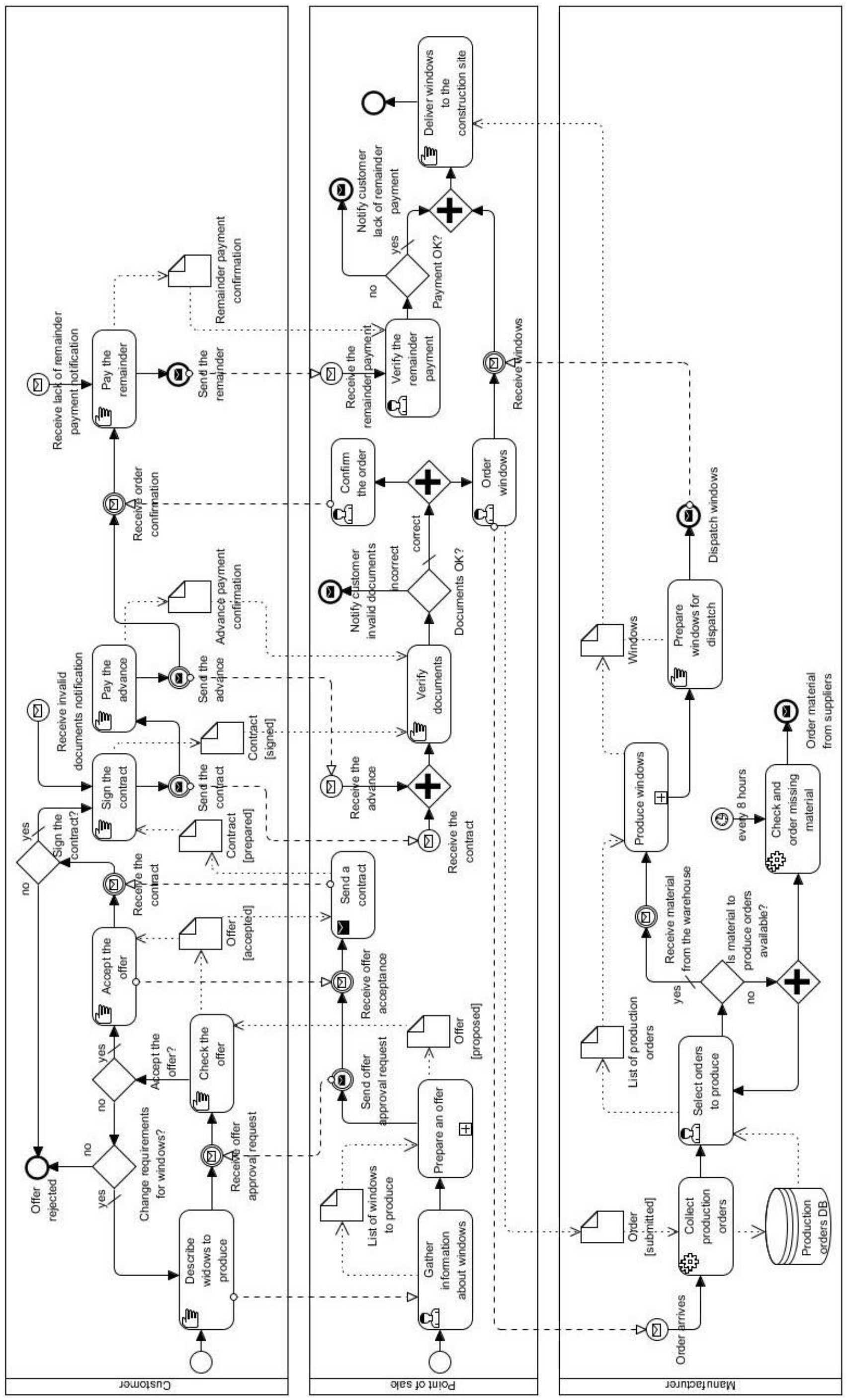

Customer
Point of sale
Manufacturer
Describe widows to produce
Receive offer approval request
Check the offer
Accept the offer?
yes
no
Change requirements for windows?
Offer rejected
Accept the offer
Receive the contract
Sign the contract?
Sign the contract
Send the contract
Receive invalid documents notification
Pay the advance
Send the advance
Receive order confirmation
Receive lack of remainder payment notification
Pay the remainder
Send the remainder
Offer [accepted]
Contract [prepared]
Contract [signed]
Advance payment confirmation
Remainder payment confirmation
Gather information about windows
List of windows to produce
Prepare an offer
Offer [proposed]
Send offer approval request
Receive offer acceptance
Send a contract
Receive the contract
Receive the advance
Verify documents
Documents OK?
correct
incorrect
Notify customer invalid documents
Confirm the order
Order windows
Receive the remainder payment
Verify the remainder payment
Payment OK?
Notify customer lack of remainder payment
Receive windows
Deliver windows to the construction site
Order arrives
Order [submitted]
Collect production orders
Production orders DB
Select orders to produce
List of production orders
Is material to produce orders available?
Receive material from the warehouse
every 8 hours
Check and order missing material
Order material from suppliers
Produce windows
Windows
Prepare windows for dispatch
Dispatch windows

Figure 3. Example of the compound model: a BPMN process diagram.

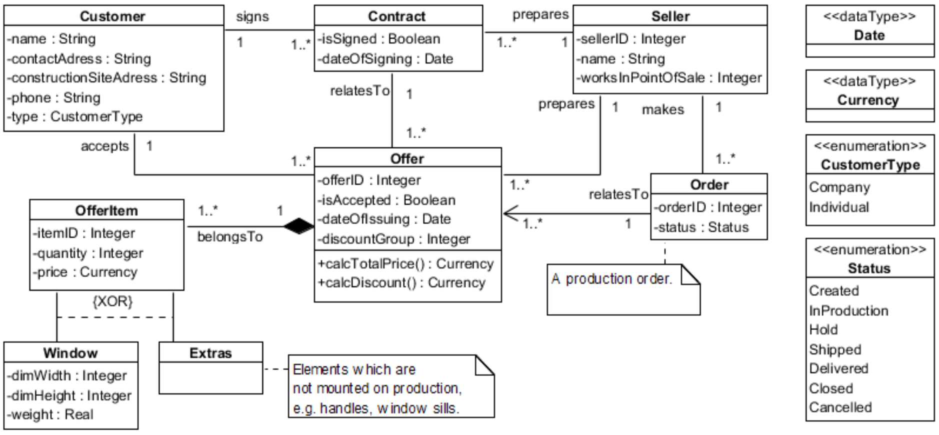


Figure 4. Example of the compound model: a UML class diagram.

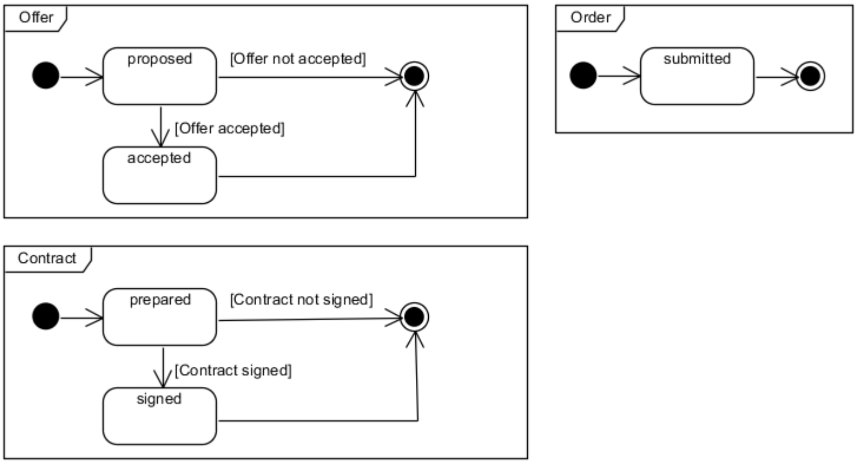


Figure 5. Example of the compound model: UML state machine diagrams.

5. Conclusions

The first part of the Chapter introduces an approach to business modeling that is based on the integration of BPMN process diagrams with UML class diagrams and UML state machine diagrams. The diagrams are interrelated and together constitute the compound model. The second part of the Chapter presents an example of the compound model with the explanation of the benefits of using the described approach.

The proposed compound BPMN process diagram and the illustrating example give rise to the question: “How to build such compound diagrams?”. As it is clear from the review of the literature, there is no commonly accepted way of business modeling. The main causes are a huge variety of scope and way of presenting of information, which are the basis for business modeling.

Usually the information is provided in the form of written or electronic documents in a variety of ways associated with the described area. On the one hand, these documents may contain information irrelevant to the modeled areas, on the other hand they may not contain all the necessary information. Therefore, usually at the beginning of the modeling an overview of important concepts is done. This overview forms a glossary of terms representing these concepts. Before further modeling steps all terms from the glossary should be validated. Next, what is often applied in practice, first a class diagram and then a process diagram are created. The class diagram represents the concepts from the glossary with the relationships among them. This justifies why class diagrams are proposed for compound BPMN process models. In the alternative proceeding, first a process diagram and then a class diagram are created. Both sequences of diagram derivations do justify the usefulness of the class diagrams in the proposed compound BPMN process models. Attaching state machine diagrams to the model is now a natural consequence of the presence of class diagrams.

It should be underlined that the compound models, contrary to the usual BPMN models, comprise both entirely static and dynamic aspects of the business domain.

It is also worth noting that the compound model could be particularly justified when we have a specific domain ontology providing basic information about the modeled domain. All ontologies take into account the static aspect and only a few additionally take into account the dynamic aspect. In this situation, creation of a class diagram as the first one is strongly justified.

The outlined approach to business modeling based on the proposed compound model requires further research. The first practical task seems to be developing a tool supporting edition and controlling integrity of the compound model. Optionally, it would be expected to have a tool supporting the verification of the correctness of the compound models against the domain with the use of the domain ontologies. Similarly, in [6] it is suggested that knowledge from the ontologies should be transferred to the software applications. More precisely, it is recommended that a method and a tool for automatic transformation of selected fragments of domain ontology directly to models should be developed.

References

[1] G. Aagesen, and J. Krogstie, Analysis and design of business processes using BPMN. *Handbook on Business Process Management 1*. Springer Berlin Heidelberg, 2010. 213-235.

[2] T. Allweyer, Human-Readable BPMN Diagrams, in: [7], p. 217-232.

[3] E. Börger, Approaches to model business processes: a critical analysis of BPMN, workflow patterns and YAWL, *Software Systems Modeling*, vol. 11, 305-318, 2012.

[4] M. Cortes-Cornax et.al., Using intensional fragments to bridge the gap between organizational and intensional levels, *Information and Software Technology*, vol. 58, 1-19, 2015.

[5] S. Drejewicz, *Zrozumieć BPMN modelowanie procesów biznesowych*, Wydawnictwo Helion, 2012.

[6] I. Dubielewicz, B. Hnatkowska, Z. Huzar, and L. Tuzinkiewicz, Domain Modeling in the Context of Ontology. *Foundations of Computing and Decision Sciences*, *40*(1), 3-15, 2015.

[7] L. Fischer (ed.), *BPMN 2.0 Handbook. Methods, Concepts, Case Studies and Standards in Business Process Modeling Notation*, Future Strategies Inc., 2012

[8] J. Freund, M. Schrepfer, Best Practice Guidelines for BPMN 2.0, in: [7], p. 203-215.

[9] D. Gagné, Addressing some BPMN 2.0 misconceptions, fallacies, errors, or simply bad practices, in: [7], p. 113-124.

[10] F. Heidari, P. Loucopoulos, Quality evaluation framework (QEF): Modeling and evaluating quality of business processes, *International Journal of Accounting Information Systems*, vol. 15, 193-223, 2014.

[11] J. Kotremba, S. Raβ, and R. Singer, Distributed Business Process – A Framework for Modeling and Execution, arXiv:1309.312v2 [csMA], 18 May 2014.

[12] R. Laue, A. Awad, Visual suggestions for improvements in business process diagrams, *Journal of Visual Languages and Computing*, vol. 22, 385-399, 2010.

[13] O.I. Lindland, G. Sindre, and A. Sølvberg, Understanding quality in conceptual modeling, *Software IEEE*, Volume: 11, Issue: 2, 42-49, 1994.

[14] A. Marat and J. M. Gómez, "Derivation of Event-Based State Machines from Business Processes," *Int. Conf. New Trends Inf. Commun. Technol. Almaty Kazakhstan*, 2014.

[15] P. Mohagheghi, V. Dehlen, and T. Neple, Definitions and Approaches to Model Quality in Model-based Software Development - A Review of Literature, *Information and Software Technology*, 1646-1669, 2009.

[16] I. Moreno-Montes de Oca et. al., A systematic literature review of studies on business process modeling quality, *Information and Software Technology,* vol. 58, 187-205, 2015.

[17] G. Navarro-Suarez, J. Freund, and M. Schrepfer, Best Practice Guidelines for BPMN 2.0, in: *BPMN 2.0 Handbook First Edition*, 1st ed., Future Strategies Inc., 2010, pp. 151–165.

[18] OMG, Business Process Model and Notation (BPMN), Version 2.0, 2011.

[19] OMG, Business Process Definition MetaModel, Volume I: Common Infrastructure, Volume II: Process Definitions: http://www.omg.org/spec/BPDM/1.0

[20] OMG, Unified Modeling Language,Version 2.5, Doc. No.: ptc/2013-09-05, http://www.omg.org/spec/UML/2.5

[21] W. Reisig, Remarks on Egon Börger: Approaches to model business processes: a critical analysis of BPMN, workflow patterns and YAWL, *Software Systems Modeling*, vol. 12, 5-9, 2013.

[22] R. M. Pillat et. al., BPMNt: A BPMN extension for specifying software process tailoring, *Information and Software Technology,* vol. 57, 95-115, 2015.

[23] J. Pitschke, Business Vocabulary, Business Rules and Business Process – How to Develop an Integrated Business Model?, Presentation at the Business Rules Forum 2010, Washington, DC.

[24] M. Sadowska, An approach to assessing the quality of business process models expressed in BPMN, *e-Informatica Software Engineering Journal* (accepted for publication), 2015.

[25] R.M. Shapiro, Reference Guide XPDL 2.2: Incorporating BPMN 2.0 Process Modeling Extensions, in: [7], 267-279.

[26] B. Silver, Elements of BPMN Style: Leyel 2, in BPMN method and style, 2nd ed., Aptos: Cody-Cassidy Press, 2009.

[27] W. Wang, A Comparison of Business Process Modeling Methods, *2006 IEEE International Conference on Service Operations and Logistics, and Informatics*, 1136-1141, IEEE, 2006.

Chapter 6

Towards automatic Sumo to UML translation

1. Introduction

Domain model is a key development artifact. It captures the most important types of objects in the context of the domain, i.e. the entities that exist or events that transpire in the environment in which the system works [1, 2]. Domain models, beside a glossary, and business object model, are used to document the domain in which the system executes. It serves as a formalization of the glossary [1]. Domain model could be represented with the use of different notations, among which the most popular are Entity Relationships Diagrams, and UML class diagrams.

Domain models should be of high quality to reduce the number of changes when the development proceeds. Among quality factors the most important are [3]: consistency, completeness, and correctness (3C).

Consistency and completeness could be perceived from 2 perspectives: external, and internal, from which the external is more difficult to achieve. *External completeness* means that we identified in the domain all important entities and relationships, while *external consistency* means that we documented the identified elements in the way that preserves their semantics [4]. On the other side, domain model is *internally consistent* when it contains no contradictions, and it is *internally complete* when it doesn't contain any undefined object, and no information is left unstated or to be determined [2].

Definition of model correctness is much vaguer. Some authors define it as a mixture of consistency and completeness [3, 4], some others [4] – refer it to syntactic correctness (that meaning of correctness is used further in the paper).

Domain model is typically elaborated by a business analyst during business modeling or requirement specification phase [2]. Different elicitation techniques serve to discover the entities in the domain. However, the obtained results strongly depend on the complexity of the domain, business analyst experience, and the quality of information sources. More difficult domain, less experience analyst or poor quality sources, more likely worse quality of the resulted domain model.

On the other side, domain knowledge is often included in existing ontologies, and could be extracted from them. The extraction process could be (partially) automated, resulting in a high quality domain model. Consistency and correctness of that model could be guaranteed by construction, assuming that the source (ontology) itself is correct and consistent with a domain. The model completeness, at least internal, could be also checked.

There are many high-level ontologies currently developed, e g. BFO, Cyc, GFO, SUMO. The last one, SUMO, seems to be very promising because it became the basis for the development of many specific domain ontologies. A particular useful feature is that the notions of SUMO have formal definitions (in SUO-KIF language) and at the same time are mapped to the WordNet lexicon [5]. SUO-KIF is a variant of KIF (Knowledge Interchange Format) language [6]. Knowledge is described declaratively as objects, functions, relations, and rules. SUMO and related ontologies form the largest formal public ontology in existence today [5, 6]. What is more, the ontologies that extend SUMO are available under GNU General Public License.

The paper presents an initial version of a tool to automatic SUMO to UML translation. The tool is thought as a support for business analyst collaborating with business experts. The main functionalities include: browsing ontology content, selection of interesting elements, and translation of selected elements to UML class diagram. The presentation covers a meta-model of SUMO notions (the main input to transformation process), the tool architecture, and an example of domain model that results from the tool application. The genesis of the tool (related works) is also shortly described as well as the problems met during implementation, and the elements that will be included in next release.

The only tool available in the Internet that supports SUMO is SUMO browser, called Sigma [7]. Tools that allow creating a UML class diagram from existing ontology exist for other formalisms, e.g. OWL [8], but such a tool is unavailable for SUO-KIF. However, SUO-KIF could be translated to other formalisms, e.g. DLP [9].

SUMO was selected from existing ontologies because of the following reasons:

- It constitutes the biggest set of ontologies which is freely available; SUMO contains definitions of more than 21 thousands of terms, and more than 70 thousands of axioms; moreover, the mapping of SUMO notion to WordNet is also available [6];
- SUO-KIF language is very flexible; it allows to handle relations among three or more things directly (e.g. OWL does not); it supports statements and rules written not only in First-Order Logic, but also (at least partially), in the Higher-Order Logic (e.g. "(*believes John* (*likes Bob Sue*))", when the second argument of "believes" is a proposition) [6];
- Existing translation of SUMO to OWL is a provisional and necessarily lossy [6], what put in question its usefulness; on the other hand it is possible to perform the opposite translation from OWL to SUMO, what seems more promising, because the result could be extended with the usage of SUO-KIF features;
- The flexibility of SUO-KIF is very similar to SBVR standard [18], promoted by OMG, defining the meta-model for representation of business vocabulary, and business rules; SBVR statements could be directly translated either to SUO-KIF or to UML.

UML was selected as the target language for the translation because it is a general purpose modeling and specification language, commonly used not only by programmers, but also by business analysts. Besides Entity Relationship Diagram it is the often selected notation to describe domain models. Together with OCL it forms a very useful tandem to define constraints on the domain behavior in the formal way. UML class diagram could be easily translated to other representations, either more business oriented like SBVR (e.g. [18]) or more program oriented like java, c#, SQL (e.g. [20]).

2. Related works

The paper [10] is the first in a series considering SUMO ontology as a source for domain modeling. It presents an initial set of mapping rules between SUMO notions and UML notions, and identifies the elements difficult to extract, e.g. attributes in SUMO are defined for instances, typically within if-then rules, not for classes, as it is in UML.

The paper [11] presents an outline of a systematic approach to the development of domain model on the basis of selected SUMO ontologies. The approach involves only a few steps. It starts with needs description, next goes through identification of business processes in the area of interests that help to decide if a notion within an ontology is in the area of interests (and should be translated to UML) or not. After analysis of selected elements, they are translated (manually) to a UML class diagram. The approach was checked by example. Some SUMO-UML mappings were also refined. The main problems the authors claim about are: (a) ontology size – it contains many irrelevant (out of interesting scope) elements; (b) domain knowledge is spread over many ontologies (files); (c) some facts are defined at very general level (predicates between *Object*, *Physical*) what makes the interpretation more difficult.

In the paper [12] the refined version of the approach from [11] is presented. The approach also consists of only few steps, but their definition is much more formal and close to implementation needs. The main idea of the approach is a guided selection of SUMO extract, which will be farther translated to UML. The paper also proposes some new transformation rules, e.g. transformation of unary functions. The general finding of that work is that the process of knowledge extraction must be supported by a tool. Otherwise the process, even if the results are promising, is very time consuming, and error prone.

The contributions of this paper are as follows:

- Meta-model of SUMO notions used within a transformation process (see the subchapter 3).
- Definition of transformation tool architecture, and a set of static-consistency rules designed and implemented to check the internal consistency and completeness of SUMO ontology (see the subchapter 4).
- Verification and correction of transformation rules defined in [10-12]; the subset of implemented rules (including the changed ones) is presented in the subchapter 5.

3. Meta-model of SUMO notions

To support SUMO to UML transformation process the content of SUO-KIF files has to be represented at the higher abstraction level, that enables both: to check static consistency rules, and to perform the transformation process itself. This is achieved with so called meta-model of SUMO notions – see Fig. 1.

The diagram reflects physical structure of SUO-KIF file which can be perceived as a set of sentences. A SUMO sentence is represented by *Sentence* abstract class – a parent of all possible kinds of statements in SUMO. Each sentence belongs to exactly one *OntologySegment* (SUO-KIF file). Below there is a short description of concrete sentence classes:

a) *LogicalSentence* – a sentence starting with a logical operator, e.g. (=> …), (<=> …)
b) *QuantifiedSentence* – a sentence starting with quantifier: (*forall* …) or (*exists* …)
c) *RelationalSentence* – a sentence starting with a name of function or relation: (*name* ….).

It is assumed that only sentences written at the first level are instantiated by SUMO to UML translator, e.g. the text: "(=> (*instance ?REL BinaryPredicate*) (*valence ?REL 2*))" will be instantiated as one sentence even if it contains 2 internal sub-sentences. SUMO comments are omitted by the parser.

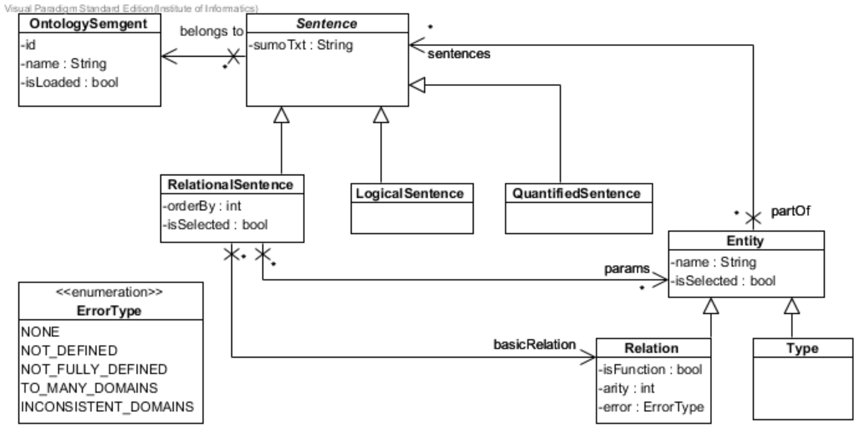


Figure 1. Meta-model of SUMO notions – main elements.

The right side of the class diagram shows the structure of SUMO notions. *Entity* is "the root node of the ontology" [5]. *Entity* is associated with all sentences which it is a part of.

Entity is the parent for two UML classes interesting in the context of considered transformation:

a) *Relation* – definition of SUMO relation or function, together with its domain/range (see Fig. 2),
b) *Type* – represents SUMO notions that can be instantiated, e.g. *BinaryPredicate*.

Each instance of *RelationalSentence* is linked to one *Relation* (*basicRelation* role), and many *Entities* involved (*params* role).

Some specific relational sentences (defined in SUMO upper ontology) play crucial role in the transformation process. Up to now six types of such sentences were identified:

a) Documentation sentence (*DocumentationSent*) – a sentence starting with "(*documentation* …)"; contains documentation (an instance of *SymbolicString*) in a specific language for a specific entity;
b) Instance sentence (*InstanceSent*) – a sentence starting with "(*instance* …)"; is associated with an entity (instance), and a type for that instance;
c) Subclass sentence (*SubclassSent*) – a sentence starting with "(*subclass* …)"; used to describe inheritance hierarchy between SUMO classes; is associated with parent and child types;
d) Subrelation sentence (*SubrelationSent*) – a sentence starting with "(*subrelation* …)"; allows to describe inheritance hierarchy between SUMO relations; is associated with parent and child relations;
e) Domain sentence (*DomainSent*) – a sentence starting either with "(*domain* …)" or "(*domainSubclass* …)"; represents domain element (*Type*) for a specific relation;
f) Range sentence (*RangeSent*) – a sentence starting either with "(*range* …)" or "(*rangeSubclass* …)"; represents a range (*Type*) for a function (*Relation* with *isFunction* attribute set to true).

In the future, the list will be extended to represent for example *partition or part* relations.

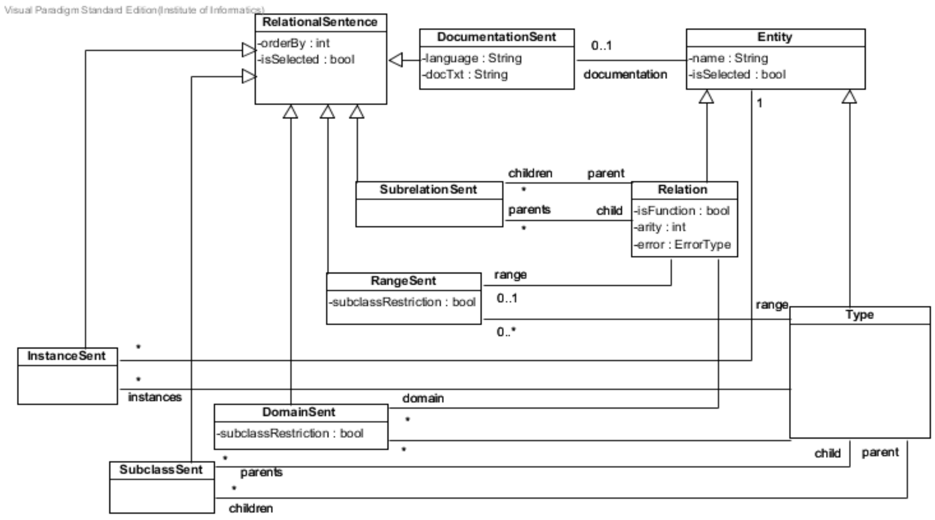


Figure 2. Meta-model of SUMO notions – hierarchy of relational sentences.

4. Architecture of SUMO to UML translator

SUMO to UML translator is implemented in java 8 with Swing library. The main functional elements of the translator are presented on a component diagram in see Fig. 3.

End-user is allowed to select any subset of ontology SUO-KIF files (called ontology segments) to be loaded by the tool. The loading process is controlled by

SumoLoadConttroller component, and is presented – with the use of a sequence diagram – in Fig. 4.

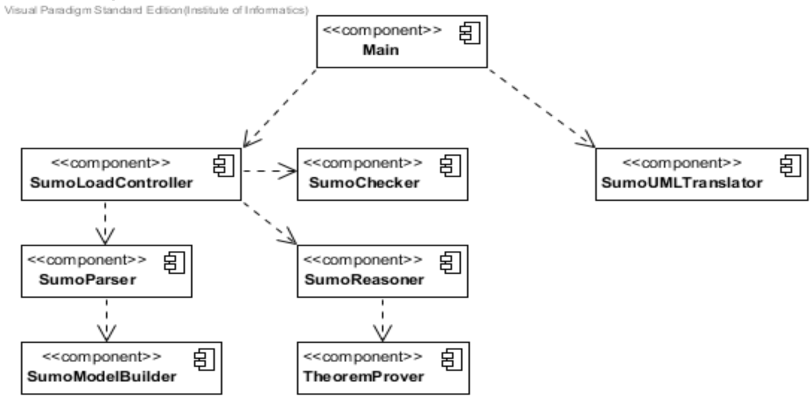

Visual Paradigm Standard Edition(Institute of Informatics)
<<component>> Main
<<component>> SumoLoadController
<<component>> SumoChecker
<<component>> SumoUMLTranslator
<<component>> SumoParser
<<component>> SumoReasoner
<<component>> SumoModelBuilder
<<component>> TheoremProver

Figure 3. Architecture (functional view) of SUMO to UML translator.

SumoLoadController runs *SumoParser* to: (a) check the syntax correctness of the file, (b) walk through all tokens in the file and to call *SumoModelBuilder* to translate SUMO sentences into internal SUMO model representation. *SumoParser* was generated by antlr [13] on the basis of SUO-KIF context-free grammar [14].

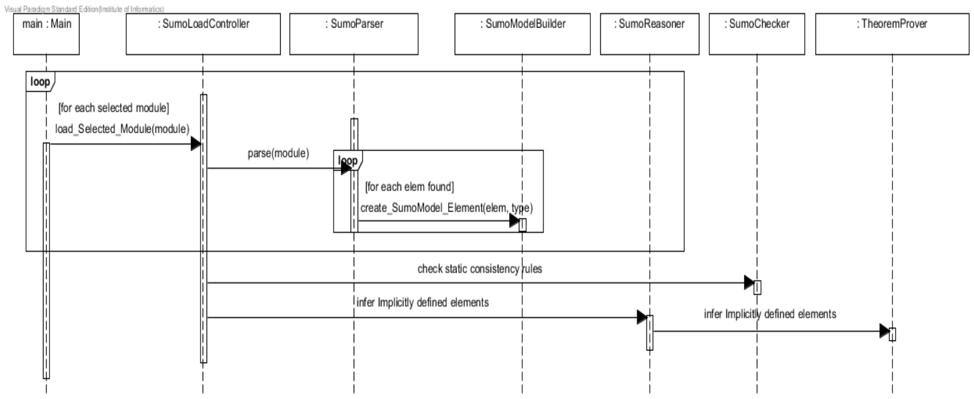

Visual Paradigm Standard Edition(Institute of Informatics)
main : Main
: SumoLoadController
: SumoParser
: SumoModelBuilder
: SumoReasoner
: SumoChecker
: TheoremProver
loop
[for each selected module]
load_Selected_Module(module)
parse(module)
loop
[for each elem found]
create_SumoModel_Element(elem, type)
check static consistency rules
infer Implicitly defined elements
infer Implicitly defined elements

Figure 4. Processing of ontology segment.

Unfortunately, it appeared that SUMO ontology suffers from many bugs, that can't be found by the parser (according to the rules formulated in context-free grammar). The bugs could negatively influence the correctness of the intended transformation process. Below you have representative examples of them (state on the 12th of April, 2015):

a) lack of domain definition for some relations: e.g. *adjacentOrientation*

b) overriding of domain definition:
(*domain emotionTendency 1 Agent*),
(*domain emotionTendency 1 EmotionalState*)

c) inconsistency between predicate instance and predicate domain:
(*domain 2 rateDetail Formula*) instead of (*domain rateDetail 2 Formula*)

d) inconsistency between subrelation domain and its parent domain:
(*subrelation ingredient material*)
(*domainSubclass ingredient 2 PreparedFood*)
(*domain material 2 CorpuscularObject*)
e) internal inconsistency between different statements:
(*instance visitorParameter BinaryPredicate),*
(*domain visitorParameter 3 WebSite*) – the 3rd domain can't be specified for binary predicates

So, there was a strong need to implement *SumoChecker* component, which main functionality is to perform different consistency checks. The elements with bugs are marked and reported by the tool, so the user has an opportunity to correct the input.
Up to now, 3 consistency rules were implemented:

a) Rule for checking domains for predicates and functions – checks if the number of domain definitions is consistent with the type of predicate/function, e.g. any instance of *BinaryPredicate* should have 2 domain definitions;
b) Rule for checking subrelation domains – checks if subrelation domains are consistent with domains of subrelation parents;
c) Rule for checking consistency between relation arguments and relation domains – for relation instance like "(*instance Aruba LandArea*)" the rule checks if arguments (*Aruba*, *LandArea*) are instances of appropriate domains (*Entity*, *SetOrClass*).

All bugs found by *RuleChecker* for different ontology segments are successively reported to Mr. Adam Pease, the author of SUMO, and they are gradually corrected by him and his co-workers.

The process of consistency checking could be switched off in the future, when the bugs will be fixed, and switched on only on demand, for new ontology segments.

As was mentioned in the previous chapter, domain knowledge is spread over different SUO-KIF files what is not very convenient for transformation. That is why a separated component – *SumoReasoner* – was introduced. Its main responsibility is to update previously generated SUMO model by inferring information indirectly defined in SUMO, e.g.: a subrelation could inherit domain definition from its parents; in such case *SumoReasoner* copies domain from the parent to all its children.

It is also planned (that feature has not been implemented yet), that *SumoReasoner* will communicate with selected theorem prover to reason knowledge from the rules, e.g. about class attributes.

The new version of sigma tool [7] is prepared to collaborate with E prover [15]. E prover can deliver answers for specifically marked conjecture formulas. Sigma has implemented mapping rules between SUO-KIF and TPTP formalism used by E prover. In consequence, a user can formulate questions like: (*instance ?X BinaryPredicate*) to find out all instances of *BinaryPredicate*.

The transformation process is realized by *SumoUMLTranslator* component. It produces – with the use of eclipse.emf and eclipse.uml2 frameworks, an instance of UML model, and stores it in a file (*.uml), that can be read in a form of a tree or can be visualized on a diagram with additional tools, like e.g. Papyrus [16].

5. Examples of transformation rules

This subchapter shortly presents the implemented transformation rules focusing on those that were changed in reference to the previous publications [10-12].

Selected transformation rules are presented in table 1.

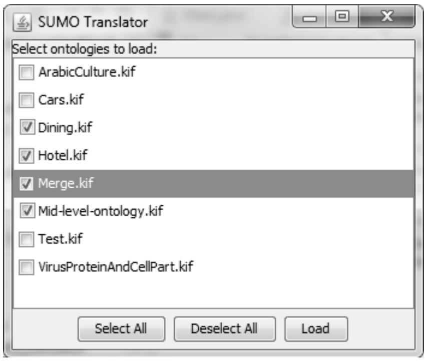
Table 1. The subset of SUMO-UML transformation rules

SUMO element	UML element	Comment
Direct or indirect subclass of *Entity*, e.g. *Agent, Reservation*, *Integer*	Class	Data values like Integers are also represented as separate classes (what results in uniform representation of relations)
Binary (including self) and higher arity relations with all domains defined in the form "(*domain relation class*)", e.g. "(*domain customer 1 CognitiveAgent*), (*domain customer 2 CognitiveAgent*)"	Association, e.g. *customer*, *numberOccupant*	Previously, when one of domains in relation was a data value, e.g. Integer, the relation was represented either as an attribute (for binary relation) or an association class; now, all of binary or higher arity relations are represented in the same way, as associations
Relation domain defined in the form "(*domainSubclass relation int class*)", e.g. "(*domainSubclass roomAmenity 1 HotelUnit*), (*domainSubclass roomAmenity 2 Physical*)"	Generalization class, e.g. *Physical_Subclasses*, *HotelUnit_Subclasses*	*domainSublcass* is a constraint meaning that the int'th element of each tuple in relation must be a subclass of a specific class; that notion is represented by UML generalization set
Binary (including self) and higher arity relations for which at least one domain is defined in the form "(*domainSublcass relation int class*)", e.g. "(*domainSublcass roomAmenity 1 HotelUnit*), (*domainSubclass roomAmenity 2 Physical*)"	Association among the results of translations of relation domains including generalization sets, e.g. *roomAmenity* (association between *Physical_Subclasses* and *HotelUnit_Subclasses*)	The previous transformation was incorrect (misinterpreted semantics); the association used to link classes; the new association links generalization sets
Subrelation relationship when the parent relation has one or more its domains defined in the form "(*domainSublcass* ...)", e.g. "(*subrelation paidRoomAmenity roomAmenity*)"	Association with "subsetted" property, e.g. association end of *paidRoomAmenity* will be a subset of association end of *paidAmenity*	*subrelation* is a constraint meaning that every tuple of a child relation is also a tuple of a parent relation; in the UML 2.5 such a feature is represented by a subset constraint

6. Usage example

To demonstrate the functionality of SUMO to UML translator the example, described in [12], is reused. It aims in elaborating an initial domain diagram based on Hotel domain ontology, and ontologies it is based upon (e.g. Merge.kif, Mid-level-ontology.kif, Dining.kif) [5].

Fig. 5 shows the initial form which allows a user to select interesting ontologies (ontology segments).



Select ontologies to load:

- [ ] ArabicCulture.kif
- [ ] Cars.kif
- [✓] Dining.kif
- [✓] Hotel.kif
- [✓] Merge.kif
- [✓] Mid-level-ontology.kif
- [ ] Test.kif
- [ ] VirusProteinAndCellPart.kif

Figure 5. SUMO to UML translator – the initial form.

After file loading *SumoChecker* component reports found bugs. Examples of such bugs are presented below:

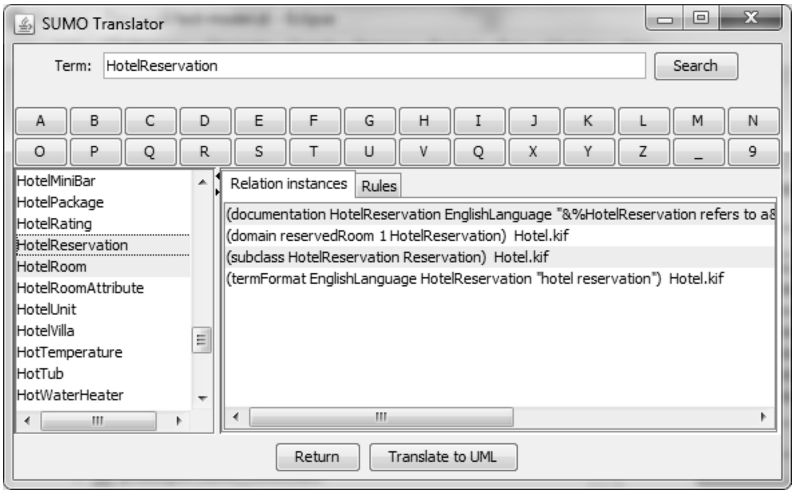
reservingEntity lacks of meta-data

powerPlant 1 domain - Device - doesn't fit parent - component - domain CorpuscularObject

powerPlant 2 domain - Artifact - doesn't fit parent - component - domain CorpuscularObject

SUMO sentences which are the source of bugs are marked in red in the main window.

Within the main window a user can search/browse SUMO content. On the left there is a list of all entities found in selected SUMO ontologies. Because the number of entities is very big the view could be limited only to entities whose name starts with specific letter. On the right there is a set of sentences the entity is part of. There is also *Rule* tab containing axioms referring selected entity.



Term: HotelReservation

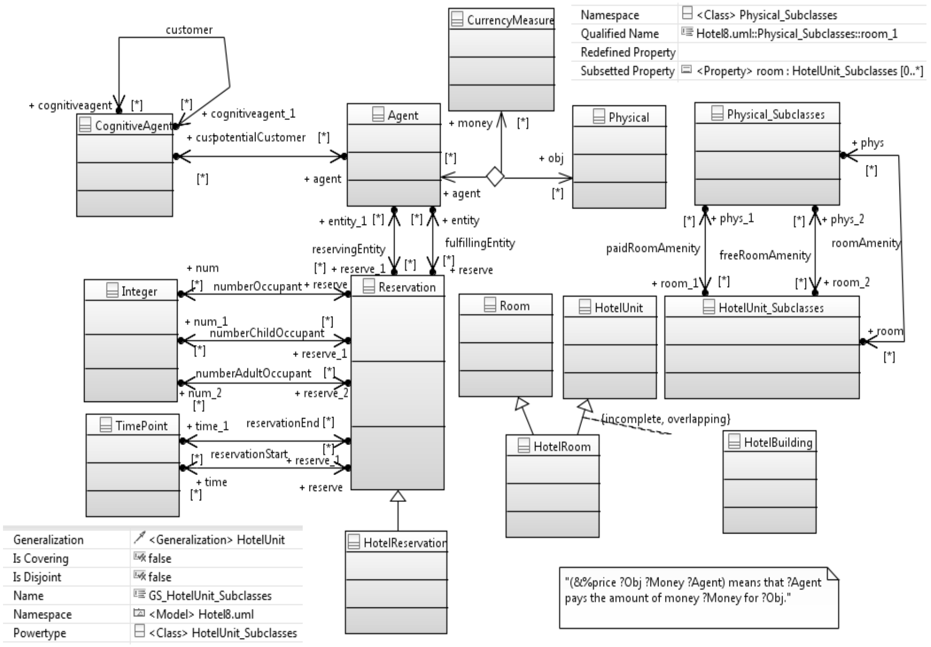
HotelMiniBar
HotelPackage
HotelRating
HotelReservation
HotelRoom
HotelRoomAttribute
HotelUnit
HotelVilla
HotTemperature
HotTub
HotWaterHeater

```
(documentation HotelReservation EnglishLanguage "&%HotelReservation refers to a&
(domain reservedRoom 1 HotelReservation)  Hotel.kif
(subclass HotelReservation Reservation)  Hotel.kif
(termFormat EnglishLanguage HotelReservation "hotel reservation")  Hotel.kif
```

Figure 6. SUMO to UML translator – the main window.

By a double click a user can select either entities or sentences to be translated to UML. Selected elements are marked in yellow – see Fig. 6. If a relation is selected its domains are automatically selected as well.

Fig. 7. presents the result of transformation done by the tool. For readability purposes the generated file is opened in Papyrus eclipse plug-in [16]. Examples of elements that can't be visualized ({subset} for association ends, properties of generalization set) are presented in the form of tables (see left-bottom and right-top corners).



Namespace	<Class> Physical_Subclasses
Qualified Name	Hotel8.uml::Physical_Subclasses::room_1
Redefined Property	
Subsetted Property	<Property> room : HotelUnit_Subclasses [0..*]

Generalization	<Generalization> HotelUnit
Is Covering	false
Is Disjoint	false
Name	GS_HotelUnit_Subclasses
Namespace	<Model> Hotel8.uml
Powertype	<Class> HotelUnit_Subclasses

Figure 7. SUMO to UML transformation example (automatic translation).

In comparison to the transformation performed manually (see Fig. 8) more than 70% of interesting elements were generated by the translator. Composition relationship between *HotelBuilding* and *HotelRoom* is absent because in manual transformation it was concluded from an axiom. Transformation of attributes is not implemented.

During the implementation stage some errors in the proposed transformation ([12]) were discovered, e.g. relations: *roomAmenity* and its children (*paidRoomAmenity*, *freeRoomAmenity*) have domains that are classes, what was not taken into account in [12]. Now, these relations are transformed to associations between appropriate power types. Also, the transformation of relations with one domain being a data value (e.g. Integers), was changed. Previously, such relations were modeled as attributes or association classes. Currently, relations are translated in the uniform way, and those with arity starting from 2 are represented by UML associations.

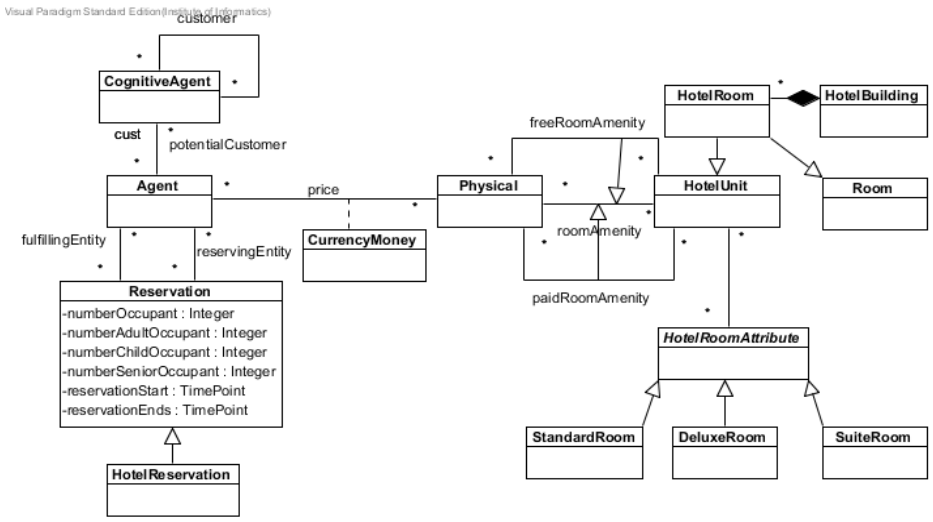


Figure 8. SUMO to UML transformation example
(manual translation, previous version of transformation rules).

7. Problems to be addressed

7.1. Meta-classes and meta-relations

SUMO, similarly to UML, is described in SUMO itself. Some elements of SUMO play a role of meta-classes, i.e. classes the instances of which are functions or relations; examples include *BinaryPredicate*, *IrreflexiveRelation*. Meta-classes are not directly translated to UML class diagram but they define important properties of other transformed elements, e.g. the arity of relations/functions. At that moment only arity is transformed. Another relation property, e.g. “reflexivity” constraint is not translated, but that could be done with the use of OCL.

Meta-relations are those relations that describe relationships between 2 or more classes or 2 or more relations; examples include: *subclass*, *partition*, *disjoint* for classes, and *subrelation*, *disjointRelation* for relations. In the current version of the tool only *subclass*, and *subrelation* meta-relations are transformed however the rest could be rather easily addressed, e.g. with the use of UML constraints for generalization sets, like {disjoint}, {complete}.

7.2. Class-creating functions

SUO_KIF has several functions that return a new class. For example *UnionFn* returns an anonymous class being a union of two classes, e.g. “(*domain CardinalityFn* 1 (*UnionFn SetOrClass Collection*))”, or *ExtensionFn* “which maps an attribute into the class whose condition for membership is the attribute”, e.g. “(*cardinality* (*ExtensionFn DevelopedCountry*) 35)” [5].

The SUMO sentences containing such functions are not transformed at that moment.

7.3. Class properties

In SUMO a property holds between an instance of entity and an instance of attribute. In UML properties are defined directly at class level and are shared (in structural sense) among all class instances. The SUMO class to which a given attribute is assigned to can be inferred. however the support of external tool like a theorem prover is needed. The integration with E prover is planned in the nearest tool release.

Additional problem is that instances of SUMO attributes could form a multi-level hierarchy, defined with *subAttribute* relation.

7.4. Axioms

SUMO axioms introduce constraints on ontology instances. The example below stays that every instance of *HotelBuilding* must have an instance of *HotelRoom* related as a proper part (asymmetric relation).

```
“(=>
    (instance ?HOTEL HotelBuilding)
    (exists (?ROOM)
      (and
        (instance ?ROOM HotelRoom)
        (properPart ?ROOM ?HOTEL)
      )
    )
  )”
```

Some of such axioms could be expressed directly in UML (e.g. the constraint above can be presented by composition relationship), some other could be translated into OCL. The current version of the tool allows reading the axioms but they can’t be selected for transformation.

8. Summary

The paper presents implemented features of the tool that supports SUMO to UML translation. The proof of concept shows that such transformation is feasible however is not trivial. Implementation problems result from different approaches to knowledge representation used in both formalisms. UML class diagram focuses on entities represented by classes and relationships between them. SUMO is much more flexible, enabling definition of axioms or relationships also between objects e.g. *subAttribute*.

Transformation rules implemented within the tool were identified and described in [10-12]. They were revised, some mistakes were found and corrected (however not all are included in the current tool release).

Static consistency rules allowed discovering many inconsistencies in existing ontologies, what resulted in increasing of their quality (most of found mistakes were corrected by SUMO developers).

The results of the tool applications are promising. The obtained domain class diagram is consistent, correct and complete to the level to which the input ontology has these features. These are the main benefits the tool can bring to potential users. Business expert or business analyst can use the tool to find out interesting notions, select them, and translate to a UML class diagram with a set of OCL constraints with one click. The user is warned about incompleteness, and inconsistencies found in the original files. He or she can experiment with transformation results, selecting new elements or un-selecting previously selected. The obtained UML model can be re-factored, and next transformed to other representations, e.g. programming languages.

The next release of the tool will address problems presented in subchapter 6. The usability of the final version of the tool is planned to be checked by business analysts.

References

[1] K. Bittner, I. Spenc, *Use Case Modeling*, Addison-Wesley Professional, 2003.
[2] I. Jacobson, G. Booch, J. Rumbaugh, *The Unified Software development process*, Addison Wesley Professional, 1999.
[3] D. Zowgi, V. Gervasi, The Three C's of Requirements: Consistency, Completeness, and Correctness, Foundations for Software Quality, Essen, Germany, Essener Informatik Beitiage (2002), 155-164.
[4] P. Mohagheghi, V. Dehlen, T. Neple, Definitions and approaches to Model Quality in Model-Based Software Development, A Review of Literature, *Information and Software Technology* (2009), 1646-1669.
[5] Suggested Upper Merged Ontology, http://www.ontologyportal.org (last access: 27 June 2015)
[6] A. Pease, Ontology: A practical Guide. Articulate Software Press, Angwin, 2011.
[7] http://sourceforge.net/projects/sigmakee/files/ (last access: 27 June 2015)
[8] http://protegewiki.stanford.edu/wiki/OWL2UML (last access: 27 June 2015)
[9] F. M. Suchanek, Ontological Reasoning for Natural Language Understanding, Master's Thesis in Computer Science, Saarland University, Germany, 2005 (available at http://suchanek.name/work/publications/master.pdf, last access 27 June 2015)
[10] B. Hnatkowska, Z. Huzar, I. Dubielewicz, L. Tuzinkiewicz, Problems of SUMO-like ontology usage in domain modelling, *Intelligent Information and Database Systems*, eds. N. Nguyen, B. Trawiński, K. Somboonviwat, Lecture Notes in Computer Science (2014), 352-363.
[11] I. Dubielewicz, B. Hnatkowska, Z. Huzar, L. Tuzinkiewicz, Domain Modelling in the Context of Ontology, Foundations of Computing and Decision Sciences (2015), 3-15.
[12] B. Hnatkowska, Z. Huzar, I. Dubielewicz, L. Tuzinkiewicz, Development of domain model based on SUMO ontology, accepted for publication, DEPCOS conference, 2015.
[13] http://www.antlr.org/download.html (last access 27 June 2015)
[14] A. Pease, Standard Upper Ontology Knowledge Interchange Format, 2009, http://sigmakee.cvs.sourceforge.net/viewvc/sigmakee/sigma/suo-kif.pdf (last access 27 June 2015)
[15] S. Schultz. System Description: E 1.8. Proceedings of the 19th LPAR, LNCS 8312 (2013), 477-483
[16] https://www.eclipse.org/papyrus/
[17] OMG, Unified Modeling Language, Version 2.5, September 2013
[18] OMG, Semantics of Business Vocabulary and Business Rules (SBVR), Version 1.2, November 2013
[19] J. Cabot, R. Pau, R. Raventós, From UML/OCL to SBVR specifications: A challenging transformation, Information Systems, Volume 35, Issue 4 (2010), 417-440.
[20] A. Marinos, S. Moschoyiannis, P. Krause, An SBVR to SQL Compiler, Proceedings of the RuleML-2010 Challenge (2010), http://ceur-ws.org/Vol-649/paper7.pdf (last access 27 June 2015)

Chapter 7

PIM-PSM Pattern-Aware Transformations

1. Introduction

The OMG's MDA is an approach supporting software development process by models and model transformations, which in principle is based on concept of separation of specification from implementation [6]. The MDA only sketches definition of models, model transformations and development process. Models in the MDA are defined at various levels of abstraction, in particular: PIM corresponds to level of system analysis and is related to system analysis model; PSM corresponds to various levels of details taking into consideration implementation platform, i.e. from architectural/design level to implementation level. Transformations in the MDA, in particular, are responsible for converting a PIM model to PSM level models, ideally in automatic manner.

In this paper we are detailing PIM, PSM and model transformations, and addressing one of major challenge of the MDA, i.e. automation of the path from models to executable systems [6]. Not all models are capable to be used for automation, the models have to be welldefined, i.e. requirements need to be sufficiently detailed and precisely defined [6]. Thereby, we introduce modelling standard of well-defined PIM models defined by set of allowed model elements and rules/constraints on elements creation and association. We narrow the standard to domain of information system. In order to make the standard applicable to commerce projects, it is derived from industry-based analysis modelling method [5]. System analytic is responsible for PIM creation.

Similarly, PSM have to be well-defined, nevertheless PSM represents design at various level of details. PSM at architectural/design level, is expressed by model elements representing architectural and/or design notions. However, PSM at implementation level is mapped directly to implementation of a system, it has to be expressed only using implementation platform notions. We narrow target implementation platform to Java platform. Hence, implementation level PSM consists of packages, classes, interfaces, operations, etc. mapped to respective Java language notions.

We propose transformation approach, which generates automatically implementation of a system, i.e. taking into consideration structural and behavioural aspects, in context of requirements defined in PIM, within a given architecture. In order to deal with complexity of PIM-PSM mappings, a particular transformation is a sequence of parametrized constituent transformations, i.e. transformation chain, in which a PIM is modified in successive steps until final PSM is reached. Each constituent transformation is responsible for applying architectural or design decision, in particular it is application of an architectural or design pattern. A transformation chain is created by an architect/designer, who selects and configures architectural and/or design patterns in order to shape a system architecture meeting functional and non-functional requirements. Thereby, various system designs might be expressed using transformation chains.

The chapter is organized as follows. We are describing PIM and PSM modelling standards in Section 2 and Section 3 respectively. Section 4 is dedicated to transformation approach and Section 5 introduces constituent transformations. An example transformation chain is discussed in Section 6. Finally, conclusions on the chapter are provided in section 7.

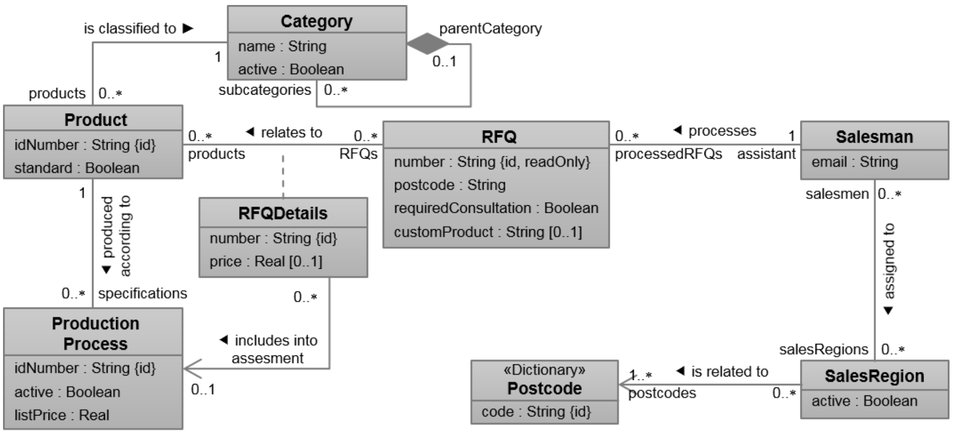


Figure 1. *Analysis Models - Information Model example*

2. Analysis Model - PIM

*Analysis Mod*el is specification of a system that abstracts from implementation details. In the context of the MDA approach, *Analysis Model* corresponds to PIM. In presented approach the model consists of:

- *Information Model* defines structure of information processed within the system, which is expressed by UML class diagram [11] (example is shown in Figure 1), together with lifecycles' specifications defined in form of *State-Machines*;
- *Use-case Model* defines *UseCases* and *Actors*, i.e. functionalities of the system and users who use those functionalities. A functionality is specified by: (a) use-case scenario, i.e. interactions between Actors and the system, modelled by an *Activity*, and (b) user interface model, i.e. graphical user interface and navigation paths between elements of user interface.

Use-case scenario is specified by means of Activities – each *UseCase* has an *Activity* as the *classifierBehavior* property defining its flow. The following section explains proposed extensions of an Activity's model elements that are used in the use-case scenario's specification to enable the PIM to PSM transformation. Activity diagram showing an example of use of described constructions is shown in Figure 2. Relations between Activities and other elements of *Analysis Model* are provided in [5].

An *Activity* is divided into *ActivityPartitions*, which represent participants of the behavior and group *Actions* for which they are responsible. Partitions used in the presented approach represents: actor's actions, behavior of a system presentation layer, behavior of a system logic layer.

An *Actor* is responsible for data input action, data modification, selection of data to further processing, option selection or decision confirmation. A presentation layer is responsible for actions, like data presentation, allowing data edition and option selecting etc. Business logic is specified in a logic layer, by means of read or write persistent data actions, data transformation or verification, calls to external system services etc.

Use-case scenario defined by an Activity starts from event received by an *AcceptEventAction* that represents a *UseCase's* trigger. The *AcceptEventAction* is marked by user-defined «UseCaseTrigger» stereotype.

Model elements of an *Activity* allow to specify all a *UseCase's* flows (i.e. main, alternate and exception) within an *Activity*. The *DecisionNode* separates particular flows, which may be steered by:

- Results of system processing, i.e values generated by previously executed Actions are used in evaluation of conditions of edges outgoing from the *DecisionNode*;
- User interface's events triggered by a user (like pressing a button) expressed by a *DecisionNode* with outgoing edges for each event definition. To make explicit connection between flows and user interface's elements, stereotype «EventDriven», pointing to pressed button (defined by *event* tagged value), is attached to each outgoing edge;
- An actor's decision on Actions to be taken, which is expressed by a *DecisionNode* with outgoing edges without any conditions.

Objects flows within an *Activity* are expressed by notion of *Pins* (*InputPin*, *OutputPin* and *ValuePin*), whose type must be a kind of *Class* from *Information Model* [11]. For the sake of readability and to enable the PIM to PSM transformation, following notation conventions and extensions are used:

- All explicit objects processed with an Action are available via *InputPins* and all needed objects related to those explicit are obtained by dot-notation within *Classes* of *Information Model*;
- All results of an *Action* processing are stored in *OutputPins* of the *Action*;
- To simplify frequent access to particular objects (in particular in case of multiple "temporary" data modification), a *DataStore* with «UseCaseLocalVariable» stereotype defined in presented approach is used;
- For an *Action* in the presentation layer, only *InputPins* are defined, which correspond to values to be displayed by elements of graphical user interface;
- For an *Action* in the actor layer, only *OutputPins* are defined, which correspond to values provided or selected by an *Actor*;
- For an *Action* accessing persistent objects, *InputPins* correspond to parameters of persistent objects selection criteria, while *OutputPins* contains selected persistent objects;

- For an *Action* storing persistent objects, *InputPins* contain objects to be persisted. *OutputPins* are defined conditionally, when persistent objects are modified within the *Action*, and contain input and modified objects;
- For an *Action*, which transforms or computes values and/or objects, *InputPins* contains computation input parameters, while results are stored in *OutputPins* of the *Action*.

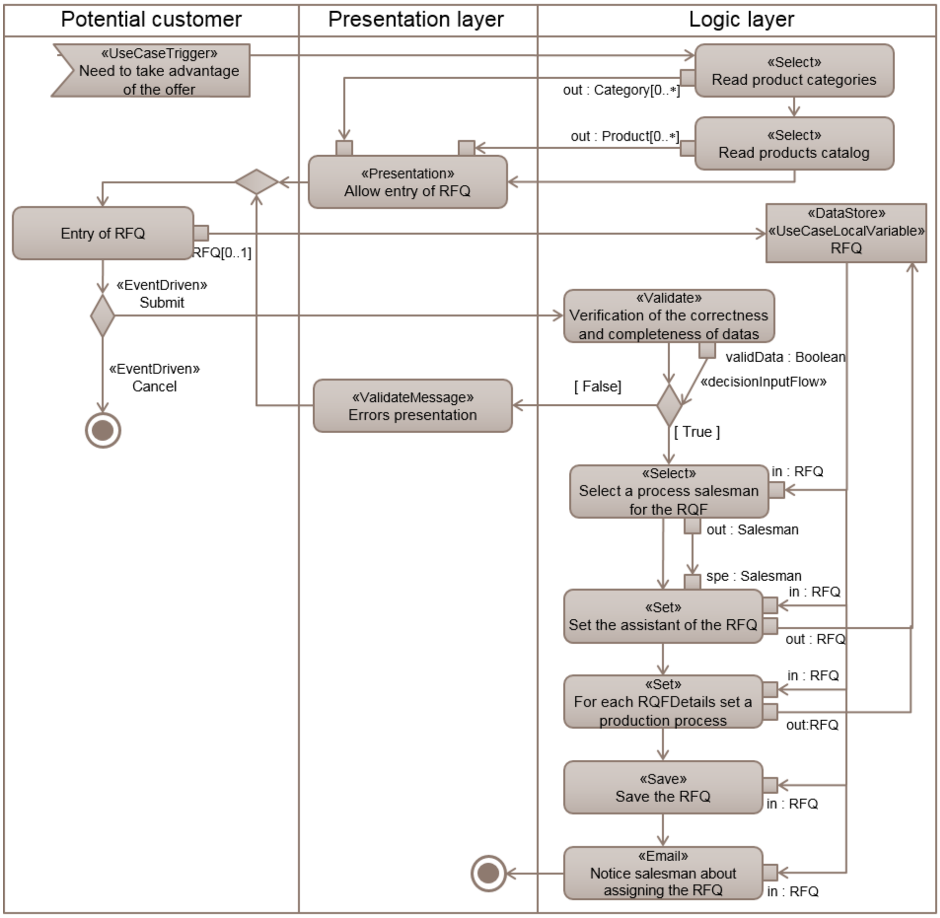

Potential customer
Presentation layer
Logic layer
«UseCaseTrigger» Need to take advantage of the offer
«Select» Read product categories
out : Category[0..*]
«Select» Read products catalog
out : Product[0..*]
«Presentation» Allow entry of RFQ
Entry of RFQ
RFQ[0..1]
«DataStore» «UseCaseLocalVariable» RFQ
«EventDriven» Submit
«Validate» Verification of the correctness and completeness of datas
validData : Boolean
«decisionInputFlow»
«EventDriven» Cancel
[False]
«ValidateMessage» Errors presentation
[True]
«Select» Select a process salesman for the RQF
in : RFQ
out : Salesman
spe : Salesman
in : RFQ
«Set» Set the assistant of the RFQ
out : RFQ
in : RFQ
«Set» For each RQFDetails set a production process
out:RFQ
«Save» Save the RFQ
in : RFQ
«Email» Notice salesman about assigning the RFQ
in : RFQ

Figure 2. *Analysis Models - Use-Case scenario example*

Actions defined within the presentation layer might be refined by following stereotypes:

– «Presentation» – an Action is realized by a user interface's element specified by *elementUI* tagged value;

– «PresentationMessage» – an *Action*, which displays various types of *messages*, like information, warning, error messages to a user. Message content is provided in *message* tagged value, while *dialogType* tagged value points to message dialog;

– «ValidateMessage» – an *Action* is responsible for presentation of validation messages within user interface.

In order to make processing logic detailed, precise and unambiguous, *Actions* defined within the logic layer are complemented by rules and constraints on data processing. Thereby *Actions* are specified by *Object Constraint Language* (OCL) [10] statements, where an *Action's InputPins* and *OutputPins* correspond to input and output parameters respectively.

For the sake of specification readability and precision, following notation conventions and extensions as well as interpretation are assumed:

- operation *allInstances*() – returns all persistent objects of a given *Class*;
- operation *oclIsNew*() – verifies whether the object is new (created in the *Action*, not persisted yet), whether properties of the object have default values, and the object is in an initial *State* with respect to a corresponding *State-Machine*;
- expression: *out=in* – states that the *out* object is the *in* object, which properties might be modified in remaining part of a OCL statement.

Following stereotypes are available for all *Actions* presented in logic layer:

- «Select» – an *Action* selects persistent objects. A criterion of selection is given in *criterion* tagged value, which might be parameterized by values of *InputPins*. Example statements for «Select» *Actions* from the *Activity* on Figure 2 are given below:
  - 'Read product categories': *criterion* = *out=Category.allInstances()->select(active),*
  - 'Read products catalog': *criterion* = *out=Product.allInstances()->select(standard),*
  - 'Select a process salesman for the RQF': *criterion* = *out=SalesRegion.allInstances()->select(active and postcodes->includes(in.postcode))->flatten()->any(true)*
- «Set» – an *Action* sets properties values of a processed object. The values are defined using OCL statements. Example statements for «Set» *Actions* from the *Activity* on Figure 2 are given below:
  - 'Set the assistant of the RFQ': expression= out=in and out.assistant=spe
  - 'For each RQFDetails set a production process': *expression* =

    out=in and out.RFQDetails->forAll(sp |
    let ap:Set(productionProcess)=
    sp.products.specifications->select(active) in
    if ap->size()=1
    then sp.productionProcess=ap->any(true)
    else (sp.productionProcess.oclIsUndefined() and
    out.requiredConsultation)
    endif) and (not out.customProduct.oclIsUndefined()
    implies out.requiredConsultation)
- «Save» – an *Action* persists an object. It is assumed that properties of the stored object are computed in predecessor Actions on the logic layer and

Actions on the actor layer related to providing data via user interface, but also they results from constraints on Information Model, like {*id*};

- «Delete» – an *Action* deletes a persistent object. It also remove recursively all constituent objects, i.e. related by means of composite association;
- «Validate» – an *Action* validates data provided by an Actor using user interface's elements according to validation rules;
- «Email» – an *Action* sends e-mail message. A message of an e-mail is defined in *text* tagged value, and the *email* recipient in email tagged value.

3. Implementation Model – PSM

Implementation Model defines a system only in terms of implementation languages, platforms, libraries, API etc., and it is detailed enough to generate directly complete implementation of the system. In the context of MDA approach, *Implementation Model* corresponds to PSM at the very end of transformation process, i.e. which contains all implementation details required to generate complete source codes of a system.

In the chapter, *Implementation Model* defines a system in terms of Java platform, and related libraries, frameworks, etc.. Thereby, it determines set of model elements that might be used in the model:

- structural elements – which are expressed by elements of a class diagram: *Packages*, *Classes*, *Interfaces*, etc., and corresponds to Java packages, classes, interfaces etc.;
- behavioural elements – which are *Interactions*, and are expressed by elements of a sequence diagram: *Messages*, *Lifelines*, *CombinedFragments*, etc., and corresponds to Java implementation of operations;

Implementation Model consists of *Packages*, which contain *Classes* and *Interfaces*. Each *Class* or *Interface* can imports either *Classes*, *Interfaces*, or whole *Packages* (in fact all elements in *Packages*), which in Java means to import either particular class/interface, or import content of given package [11].

Following rules should be taken into account by structural elements of *Implementation Model*:

- Every *Class* or *Interface* contains *Properties*, with names and types, and *Operations*, with names and parameters, which are Java attributes and operations respectively, please refer to Figure 3.
- Each *Operation* has always one return parameter, and can has zero or more input parameters.
- Types of *Properties* and *Operations'* parameters are either a primitive type or a reference to any accessible *Class* or *Interface* representing Java classifiers.
- Additionally, each Java attribute or parameter can define collection of values, i.e. table, which is ordered and not unique, what in terms of a modelled attribute/parameter definition means: *isOrdered=true, isUnique=false, lowerValue=0,upperValue=**.

- Each *Property* or *Operation* has additional modifiers: visibility, whether a feature *isStatic*, etc. which are mapped to equivalents in Java language.
- *Interfaces* can have only *Properties* that are static and has public visibility.

When a system is defined in *Implementation Model*, it has to refer to existing Java Software Development Kit, Java API, frameworks, libraries etc. Thereby, it is possible to import to *Implementation model* external models containing *Packages*, *Classes* and *Interfaces*, which corresponds to required APIs, libraries, frameworks, etc.. Obviously, not full models are incorporated, only elements that are accessible, without any operations implementation.

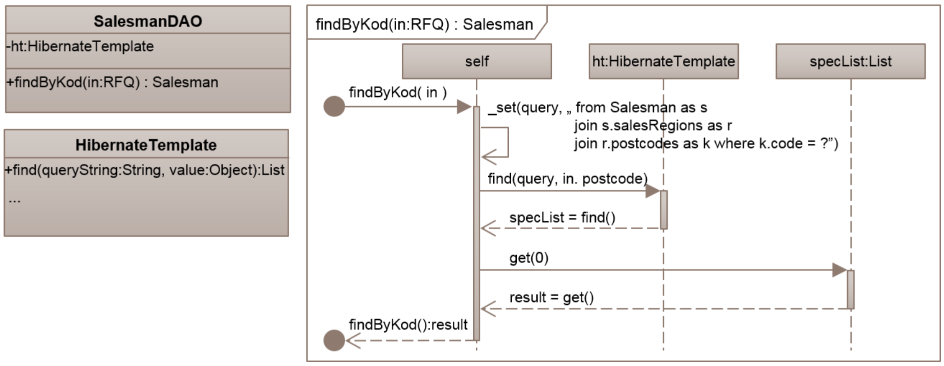

SalesmanDAO
-ht:HibernateTemplate
+findByKod(in:RFQ) : Salesman
HibernateTemplate
+find(queryString:String, value:Object):List
...
findByKod(in:RFQ) : Salesman
self
ht:HibernateTemplate
specList:List
findByKod(in)
_set(query, „ from Salesman as s
join s.salesRegions as r
join r.postcodes as k where k.code = ?”)
find(query, in. postcode)
specList = find()
get(0)
result = get()
findByKod():result

Figure 3. Implementation Model example

All Operation of *Classes* defined in *Implementation Model* have to point to behavior, which corresponds to method definition in Java. The behavior is defined by an *Interaction* using sequence diagram, with respect to convention presented below [11].

- The *Interaction* consists of a *self LifeLine* which represents an object, or a *Class* in case of a static *Operation*. The *self LifeLine* defines a sequence of *Messages*, which corresponds to a sequence of statements in Java method. Only interactions between the *self LifeLine* and other *LifeLines* are shown, no other *Message* exchanges are added in the *Interaction*, please refer to Figure 3.
- Local variables of Java methods are defined as *Properties* of the *Interaction*, i.e. they are created before the *Interaction's* execution, are destroyed after. The *Properties* have to be initialized before they are read.
- First received *Message* on the *self LifeLine* is a *Message* from environment corresponding to the *Operation* call, in form of a found synchronized *Message* (*messageKind=found, messageSort=synchCall*), optionally with arguments actually being call parameters.
- Assigning value to a *Property*, which corresponds to Java assignment statement, is defined by convention, i.e. a recursive, asynchronous *Message _set(var,l-value)*
(please refer to *_set Message* on Figure 3) where:

- *var* – the *Property* name, which can be either the *Property* of the *Operation's Class*, or *Property* of the *Interaction*, which corresponds to Java local variable,
- *l-value* – OCL statement, limited to simple statements referring only to *Properties*, which evaluates to value of *var's* type.

- Calls of other *Operations* are modelled by sending a call *Message* (*messageSort=synchCall*), which points to the called *Operation* and arguments representing the called operation's actual parameters. The *Message* is sent to a *LifeLine* which represents a called object, which is specified either by the *Interaction's Property* or the *Operation's Class*. Please refer to example calls on Figure 3.
- A new object is created using a create *Message* (*messageSort=createMessage*), which arguments are actual parameters of the new object's constructor *Operation*.
- Control instructions like *if, while* are modelled using *CombinedFragments*, i.e. *alt, loop*. A condition of combined statement are defined using OCL statements.
- The Interaction finishes by sending *Message* to environment, in form of a lost return *Message* (*messageKind=found, messageSort=reply*) with arguments containing return value.

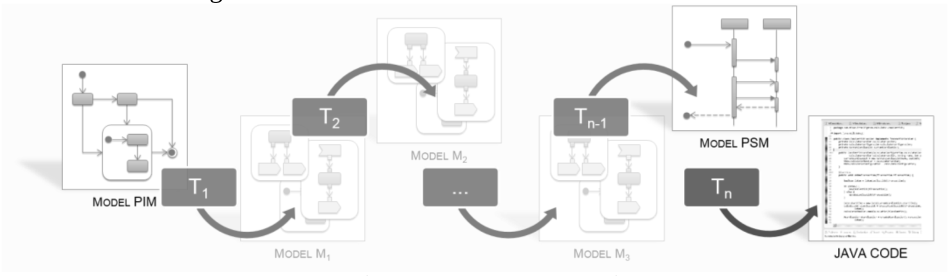


Figure 4. Transformation Composition specification

4. Transformation approach

Main goal of the transformation approach is to support transformations of PIM to PSM at implementation level, from which base code of a system is generated. Thereby, the transformation approach should be able to perform mapping of both structural and behavioural aspects of a system, in order to fully implement all requirements stated in a PIM. Considered transformations would be rather complex, hence an ability to deal with

compound cases should be also addressed by the approach. Moreover, we want to reuse once written transformations in other projects, i.e. the approach should support reusability. Finally, the approach should represent design decisions in explicit way, to support architects/designer in system designing activities.

We address above requirements by transformation chains and constituent transformations. A transformation chain is a sequence of transformations steps, executed in a given order, please refer to $T_1;T_2;\ldots;T_{n-1};T_n$ on Figure 4. Model-to-model [8] and model-to-text [9] mappings are allowed [3]. Each transformation step calls a constituent transformation with configured parameters values.

A model-to-model constituent transformation is responsible for applying architectural or design decision. A model-to-text constituent transformation is responsible for source code generation. In assumptions, a constituent transformation should be reusable, i.e. it should be possible to include it as a part of other transformation chain. In most cases, the constituent transformation encodes knowledge present in architectural or design patterns, libraries etc., hence it is an application of pattern, or application of platform library or framework usage, or mapping of analysis object-oriented constructs to design/implementation object-oriented level. Each constituent transformation has formal parameters, steering transformation execution, e.g. parametrizing application of a design pattern.

A software designer is responsible for selecting constituent transformations, ordering them properly, and setting suitable configurations to generate an architecture, which fulfils functional and non-functional requirements. In particular, the designer has to select all necessary transformations that will transform PIM to a valid PSM at implementation level, i.e. model fulfilling constraints defined in Section 3.

Let's discuss an example presented on Figure 4. When a transformation chain is run, a model (a PIM at the beginning of the run, please refer to Model PIM on Figure 4) is gradually processed in each transformation step. The step passes the model and parameters values to a constituent transformation (please refer to $T_1;T_2\ldots;T_{n-1}$ on Figure 4) and executes it creating updated version of the model (please refer to $M_1;M_2;\ldots;M_n$), what corresponds to application of a design decision. The process is repeated to end of the chain, which should result in generation of a PSM at implementation level (please refer to Model PSM on Figure 4). Finally, a model-to-text transformation generates a source code from the PSM (please refer to T_n, Java Code, and Model PSM on Figure 4).

Summing up, the proposed approach supports compound transformations by transformation chains. It provides reusability by model-to-model and model-to-text constituent transformations, which may be used within different transformation chains configuration. It represents explicitly design decisions by transformation chains, constituent transformations and their configuration. Mapping of structural and behavioral aspects is covered by transformation chains, which are discussed in Section 5.

5. Constituent Transformations

We distinguish following types of model-to-model constituent transformations. Constituent transformations applying architectural or design patterns to a model. We are considering Layer and Domain Objects architectural patterns [12], and Explicit Interface [12], Data Access Object [1] design patterns. Another constituent transformation type applies usage of target platform libraries and frameworks. In this chapter we are introducing only Hibernate ORM transformation. In the separate category are mappings of analytic object-oriented constructions to implementation level, represented in this chapter by Analysis Information Model transformation.

Additionally, there are auxiliary transformations, i.e., Split Activity, Connected Activities, Generate Operation and Activity to Interaction. They represent common operations on model elements, often used by constituent transformations.

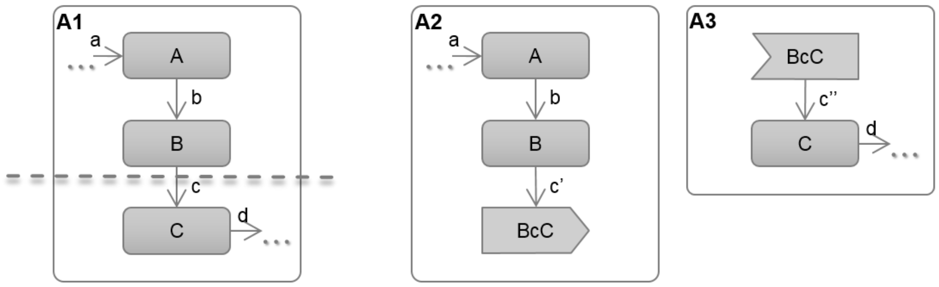
Because of space limits, we omit details on model-to-text constituents. Nevertheless, application code is created using "Model to Text Transformation Language" [9], a template-based textual approach, which generates files filling template meta-variables [3].

5.1. Split Activity

The transformation splits an Activity into set of Activities. It moves nodes from the activity to corresponding output Activities, taking into consideration splitting criteria, i.e splittingCriteria. The transformation preservers structure and dynamics of control and object flows of the activity.

Parameter	Description
activity:Activity	an *Activity* to be splitted
splittingCriteria:Set(Tuple{ activityName:String, activityNodes: Set(ActivityNode)})	splitting criteria of *activity* in form of set of pairs where: *activityName* - name of a result *Activity*; *activityNodes* – set of *ActivityNodes*, defined within *activity*, which will be moved to the result *Activity*

Results For each item of *splittingCriteria* set, a new *Activity* is created with name *activityName* and all *activityNodes* are moved to the *Activity*. For example, Figure 5 is illustration of an *Activity* division, in which horizontal dashed line sets splitting criteria. Figure 5 shows that new *Activities* A2 and A3 are created, and *ActivityNodes* A, B are moved to the *Activity* A2, but node C is moved to the *Activity* A3.



5.1.1. *Figure 5.* *An example application of Split Activity*

ActivityEdges of the *activity*, which source and target *ActivityNodes* are moved to same result *Activity*, are also moved to the result *Activity*. In Figure 5, *ActivityEdges* a and b are moved to an *Activity* A2, and *ActivityEdge* d is moved to an Activity A3. However, *ActivityEdges* of the *activity*, which source and target *ActivityNodes* are moved to different result Activities, are splitted, for instance an ActivityEdge c is splitted between *Activities* A2 and A3, please refer to 5. In such case for each splitted *ActivityEdge*: A Signal is created and labelled by unique id, and pairs of *SendSignalAction* and *AcceptEventAction* are added to resulting *Activities*.

Because of space limits we will not analyze all cases of an *Activity* division. However, a case discussed above clarifies a general idea of the Split *Activity* transformation.

5.2. Connected Activities

The transformation splits *activity* into set of connected *Activities*.

Parameter	**Description**
activity:Activity	an *Activity* to be splitted into connected *Activities*

Results An *Activity* is connected when for any two *ActivityNodes* within the *Activity* a path joining those nodes exists. The transformation finds biggest connected sub-activities within partitioned *Activity* and moves them to separate *Activities*. Since an *Activity* is a graph, the transformation finds connected components of the *activity* and moves them to separate *Activities*. The transformation splits the activity using a graph-search based algorithm.

5.3. Generate Operation

Transformation converts *activity*, that are results of Split *Activity* or Domain Objects transformations (please refer to Section 5.1 and Section 5.6 respectively), to routine-like behavior that conforms paradigms of structured programming: sequence of instructions, conditional statements, iterations, calls to other sub-routines, no jump instruction etc. Next, the transformation creates an Operation representing the *activity* and adds it to a Class, owner of the *activity*. Finally, all other *Activities* using the *activity* are modified, in order to call the Operation rather than the *activity* itself.

Parameter	**Description**
activity:Activity	modified *Activity*

Results The transformation can be run if the activity is defined in context of a *Class*, and when the *activity* is connected, please refer to Connected *Activities* transformation in Section 5.2.

The transformation creates an Operation in the context *Class* to represents the *activity*. Next, it modifies the *activity*, adds *Parameters* to the *Operation*, and updates parts of models from which the *activity* is invoked. Because of space limits, we discuss only basic scenario, however it explains idea of the transformation.

An *AcceptEventAction* and a *SendSignalAction* of the *activity* are replaced by *InitialNode* and *FlowFinalNode* respectively, please refer to the *Activity* A1 and the *Activity* A2 on Figure 6. Next, an *Operation* is created and attached to the *Class*, please refer to an *Operation opA2( )* that is created within the *Class C* on Figure 6.

On the other side, an *SendSignalAction* to the activity is changed to a *CallOperationAction*, please refer to Figure 6. The *Action A* in the *Activity B1* is changed to the *CallOperationAction opA2* in the *Activity B2*. Next, an *AcceptEventAction* is removed and all outgoing edges are pinned to the *CallOperationAction*. Please refer to the *Action D* in the *Activity B1* which is removed from the *Activity B2,* on the Figure 6.

5.4. Activity to Interaction

The transforms builds from an *Activity* an *Interaction*, with respect to constraint and rules defined in Section 3

Parameter	**Description**
activity:Activity	modified *Activity*

Results The transformation traverse each node of the *activity*, starting from nodes having no predecessors (in most cases initial nodes), and generates elements of the *Interaction,* with respect to following rules:

- *InitialNodes* are replaced by found *Messages* with arguments corresponding to input parameters of the *activity*,

– «Set» *Actions* are replaced by *Messages* corresponding to assignment statements,

– any other stereotyped *Actions* like «Select», «Save», «Validate», etc. are mapped to *Messages*, which call Implementation Platform API, libraries, etc. For instance, «Select »action might be mapped to calls to Hibernate [4] library, in particular to *HibernateTemplate* class,

- DecisionNodes are mapped to alt or loop CombinedFragments,
- *FlowFinalNodes,* or *Actions* without successors are replaced by lost *Messages* with proper return value, if any.

5.5. Layers

The transformation is an application of Layers pattern [12]. The transformation divides into layers selected behaviour, i.e. *Activities*. The *Activities* as well as related structural elements (*Classes, Interfaces*, etc.) are divided, however structural division corresponds to behaviour partitioning. According to the definition of Layers pattern [12], a layer can communicate only with other layers that are lower in hierarchy.

Parameter	**Description**
activities:Set(Activity)	*Activities* to be splitted
layers:OrderedSet(Tuple{ layer: String, activityNodes : Set(ActivityNode)})	criteria of partitioning the *activities* into layers, given as an ordered set of pairs where: *layer* – name of a layer; *activityNodes* - set of nodes, defined within one of *activities*, which will be moved to the result *Activity* of the layer

Results The transformation applies Layers design pattern [12], by creating packages *Package* for each layer. The *activities* are splitted against *layers* criteria using Split *Activity* transformation, please refer to 5.1. Resulting *Activities* are moved to corresponding *Packages*. Moreover, all structural elements (like *Classes*, *Interfaces*), used by any *Activity* in a given layer, are moved to it. If a given structural element is used by many layers, it is moved to lowest layer in a hierarchy, and proper *ElementImports* are added to all layers using the element.

5.6. Domain Objects

Transformation is an application of Domain Object pattern [12]. The transformation extracts from selected behaviour, given in form of *Activities*, independent and coherent functionalities and place them in domain objects. The objects are created with respect to a given division criteria.

Parameter	**Description**
activities:Set(Activity)	*Activities* to be splitted
domainObjects:Set(Tuple{ domainObject: String, activityNodes : Set(ActivityNode)})	criteria of partitioning the *activities* into domain object as a set of pairs where: *domainObject* – name of a domain object; *activityNodes* - set of nodes, defined within one of *activities*, which will be moved to the result *Activity* of the domain object

Results For each item in *domainObjects* a *Class* is created. Next, the *activities* are splitted against *domainObjects* criteria using *SplitActivity* transformation. Resulting *Activities* are placed in corresponding *Classes*.

5.7. Explicit Interface

The transformation applies Explicit Interface pattern [12], by creating an *Interface* for a set of *Operations* (owned by one *Class*) and modifies all callers to call the *Operations* via the *Interface*.

Parameter	**Description**
interfaceName:String	name of a resulting *Interface*
operations:Set(Operation)	*Operations* that will be present in the resulting *Interface*

Results The transformation creates an *Interface* that contains all *Operations* from the *operations*. All *CallOperationAction* in all *Activities*, are modified in order to call the *operations* via the *Interface*. Finally, an *InterfaceRealize* between the *Class* and the *Interface* is added.

5.8. Data Access Object

The transformation applies Data Access Object pattern [1]. It creates DAO *Classes* for selected *Classes* of persistent objects. *New Operations* responsible for the objects management are created and added to the DAO *Classes*. Behaviors of the *Operations* are taken from selected *Activities*.

Parameter	Description
persistentObjects:Set(Class)	*Classes* for which DAO *Classes* will be created
activities:Set(Activity)	*Activities to be splitted among* DAO *Classes*

Results Pattern application starts from splitting the *activities* into domain objects (please refer to Section 5.6) in order to group *Actions* of *activities* with respect to manipulated persistent objects *Classes*. Value of Domain Object's *activities* parameter is *activities*. Value of Domain Object's *domainObjects* are definition of objects for each persistent object in *persistentObjects*. It is expressed as follows (using simplified version of division algorithm, taking into consideration return type of an *Action*):

persistentObjects->collect(cl:Classifier| Tuple{ domainObject = cl.name+'DAO',

activityNodes = actions-> select(a:Action | a.output-> at(1).oclIsTypeOf(cl)))}.

In next step, each domain object's *Activity* is divided into connected *Activities* by Connected Activities transformation (please refer to Section 5.2). Finally, for each generated *Activity*, an Operation is created by Generate Operation transformation (please refer to Section 5.3)

5.9. Hibernate ORM

The transformation is responsible for applying usage of Java Hibernate ORM library [4]. It imports elements of Hibernate APIs, *Classes*, *Interfaces*, etc. to a PSM model. It enhances *Classes* of persistent objects with model elements required by Hibernate. Last but not least, it transforms selected Activities to Interactions, adding required calls to the Hibernate API, using Activity to Interaction transformation defined in Section 5.4.

Parameter	Description
persistentObjects:Set(Class)	*Classes of persistent objects to be managed by Hibernate ORM*
activity:Set(Activity)	*Activities* to be mapped. They should contain *Actions* that manipulate *persistentObjects*

5.10. Analysis Information Model

In fact it is a group of transformations, which removes from *Information Model* (or any selected part of a model), analytical constructions, which are not present in *Implementation Model*, e.g. n-ary association, association classes, multiple inheritance. Additionally, the transformations map data types from *Informational Model* to types available in *Implementation Model*. Another side-effect of the transformations are changes in all *Activities*, which *Actions* are using transformed elements of *Information Model*, i.e. they have to be modified to be consistent with changes in the *Information Model*.

Parameter	**Description**
elements: Set(PackagableElement)	elements of *Information Model* to be transformed

6. Case Study – Transformation Configuration and Execution

We are discussing a system supporting sales in manufacturing company. Its PIM is introduced in Section 2, (please refer to Figure 2, and Figure 1).

It is assumed that designer creates configuration of 3-tiers architecture [1]: presentation tier, business tier, and integration tier, and the system is implemented in Java, using Hibernate ORM library. Because of space limits, we discuss only a transformations for the integration tier. For the sake of simplicity we introduced herein OCL functions: actionsWithStereotypes, which returns all elements with given stereotypes and actionsOn-Partition, which returns all actions on a given partition. A transformation chain, and its steps configuration are following.

At the beginning, all analytic constructions of the PIM are converted to objecto-riented notion available in Java.

Analysis Information Model - Section 5.10
elements = *Package.allInstances()-> select(p:Package|p.name = 'InformationModel')->any(true). packagedElement*

In this step, a 3-tiers architecture is applied, by splitting the system into layers. All Activities of UseCases of the PIM are processed. Resulting layers are: Presentation, which contains Actions moved from the PIM's Presentation layer; Business, and Integration, which contain Actions moved from the PIM's Logic layer responsible for data processing and persistence respectively.

Layers - Section 5.5
elements = UseCase.allInstances()-> *collect(uc|uc.classifierBehavior.oclAsType(Activity))->asSet()* *layers* = *OrderedSet{* *Tuple{layer = 'Presentation',* *activityNodes = actionsOnPartition('Presentation layer')->*

union(actionsWithStereotypes(Set{'Validate'}))},
Tuple{layer = 'Business', activityNodes = actionsOnPartition('Logic layer') – actionsWithStereotypes(
Set{'Select','Save','Delete'})},
Tuple{layer = 'Integration', activityNodes =
actionsWithStereotypes(Set{'Select','Save','Delete'})} }

In current step, DAO Classes are created. It splits an Activity related with the Integration layer into DAO Classes, taking into consideration all Classes of persistent objects created by previous transformations steps.

Data Access Object - Section 5.8

activities = Package.allInstances()->
select(p|p.name = 'Integration').any(true).packagedElement->
select(el:PackageableElement|el.oclIsTypeOf(Activity) and el.name = 'Integration')
persistentObjects = informationalModel

Next, Hibernate ORM is applied to the system. Obviously, all Classes of persistent objects modified in previous transformations steps are passed to the persistentObjects parameter. Moreover, all Operations of the DAO Classes are translated from Activity to Interaction and calls to Hibernate ORM API are added.

Hibernate ORM - Section 5.9

activities = Package.allInstances()->
select(p|p.name = 'Integration').any(true).packagedElement->
select(cl:PackageableElement|cl.oclIsTypeOf(Class) and
cl.name.indexOf('DAO') > 0)->collect(cl:Class|cl.ownedOperations.method)->
flatten()->asSet()
persistentObjects = informationalModel

Finally, an application code is generated using already discussed model-to-text mappings.

7. Conclusions

The PIM is transformed to PSM, by sequence of constituent transformations, during which PIM is gradually modified to become a model of the system at implementation level. PIM is *Analysis Model*, which is an adaptation of the approach presented in [5], joining commercial experiences in system analysis, precision and unambiguousness. PSM is expressed by concepts of programming languages, platform elements, and calls to libraries.

Similar approaches are discussed in [2], [13], [14], [15]. In [14], [2], transformation composition are discussed, however neither approach supports design discipline in explicit way. In [13] predefined architectural and design patterns are provided, thereby only systems with fixed architecture can be generated. Finally, [15] provides most extensive approach, however it supports process of PIM generation, rather than PIM-PSM transformation.

In future, first challenge is to provide precise and expressive enough approach for modelling user interface and its behaviour, some promising solutions can be found in [7]. Other future work is to develop constituent transformations being application of other design patterns, especially related to user interface like Model-View-Controller [1], [12].

8. References

[1] Crupi J., Alur D., Malks D.: Core J2EE Patterns: Best Practices and Design Strategies, Second Edition, Prentice Hall, 2003

[2] Cuadrado J., Molina J.: Modularization of model transformations through a phasing mechanism. Software and Systems Modeling, 2009.

[3] Czarnecki K., Helsen S.: Feature-based survey of model transformation approaches. IBM Syst. J., 2006.

[4] Hibernate – Relational Persistence of Idiomatic Java, http://docs.jboss.org/hibernate/orm/4.1/manual/en-US/html_single/, 2013

[5] Kasprzyk A., Walkowiak A.: Biznesowe korzy´sci ze stosowania standardów oraz kompleksowych procesów wytwórczych w analizie biznesowej oraz systemowej. In˙zynieria oprogramowania. Badania i praktyka; Nakom, Poznań, ISBN: 978-83-63919-15-3, 2014

[6] Miller J., Mukerji J., (eds). MDA Guide Version 2.0 OMG, 2014

[7] Object Management Group. Interaction Flow Modeling Language (IFML), 1.0, http://www.omg.org/spec/IFML/, 2015

[8] Object Management Group. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Technical report, http://www.omg.org/spec/QVT/1.1, 2011

[9] Object Management Group. MOF Model To Text Transformation Language (MOFM2T), 1.0, http://www.omg.org/spec/MOFM2T/1.0/, 2008

[10] Object Management Group. Object Constraint Language (OCL), http://www.omg.org/spec/OCL/2.3.1/, 2012

[11] Object Management Group. Unified Modeling Language (OMG UML), http://www.omg.org/spec/UML/2.5, 2013

[12] Schmidt D. C., Henney K., Buschmann F.: Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing, John Wiley & Sons, 2007.

[13] Smialek M., Jarzebowski N., and Nowakowski W.: Translation of Use Case Scenarios to Java Code, Computer Science, 2012

[14] Vanhooff B., Ayed D., Van Baelen S., JoosenW., Berbers Y.: Uniti: A unified transformation infrastructure. In Model Driven Engineering Languages and Systems, 10th International Conference, MoDELS 2007, Nashville, USA, Proceedings, pages 31–45. Springer, 2007.

[15] Yue T., Briand L. C., and Labiche Y.: aToucan: An Automated Framework to Derive UML Analysis Models from Use Case Models, ACM Transactions on Software Engineering and Methodology, 2015

Chapter 8

Weak Separation of Tightly Coupled Concerns with Generic Program Representations

1. Introduction

Generic is a common way to avoid duplicating code. The Standard Template Library (STL) [1] is a premier example of engineering benefits of generic program representations. Genericity is a central theme in software reuse, component-based, pattern-driven development (e.g., facilitated by .NET™ or JEE™), and architecture-centric Software Product Line (SPL) [1][2] approaches.

Genericity, as understood in this paper, aims at achieving non-redundancy, by unifying similar software structures with generic program representations to achieve program simplification, reusability, or maintainability.

The importance of genericity in managing software complexity has been recognized for long. Macros were one of the early attempts to make programs more generic. Goguen popularized ideas of parameterized programming [3]. Type parameterization [4] (called generics in Ada, Eiffel, Java and C#, and templates in C++), higher-order functions [5], and inheritance can help avoid repetitions in certain situations. Design techniques such as iterators, design patterns [6], table-driven design (e.g., in compiler-compilers), and modularization with information hiding [7] can help us build generic programs. Generative programming techniques, such as XML-based Variant Configuration Language (XVCL) [8], build a generic program representation at the meta-level, and derive concrete programs, with possible redundancies, from the generic meta-level representation.

We can conceive a “generic program representation” as a parameterized structure that can be turned into a concrete, custom program solution by instantiating the parameters. The nature of parameters, the mechanism for instantiating parameters, and the overall process that leads to instantiating a concrete program solution from its generic counterpart depends on the techniques used for generic design. Parameterized structures can be as simple as generics or templates, or as complex as an Object-Oriented (OO) framework or a generic parser. In an OO framework, parameters are abstract classes and design patterns. Parameters for a generic parser are encoded in BNF (Backus Normal Form) definition of a programming language syntax. XVCL employs an unrestricted form of parameterization, in which any software structure may become a parameter of any other software structure.

A *concern* is any area of interest in a program solution, pertinent to functional features, quality requirements, software architecture, detail design, or implementation. The idea of separation of concerns (SoC) is to break a program into distinct concerns in order to deal with them separately. As we do so, we try to limit interactions between concerns as much as it is possible.

SoC principle states that “a given problem involves different kinds of concerns, which should be identified and separated to cope with complexity, and to achieve the required engineering quality factors such as robustness, adaptability, maintainability, and reusability” [9]. We can apply the SoC principle at the levels of program analysis, design, and implementation [10].

SoC in software development domain was first proposed by Dijkstra in early 1980’s [11], as a conceptual tool to tackle software complexity. Certain concerns can be nicely aligned with modular decomposition. Concerns that cannot be localized in modules have a crosscutting effect on program modules. Recently, a number of unconventional approaches have been proposed to help in separating and localizing some of the crosscutting concerns at the meta-level plane of program representation such as Algebraic Hierarchical Equations for Application Design (AHEAD) [12], Aspect-Oriented Programming (AOP) [13], Multi-Dimensional Separation of Concerns (MDSOC) [10], or XVCL [8].

In this paper, we concentrate on cases where physical SoC becomes difficult due to tight coupling and complex interactions of a given concern with the rest of a program. We hypothesize that in such cases generic design can offer a viable strategy for managing complexity, enhancing the visibility of concerns. Situations where concerns become difficult to separate create opportunity to observe that genericity offers a weaker, but still useful form of SoC. We hypothesize that genericity is a natural extension to SoC-concept into the areas where SoC tends to show its limits. Therefore, both principles are intimately interrelated and synergistic. We believe the reason why genericity can penetrate software deeper than SoC is that generic design is based on the notion of unifying similarity of program structures which is less formal and rigorous than SoC. In the paper, we illustrate our points with examples from lab studies and industrial projects.

As we argue in support for the above hypothesis, we also discuss the general interplay between the two principles. We discuss engineering goals addressed by the two principles, and technical means to achieve these goals. We observe that there is an overlapping area where goals and means to separate concerns coincide with those of generic design. We compare engineering qualities of software designed based on the principles of SoC and genericity, and hint at the possibility of using both principles in a synergistic way.

We use the ART[1] (a new, refined, and simplified version of XVCL [8]) to illustrate our analysis of SoC and genericity principles. In previous papers, we described XVCL’s capability to form generic program representations to support software reuse and

[1] Adaptive Reuse Technique (ART), http://art.comp.nus.edu.sg/

evolution [14][15]. In this paper, we interpret the experiences from the earlier projects from the SoC perspective. To our best knowledge, our study is the first attempt to investigate the relation between SoC and genericity. We communicate our findings in the form of observations (or a hypothesis, at best), not claims.

Section 2 discusses the relation between SoC and genericity. In Section 3, we show an example of concerns that are difficult to separate. In Section 4, we show generic program representations for the same example. Section 5 discusses yet another example, from application software. We analyze observations in Section 6. Related work and conclusions end the paper.

2. Forms of SoC and Links to Genericity

The main goal of SoC is to deal with a concern separately from other concerns. SoC benefits magnify if we can separate a concern also at the level of software design and implementation. Modularization is the most natural conventional technique to achieve SoC, and some concerns can be nicely aligned with modular decomposition of a program. In such case, a concern is localized to a single module (a component, class, or function) or a group of modules (e.g., a component layer), exposing an abstract program interface (API) to its clients. The details of implementation of a concern become hidden behind the API [7]. This is an ideal situation from the engineering point of view. To provide full localization of a concern, management of any variability within a concern should be either hidden part of a module or supported by suitable API operations. A modularized and localized concern can be easily added to or taken out from programs, making programs more generic.

Modularization is also a simple form of generic design. Here, a similarity pattern is reflected by API. Hidden implementation part of module definition plays a role of a parameter that makes a module generic. Instantiation of such a "generic module" is done by defining a specific data representation, and implementing API operations in terms of the chosen data representation.

By localizing concerns within modules, we achieve SoC and genericity at the same time.

Concerns that cannot be localized in the above sense have a crosscutting effect on modules of a primary program decomposition. Some of the crosscutting concerns can be modularized at the extra meta-level plane using unconventional approaches such as AOP [13], AHEAD [12], or MDSOC [10].

In AOP, 'introductions' and 'advices' play the role of parameters of modules of primary decomposition. We can easily inject or take out some of the aspect code from modules, which makes modules more generic. The more module's code can we place in aspects, the more combinations of aspects can we legally and meaningfully weave into a module, the more generic a module. A similarity pattern that we unify with AOP is a functional module that can appear in multiple contexts, with or without aspects. This interpretation of AOP is in tune with goals of genericity, and we can view AOP as a kind-of generic design mechanism. In fact, AOP has been considered as a technique for building SPL architectures [1][2], which justifies the above interpretation.

MDSOC [10] and AHEAD [12] aim at building programs by composing independently defined concerns. In MDSOC, there is no primary decomposition, meaning that all the concerns are treated as equal. AHEAD promotes feature-oriented programming in which features are modeled as mathematical functions, and then programs are built and evolved by refining those functions. In both cases, an architecture of concerns from which we can build specific programs by composing concerns is a generic program representation.

Component platforms hide implementation of some of the potentially crosscutting concerns, providing transparent access to them via APIs. In JEE™, containers provide a general mechanism to access, via APIs, services whose implementation crosscuts code in the containers. Examples of such services include transaction management, persistence, security, authentication/authorization, and session management, depending on a container used [16][17]. While not completely eliminating, the JEE infrastructure makes crosscutting effect more visible and reduced to calls to the container's API operations.

The above examples illustrate that whether a given concern has a crosscutting effect or not depends on a technology used. It also depends on design decisions regarding modular decomposition and other major mechanisms used in the design of a particular program.

3. Examples of "Difficult" Concerns

If we could find a way to separate most of the concerns that matter using some technique, no doubt our software engineering problems would be much lesser than they are today. However, some of the concerns, are so tightly coupled with modules of primary decomposition and with one another that their physical separation becomes difficult. These couplings may not be fully perceived at the concept level, but as analysis of the exception handling concern shows [18] "the devil is in the details". Exception handling is an example of a "difficult" concern: "The main problem is that realistic software systems exhibit very intricate relationships involving the normal-processing code and error recovery concerns" [18]. Our experiments with EHAB (Exception Handling Application Block) on .NET™ [19] also revealed difficulties to separate exception handling from the rest of the code.

Performance in real-time systems is yet another example of a "difficult" concern. Performance has pervasive impact on many design decisions. While we can conceive and express performance concern conceptually (e.g., by documenting design decisions that have to do with performance), "physical" separation of performance from functional modules or yet other concerns that interact with performance may not be feasible.

Our next example is Buffer library, a part of java.nio.* packages in JDK 1.5. Buffer library implements containers for data in a linear sequence for reading and writing. Buffer classes differ in buffer element type, memory allocation scheme, byte ordering, and access mode. Figure 1 shows a feature diagram [20] with five feature dimensions, with specific variant features listed below a corresponding feature dimension box. Each legal combination of variant features yields a unique buffer class. We end up having many buffer classes with much similarity among them [14].

Feature dimensions are some of the “concerns” in the Buffer library domain. A developer or maintainer of the Buffer library may be interested to know: “how does an element type (or access mode) affect implementation of classes?”, “can I separate certain concerns so that specific features can be incorporated into classes, and relevant code maintained, in separation from other concerns?”.

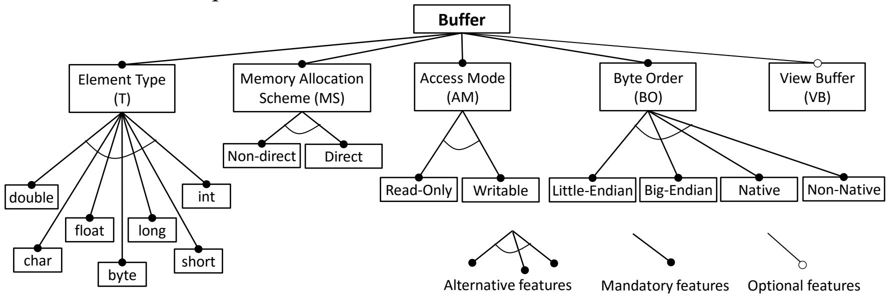


Figure 1. Features in the Buffer library

Class names reflect combination of specific features implemented into a given class. For example, DirectIntBufferR is a Read-Only buffer of integers, implemented using direct memory scheme. Classes whose names do not include ‘R’, by default are ‘W’—Writable. The Buffer library contains classes whose names are derived from a template: [MS][T]Buffer[AM][BO], where MS—Memory Allocation Scheme: Heap or Direct; T—Element Type: Int, Double, Float, Long, Short, Byte, or Char; AM—Access Mode: W (Writable, default) or R (Read-Only); BO—Byte Ordering: S (non-native) or U (native), B (Big-Endian) or L (Little-Endian). For simplicity, we can ignore VB—View Buffer, which is, in fact, yet another concern that allows us to interpret byte buffer as Char, Int, Double, Float, Long, or Short.

If successful, separation of five concerns, shown as boxes in Figure 1, would result in some “core structures” and five separately defined concerns. By composing specific features from each of the concerns into the “core structures”, we would obtain a specific buffer class implementing these features.

The number of “core structures” should be considerably smaller than the number of specific buffer classes (around 100) to make SoC worthwhile. Also, we would expect that the complexity of buffer classes represented by “core structures” plus separated five concerns would have some attractive engineering qualities, such as reduced conceptual complexity or reduced maintenance effort, over the original buffer classes in which the concerns remain intermingled.

The above view of a solution that achieves SoC again reminds generic design solution, with “core structures” playing the role parameterized representation, comprising design and code of buffer classes, and concerns playing the role of parameters that instantiate the “core structures”.

The nature of “core structures”, concerns, and composition mechanism depends on the SoC technique used. For example, in AOP, “core structures” correspond to some classes of a primary decomposition, and concerns are ‘introductions’ and ‘advises’ to be weaved into primary classes. In MDSOC, “core structures” would be treated as just yet another concern. In AHEAD, concerns are groups of features just as we described above, and “core structures” correspond to classes that are subjected to refinements.

Now we look into issues involved in trying to separate concerns in the Buffer library. To separate a concern, we must first see how a given concern affects the structure of the library and implementation of the classes that have to do with a given concern. Class naming conventions, described above, make the task of finding classes relevant to different concerns easy.

Let’s focus on the concern “buffer element type” T and observe its impact on the buffer classes.

We have no problem to do so in five classes [T]Buffer, where T is restricted to five numeric types: Int, Double, Float, Long, or Short. These classes are the same except the respective names affected by element type, highlighted in bold in Figure 2.

In the scope of five numeric types, the “buffer element type” concern can be separated by means of type parameter with Java generics [14]. (In fact, certain limitations of Java generics make type parameterization difficult even in this simple case, but here we are not concerned with language-specific limitations of Java generics. The reader can find more details in [14]).

```
public abstract class IntBuffer
…
{
  final int[] hb; // Non-null only for heap buffers
  IntBuffer(int mark, int pos, int lim, int cap, // package-private
  int[] hb, int offset)
  {  …
  }
  IntBuffer(int mark, int pos, int lim, int cap) { // package-private
  this(mark, pos, lim, cap, null, 0);
  }
  public static IntBuffer allocate(int capacity) {
  return new HeapIntBuffer(capacity, capacity);
  }
…
  public static IntBuffer wrap(int[] array, int offset, int length)
…
```

Figure 2. Fragment of class IntBuffer

Could we make “buffer element type” T an aspect, in the sense of AOP?

If we require that classes of primary decomposition are complete and can be executed, then the answer is no. Buffer element type is an integral part of any conceivable primary decomposition in the above sense, and we can’t have a buffer class without mentioning buffer element type, in either specific (such as Int or Short) or generic form.

If, on the other hand, we relax the requirement that modules of primary decomposition must be executable on their own, then we could consider a buffer element type as an aspect, provided that we can weave code related to the type at specified join points in classes of primary decomposition.

The exact points where differences among buffer classes (highlighted in bold in Figure 2) occur do not correspond to what is considered a join point in AOP. While we could place all the declarations affected by type name into ‘introductions’, and extend AOP to weave also method headers, it seems that such a solutions would not be in sync with the spirit of AOP. We rather conclude that the discussed situation is not aspect-friendly. Current form of AOP is not meant to deal with concerns that affect code in ad hoc way, at arbitrary program points. We try to strengthen this point in our further discussion.

We now extend analysis to two remaining features in the "buffer element type" concern, namely 'Char' and 'Byte'. Inspection of code reveals that class CharBuffer has a different implementation of method **toString**() than any of the numeric buffer classes. Method **toString**() converts a buffer element to a character string. In class CharBuffer, method **toString**() is trivial, just returns the buffer element, while in numeric buffer classes this method must do a proper conversion. In addition, class CharBuffer has a number of extra methods that are not needed in numeric buffer classes. The situation in ByteBuffer is analogical to CharBuffer. We see many extra methods that do not appear in numeric buffer classes or CharBuffer.

At this point, we can recap what it takes to separate concern "buffer element type" in seven classes [T]Buffer, where T is Int, Double, Float, Long, Short, Byte, or Char:

1. We must deal with varying type names and method names (e.g., 'Int' is part of method names in IntBuffer, while 'Char' is part of method names in CharBuffer).
2. We must selectively insert extra methods into certain classes.

Extra methods can be easily separated (also aspectized) and weaved into relevant classes, therefore addressing the remaining two buffer element types 'Char' and 'Byte' does not raise further complications for SoC. However, it creates a challenge for generics as extra methods cannot be represented by generic types.

The situation in groups of classes Heap[T]Buffer and Heap[T]BufferR is the same as in the group [T]Buffer.

Separation of type concern becomes more problematic when we look beyond the 21 classes in the groups [T]Buffer, Heap[T]Buffer, and Heap[T]BufferR. We see more subtle code dependencies on "buffer element type" concern. For example, in method **slice**(), buffer element type causes changes of algorithmic details as shown in Figure 3 (a constant in bold is equal to the length of a buffer element minus one, so the constant is 0 for Byte).

```
/*Creates a new byte buffer containing a shared
   subsequence of this buffer's content. */
public ByteBuffer slice() {
   int pos = this.position();
   int lim = this.limit();
   assert (pos <= lim);
   int rem = (pos <= lim ? lim - pos : 0);
   int off = (pos << 0);
   return new DirectByteBuffer(this, -1, 0, rem, rem, off);
}
```

Figure 3. Method slice() in DirectByteBuffer

We also see more drastic impact of other concerns on class implementation. For example, classes implementing 'Direct' memory allocations scheme differ a lot from analogical classes implementing 'Heap' memory allocation scheme. 'Writable' classes differ from analogical 'Read-Only' classes. The visibility of concerns becomes blurred. Trying to look for exact impact of "buffer element type" concern on class implementation becomes most difficult task, not mention separating the concern.

Still, the "buffer element type" concern seems to be the simplest case. Other concerns are even more difficult to trace and separate. Interactions between concerns are not clearly visible in class implementation. Class implementation seems to reflect the net result of concern interactions in the form that makes SoC difficult.

4. Switching Perspectives

What makes separation of “buffer element type” concern difficult is (1) much variation in the impact of different buffer element types on class implementation, and (2) subtle, ad hoc interactions between “buffer element type” and other concerns.

When dealing with “difficult” concerns, a change of the perspective from SoC to generic design is quite refreshing. Rather than looking for ways to separate concerns, we look for similarity patterns in program structures that result from interactions among combinations of concerns implemented into classes. We are still doing a fair amount of SoC, but in an approximate way, only as far as it is practically achievable.

We have the following seven groups of similar classes in the Buffer library [14]:

1. [**T**]Buffer: 7 classes at Level 1 that differ in buffer element type, **T**: Byte, Char, Int, Double, Float, Long, Short
2. Heap[**T**]Buffer: 7 classes at Level 2, that differ in buffer element type, **T**
3. Heap[**T**]BufferR: 7 read-only classes at Level 3
4. Direct[**T**]Buffer[**S|U**]: 13 classes at Level 2 for combinations of buffer element type, **T,** with byte orderings**:** **S**—non-native or **U**—native byte ordering (notice that byte ordering is not relevant to buffer element type ‘Byte’)
5. Direct[**T**]BufferR[**S|U**]: 13 read-only classes at Level 3 for combinations of parameters **T**, **S** and **U**, as above
6. ByteBufferAs[**T**]Buffer[**B|L**]: 12 classes at Level 2 for combinations of buffer element type, **T,** with byte orderings: **B**—Big-Endian or **L**—Little-Endian
7. ByteBufferAs[**T**]BufferR[**B|L**]: 12 read-only classes at Level 3 for combinations of parameters **T**, **B** and **L**, as above.

Similarities among classes manifest themselves as methods and attribute declarations that appear in different classes in similar form. Some classes contain extra methods that do not appear in other still similar classes.

It should be noticed that seven groups of similar classes are organized around concerns: each group is characterized by concerns that vary across classes in a group, and yet other concerns that are fixed.

We now proceed to the part where we apply generic design to unify similarity patterns with the help of a generative technique of the ART. As we define generic solutions using conventional programming technologies (languages and platforms) together with the ART, we call the approach *mixed-strategy*.

We start with a concrete program, or at least with some idea of a program’s component/class architecture, and its partial implementation. In case of our experiment, we start with the existing Java buffer classes. We represent each group of similar program structures (methods or classes), with unique, generic customizable structure built with the ART applied on top of Java.

We can imagine that the ART decomposes a conventional program in its own way, wrapping structures of a subject program (of any granularity and type) within the ART constructs to make them generic. It is important to notice that unification of similarity patterns occurs only at the level of an ART representation (left-hand-side of Figure 4). An executable program derived from the ART representation may still contain repetitions, if that’s required or unavoidable. Sometime repetitions are required for performance or reliability reasons. Yet other may be unavoidable given a programming technology used (e.g., on JEE™ or .NET™ platforms [21], see also [22]), and/or taking into account possibly yet other design goals a program must meet [14].

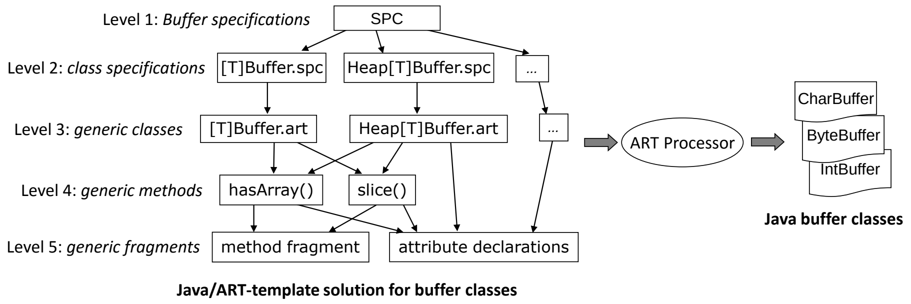

Level 1: *Buffer specifications*
SPC
Level 2: *class specifications*
[T]Buffer.spc
Heap[T]Buffer.spc
...
Level 3: *generic classes*
[T]Buffer.art
Heap[T]Buffer.art
...
Level 4: *generic methods*
hasArray()
slice()
Level 5: *generic fragments*
method fragment
attribute declarations
ART Processor
CharBuffer
ByteBuffer
IntBuffer
Java buffer classes
Java/ART-template solution for buffer classes

Figure 4. A Java/ART mixed-strategy solution for buffer classes

A building block of an ART generic program representation is called an ART template. An overall solution, a hierarchy of generic structures, is called an ART-template solution. In case of the Buffer library, we build a generic representation in combination of Java and ART, therefore we call it a mixed-strategy Java/ART-template solution, shown in Figure 4. An arrow between two ART templates: X → Y means that Y is used, after possible adaptations, to build X.

We derive all the classes in each of the seven groups of similar classes from the ART-template solution shown in Figure 4. Each of the Level 3 ART templates plays the role of a template defining a common part for all the classes in the respective group. For example, seven classes in the group [T]Buffer are derived using [T]Buffer.art as a template. ART template [T]Buffer.spc contains specifications instructing the ART Processor how to adapt [T]Buffer.art and other ART templates at levels below it to derive classes in the [T]Buffer group. We have analogical solutions in parts of the buffer ART-template solution for other six groups of similar classes.

In our example, for the sake of comparison, we designed ART-template solution so that classes produced by the ART Processor are no different from the original classes in the Buffer library.

The essence of an ART template is that it can be adapted to produce its instances (e.g., specific classes in a group). Smaller granularity building blocks for classes are defined at Level 4 (methods) and Level 5 (fragments of method implementation or attribute declaration sections). Therefore, small-granularity generic solutions (represented by the lower-level ART templates) are composed, after possible adaptations, to construct required instances of higher-level generic solutions (represented by higher-level ART templates).

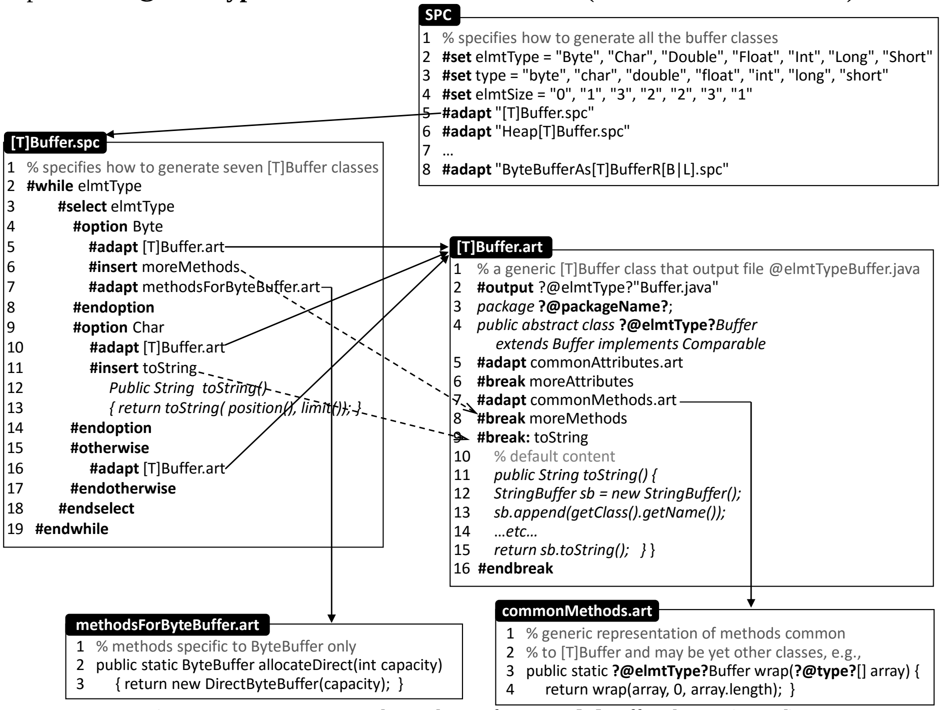
ART templates at Level 1 and Level 2 tell the ART Processor how to derive specific buffer classes from the ART-template solution. The top-most template called SPC sets up global parameters and exercises the overall control over the generation process. Specifications of controls for each of the seven groups of similar classes are at Level 2.

ART Processor interprets an ART-template solution starting from the SPC, traverses ART templates below, adapting visited ART templates and emitting the custom program. By varying specifications, we can instantiate the same ART-template solution in different ways, deriving different, but similar, program components from it. In that sense, an ART-template solution forms a generic program representation that enables reuse within a single program or across programs. In the latter case, an ART-template solution implements a concept of the SPL architecture [2].

To better see the nature of an ART-enabled generic solution and its relation to SoC, we now explain the parameterization and adaptation mechanism, which is the "heart and soul" of how the ART achieves genericity. Figure 5 shows the details of a fragment of the Java/ART-template solution shown on the left-hand-side of Figure 4.

ART variables and expressions are parameters. Typically, names of program elements manipulated by the ART, such as components, source files, classes, methods, data types, operators, or algorithmic fragments, are represented by ART expressions. Using such parameters, rather than concrete names, makes ART templates more generic, adaptable to fit into multiple contexts. For example, names and other parameters of the seven similar classes [T]Buffer are represented by ART expressions in the ART template [T]Buffer.art (Figure 5).

#set command assigns values to a variable. For example, **#set** command in line 2 of the SPC assigns values listed on the right-hand-side to a variable named elmtType. Expression **?@elmtType?** refers to one of such values (further details to follow).



SPC
```
1  % specifies how to generate all the buffer classes
2  #set elmtType = "Byte", "Char", "Double", "Float", "Int", "Long", "Short"
3  #set type = "byte", "char", "double", "float", "int", "long", "short"
4  #set elmtSize = "0", "1", "3", "2", "2", "3", "1"
5  #adapt "[T]Buffer.spc"
6  #adapt "Heap[T]Buffer.spc"
7  …
8  #adapt "ByteBufferAs[T]BufferR[B|L].spc"
```

[T]Buffer.spc
```
1  % specifies how to generate seven [T]Buffer classes
2  #while elmtType
3       #select elmtType
4          #option Byte
5             #adapt [T]Buffer.art
6             #insert moreMethods
7             #adapt methodsForByteBuffer.art
8          #endoption
9          #option Char
10            #adapt [T]Buffer.art
11            #insert toString
12               Public String  toString()
13               { return toString( position(), limit()); }
14         #endoption
15         #otherwise
16            #adapt [T]Buffer.art
17         #endotherwise
18      #endselect
19  #endwhile
```

[T]Buffer.art
```
1  % a generic [T]Buffer class that output file @elmtTypeBuffer.java
2  #output ?@elmtType?"Buffer.java"
3  package ?@packageName?;
4  public abstract class ?@elmtType?Buffer
       extends Buffer implements Comparable
5  #adapt commonAttributes.art
6  #break moreAttributes
7  #adapt commonMethods.art
8  #break moreMethods
9  #break: toString
10    % default content
11    public String toString() {
12    StringBuffer sb = new StringBuffer();
13    sb.append(getClass().getName());
14    …etc…
15    return sb.toString();  } }
16 #endbreak
```

methodsForByteBuffer.art
```
1  % methods specific to ByteBuffer only
2  public static ByteBuffer allocateDirect(int capacity)
3      { return new DirectByteBuffer(capacity);  }
```

commonMethods.art
```
1  % generic representation of methods common
2  % to [T]Buffer and may be yet other classes, e.g.,
3  public static ?@elmtType?Buffer wrap(?@type?[] array) {
4      return wrap(array, 0, array.length);  }
```

Figure 5. A Java/ART-template solution for seven [T]Buffer classes (partial)

ART parameters also play an important role of control elements that mark traces of customization changes related to a single source, that span across multiple ART templates. *This "source" often represents a concern or a specific feature within a concern*. For example, elmtType is one of the variables that mark customizations related to "buffer element type" concern. The ART Processor propagates variable values from an ART template where the value of a variable is set, down to the adapted ART templates. While each ART template usually sets default values for its variables, values assigned to variables in higher-level templates take precedence over the locally assigned default values. Thanks to this overriding rule, ART templates become generic and adaptable, with potential for reuse in unifying similarity patterns in many contexts.

Other ART commands, such as **#select**, **#insert** into **#break,** and **#while,** collectively help us design generic solutions. At the same time, they also contribute to enhancing the visibility concerns. **#select** command directs processing into one of the many pre-defined branches (called **option**s), based on the value of its control variable. With **#insert** command, we modify ART templates at designated **#break** points in arbitrary ways. ART expressions, **#select** and **#insert** into **#break** are analogous to AOP's mechanism for weaving 'advices' at specified join points. The difference is that while AOP specifies joint points in a descriptive way, **#insert**s modify ART templates in arbitrary ways, at any explicitly designated **#break** points.

#while command iterates over ART template(s), with each iteration generating similar, but with minuscule differences, program structures. **#select** command in the **#while** loop allows us to derive classes in each of the seven groups discussed in Section 3. This is a key element of the ART strategy that allows us to unify similarity patterns at the level of mixed-strategy representation (i.e., in an ART-template solution), and still have repetitions in a program that ART Processor derives from an ART-template solution.

Now, we comment on the above mechanisms in more details, referring to Figure 5 that shows a partial Java/ART-template solution for the Buffer classes. ART commands and references to ART variables are shown in bold. References to ART variables (highlighted in bold) can be embedded in the code. For example, a reference to ART variable elmtType is written **?@elmtType?** (line 4 in [T]Buffer.art), which is replaced by the variable's value during processing. Figure 6 shows generic method **slice**() from Direct[T]Buffer[S|U] classes (a specific instance of method **slice**() is shown in Figure 3). Values of variables set in SPC reach all their references in adapted ART templates. The value of variable byteOrder is set to an empty string, 'S' or 'U', in a respective **#set** command placed in one of the ART templates that **#adapt**s ART template slice.art (not shown in our figures).

slice.art

```
public ?@elmtType?Buffer slice() {
   int pos = this.position();
   int lim = this.limit();
   assert (pos <= lim);
   int rem = (pos <= lim \? lim - pos : 0);
   int off = (pos << ?@elmtSize?);
   return new Direct?@elmtType?Buffer?@ByteOrder?(this, -1, 0, rem, rem, off);
}
```

Figure 6. Generic method slice() recurring in 13 Direct[T]Buffer[S|U] classes

The **#while** loop in [T]Buffer.spc (lines 2–19) is controlled by a multi-value variable, namely elmtType. The i'th iteration of the loop uses i'th value of the variable. In each iteration, the **#select** command uses the current value of elmtType to choose a proper **#option** for processing.

#output command in [T]Buffer.art (line 2) defines the name of a file where ART Processor will emit the code for a given class.

Having set values for the ART variables, the SPC initiates generation of classes in each of the seven groups of similar classes via suitable **#adapt** commands. ART template [T]Buffer.art defines common elements found in all seven classes in the group. Five of those classes, namely DoubleBuffer, IntBuffer, FloatBuffer, IntBuffer, and LongBuffer differ only in type parameters (as in the sample method **wrap()** shown in ART template commonMethods.art). These differences are unified by ART variables, and no further customizations are required to generate these five classes from [T]Buffer.art. These five classes are catered for in **#otherwise** clause under **#select** (lines 15–17 in [T]Buffer.spc). However, classes ByteBuffer and CharBuffer have some extra methods and/or attribute declarations. In addition, method **toString**() has different implementation in CharBuffer than in the remaining six classes. Customizations specific to classes ByteBuffer and CharBuffer are listed in the **#adapt** commands, under **#option** Byte and **#opetion** Char, respectively.

The above described ART template solution is meant to illustrate our points about relationship between genericity and SoC. Evaluation of engineering qualities of ART-template solution is not in the scope of this paper. We refer the reader to the website for the discussion of trade-offs involved in applying the ART.

5. Another Example of a "Difficult" Concern

Buffer library is a very special type of a program. In this section, we show how a problem observed in the Buffer library occurs in an application software.

A Domain Entity Management System (DEMS) is contributed by ST Electronics Pte Ltd (STEE), an industrial partner in our projects. DEMS was implemented in C# that contained 117 classes covering GUI, service and database layers. DEMS involved 13 domain entities (such as *User* or *Task*) with up to 10 operations per entity (such as *Create* or *Delete*). Each combination of entity-operation is implemented by a pattern of collaborating components, two of which are shown in Figure 7. Each such pattern involves classes from four system layers. Each box in Figure 7 contains a number of classes pertaining to user interface, business logic, database communication, or database table definition layer.

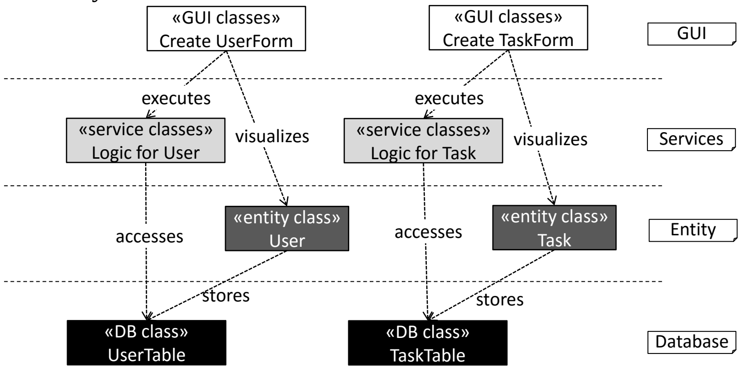


Figure 7. A recurring pattern of components

Some of the concerns in DEMS are domain entities, operations, and the four system layers shown in Figure 7.

Separating domain entity concern would mean that any entity-specific code would have to be isolated in a form that could be injected into the rest of DEMS using some composition mechanism. Operation concern is symmetric to domain entity concern, and its separation would require a similar solution.

SoC along the domain entity or operation dimension is difficult because of much differences in the requirements for specific domain entities (such as *User* or *Task*) operations that apply to different entities (such as *CreateUser* or *CreateTask*). The essence of difficulties is same as in the case of Buffer library, namely (1) much variation in the impact of different domain entities on operations, and (2) subtle, ad hoc interactions between concerns.

Now we look at the problem from the genericity perspective. There are many similarities among patterns of components implementing the same operation for different entities. There are also differences among patterns caused by different meaning of domain entities: For example, operation *Create* for a Task required different types of data entry and data validation than *Create* for a *User*. Ad hoc, induced by real-world DEMS requirements, nature of difference among patterns makes it difficult to design "generic pattern" using conventional techniques, but such a solution can be built with the ART.

Figure 8 shows an outline of DEMS as a generic C#/ART-template solution. At Level 4, each group of operations such as *CreateUser*, *CreateTask*,… has been represented by one generic operation parameterized by the respective domain entity. Similarities among different operations for the same entity (e.g., *CreateUser*, *UpdateUser*,…) are unified at Level 5. ART templates at Level 5 represent generic classes, building blocks for DEMS operations, as indicated by ART templates referenced from more than one operation (e.g., generic classes labeled with CU are reused in construction of *Create* and *Update* for various entities).

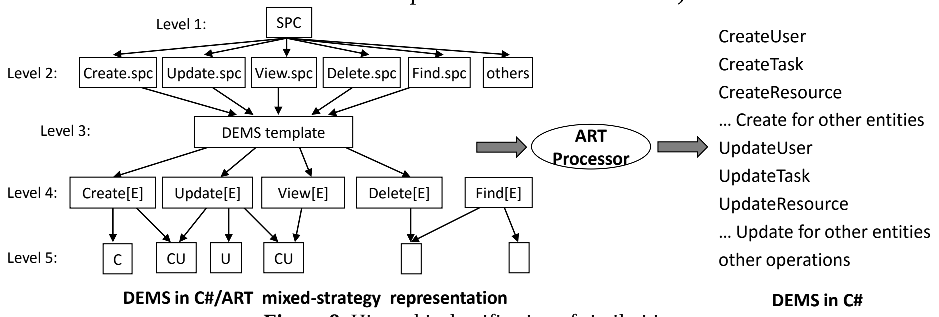


Figure 8. Hierarchical unification of similarities

The top-most template SPC contains global controls and parameter settings that specify the overall process of constructing DEMS from the templates below. 'DEMS template' at Level 3 defines the structure of the DEMS architecture, that is the organization of component patterns implementing various operations plus any other functions supported by DEMS, not discussed in this example. ART-template solution described above should be accepted as a proof of concept. Lack of space does not allow us to evaluate the cost-benefit trade-offs involved in the application of the ART. We refer the reader to the website.

6. Summary and Analysis of Observations

The examples discussed above illustrate some difficulties to achieve clean SoC, and how generic design, by looking at the problem from a different angle, achieves a weaker form of separating concerns. Now, we summarize observations, trying to distil observations that carry some more general message from those that are specific to our examples or to the use of the ART.

Both SoC and genericity are realized by a mixture of top-down and bottom-up activities.

In SoC, first intentions are conceived at the concept level, and then we try to separate concerns at the design and implementation levels. Moving from the concept level down to the design and implementation, we observe the nature of concern design/implementation, and identify yet other "lower-level" concerns.

In generic design, first we identify similarity patterns inherent in application domain concepts. In case of platforms such as JEE™ or .NET™, we also consider recurring patterns of program organization induced by a platform, as we can expect to see them in any program developed on a given platform. Then, as we design and implement a program (or work with an existing program as in our example), we observe similarities in the actual program structures. For significant groups of such similar program structures, we design generic, adaptable representation.

At times, SoC cannot be achieved at the actual program level, using features of conventional programming languages. The same is true for generic design. When conventional techniques fail to deliver a workable solution, AOP and the ART try to overcome the problem at an extra meta-level plane.

SoC at the design/implementation level increases genericity of program structures. We can view program structures as being "parameterized" by concerns. By composing concerns, we instantiate program structures in variant forms. In that sense, program structures gain genericity and reusability due to SoC. We observe this in the case of concerns that can be separated using conventional programming techniques (such as modularization or generics), as well as concerns that can be separated by supporting techniques such as AOP, MDSOC, AHEAD, JEE containers, XVCL, or ART.

In case of separable concerns, there may be still a room for generic design, as program structures parameterized by concerns may still exhibit similarity due to yet other reasons not related to given concerns. For example, we can apply AOP to separate certain aspects, but modules of primary decomposition may still contain similarities induced by similar user-level requirements. These similarities create opportunities for generic design to further simplify software solution.

We believe the above observations are general. Our discussion of "difficult" concerns, becomes necessarily dependent on a technique used for generic design in our experiments that is the ART. In both examples discussed in Sections 4 and 5, we can see an element of SoC, however we give priority to one concern at the expense of others. In the Buffer library, we bet on "buffer element type" concern. ART variables set in the top-most SPC are all related to this concern and they navigate the process of adapting ART templates below. These variables and ART constructs controlled by them enhance the visibility of the "buffer element type" concern. We can see the impact of buffer element types on the ART templates below the SPC and other ART templates adapted from there.

ART representation improves the visibility of other concerns, due to groupings of similar classes into groups, but here the SoC is less systematic.

In the DEMS example, we give priority to separating "operation" concern over "domain entity" concern. A criterion in making this decision is the extent of similarity in operations across domain entities as opposed to domain entities across operation.

Our technology-dependent experiences seem to point to observations of a general nature: The concept of similarity is less formal than the concept of cleanly separated concerns. We can identify similar program structures by top-down domain analysis, combined with bottom-up analysis of design and code (possibly supported by clone detector [23][24]). We can zoom into similarity areas that are significant. Having identified a group of similar program structures, we can always analyze the exact differences among them.

While it is relatively easy to find similarities, spotting the exact impact of "difficult" concerns is more difficult. Focusing on similarities, we do not even have to fully understand the exact nature of a given concern or complex interactions among the concerns. Instead, we stay at the level of observing the symptoms of net effect of concern interactions.

7. Related Work

Modular decomposition with information hiding [7], macros, generics in Ada or Java [4], templates in C++, other forms of parameterization such as higher order functions [5], inheritance with dynamic binding, and design patterns [6] are some of the conventional design techniques to achieve genericity. AOP [13] and MDSOC [10] support genericity by separating cross-cutting concerns. In AHEAD [12] (based on the earlier Batory's work on GenVoca), genericity is supported by feature composition and refinement. Many techniques described under the umbrella of generative techniques [25], notably meta-programming with C++ templates, achieve genericity as well as certain forms of SoC. Domain analysis [26] is essential in identifying high-level, large granularity patterns of similarity. Generic solutions unifying such patterns are most beneficial for programmer's productivity as they can significantly reduce the size and complexity of the solution. Software architectures [1][2], architectural styles [27], and patterns [2] help developers avoid repeatedly designing the same solution by providing component plug-in plug-out capability. Component platforms such as JEE™ or .NET™, provide also an infrastructure for reuse of pre-defined common services.

Code cloning has received much attention in research. As clones are closely related to the notions of similarity patterns and genericity, we discuss them in this section. Cloning has been studied in the context of re-engineering [28], refactoring [29] and clone detection [28][23][24]. In an empirical study of cloning practices Kim et al. [22]

observed that “Limitations of particular programming languages produce unavoidable duplicates in a code base”.

8. Conclusions

We considered situations where attempts to cleanly separate concerns fail. We showed that generic design can enhance the visibility of inseparable concerns, offering a weaker, but still useful form of SoC. We believe the reason why genericity can penetrate software areas deeper than SoC is that genericity, based on the notion of unifying similar program structures, is less formal and rigorous than SoC: Arbitrary software structures that exhibit enough similarity can be unified with generic program representations, using unconventional techniques such as the ART. This makes genericity technically easier to achieve than SoC.

In this paper, we made yet other observations, in the form of hypothesis rather than claims, about the general inter-play between SoC and genericity: There is an overlapping area where the goals of SoC and genericity, as well as means to achieve them, are the same. For example, type parameterization or modularization with information hiding separates a concern and achieves genericity at the same time. We can view program structures as being “parameterized” by concerns. By composing concerns, we instantiate program structures in variant forms. In that sense, program structures gain genericity and reusability due to SoC. In case of separable concerns, there may be a room for generic design to further improve engineering qualities of a program solution, as program structures parameterized by concerns may still exhibit similarity due to yet other reasons not related to given concerns. For example, we can apply AOP to separate certain aspects, but modules of primary decomposition may still contain similarities induced by similar user-level requirements.

In our future work, we conduct comparative studies to zoom deeper into the interplay between SoC and genericity principles, and further test our observations. We hope the proponents of techniques that have to do with SoC and genericity will contribute their solutions to selected problems. No doubt comparison of solutions developed using different techniques would allow us to see clearer the potentials and limitation of each of the discussed principles, and their synergistic application to form maintainable and reusable software representations.

Concerns related to different areas of a software system have different properties. For example, user requirement-level concerns, reflected in user interface and business logic software layers, tend to be less separable than software functions typically addressed by aspects [13]. An interesting area of study is development of a concern ontology. A concern ontology would help one express research results on SoC and genericity in more precise terms. We plan to extend our study described in this paper to cover possibly wide range of concern types.

References

[1] D. Musser and A. Saini, *STL Tutorial and Reference Guide: C++ Programming with Standard Template Library*, Addison-Wesley, USA, 1996.

[2] J. Bosch, *Design and Use of Software Architectures – Adopting and Evolving a Product-Line Approach*, Addison-Welsey, USA, 2000.

[3] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, Addison-Wesley, USA, 2002.

[4] J.A. Goguen, Parameterized Programming, *IEEE Trans. on Soft. Eng.*, SE-10(5), 1984, pp. 528–543.

[5] R. Garcia, J. Järvi, A. Lumsdaine, J. Siek, and J. Willcock, A Comparative Study of Language Support for Generic Programming, *Proc. 18th ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications*, 2003, pp. 115–134.

[6] S. Thompson, Higher Order + Polymorphic = Reusable, unpublished manuscript available from the Computing Laboratory, University of Kent, http://www.cs.ukc.ac.uk/pubs/1997.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns – Elements of Reusable Object-Oriented Software*, Addison-Wesley, USA, 1995.

[8] D. Parnas, On the Criteria To Be Used in Decomposing Software into Modules, *Communications of the ACM,* 15(12), 1972, pp.1053–1058.

[9] XVCL (XML-based Variant Configuration Language), http://xvcl.comp.nus.edu.sg.

[10] Aspect-Oriented Software Architecture Design Portal, http://trese.cs.utwente.nl/taosad/separation_of_concerns.htm.

[11] P. Tarr, H. Ossher, W. Harrison, and S. Sutton, N Degrees of Separation: Multi-Dimensional Separation of Concerns, *Proc. International Conference on Soft. Eng.*, *ICSE'99,* Los Angeles, 1999, pp. 107–119.

[12] E.W. Dijkstra, On the Role of Scientific Thought, *Selected Writings on Computing: A Personal Perspective*, Springer-Verlag, New York, 1982, pp. 60–66.

[13] D. Batory, J.N. Sarvela, and A. Rauschmayer, Scaling Step-Wise Refinement, *Proc. Int. Conf. on Soft. Eng., ICSE'03*, 2003, Portland, Oregon, USA, pp. 187–197.

[14] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.M. Loingtier, and J. Irwin, Aspect-Oriented Programming, *European Conf. on Object-Oriented Programming,* Finland, 1997, pp. 220–242.

[15] S. Jarzabek and S. Li, Unifying Clones with a Generative Programming Technique: A Case Study, *Journal of Software Maintenance and Evolution: Research and Practice*, 18(4), 2006, pp. 267–292.

[16] S. Jarzabek, *Effective Software Maintenance and Evolution: Reuse-based Approach*, Taylor & Francis, USA, 2007.

[17] A. Mesbah and A.V. Deursen, Crosscutting Concerns in JEE Applications, *Proc. 7th IEEE International Symposium on Web Site Evolution, WSE'05,* Budapest, Hungary, Sept. 2005, pp. 14–21.

[18] Private communication with Ali Mesbah and Arie van Deursen, authors of [16].

[19] F. Filho, N. Cacho, E. Figueiredo, R. Maranhao, A. Garcia, and C. Rubira, Exceptions and Aspects: The Devil is in the Details, *Int. Symp. Foundations of Soft. Eng*., *FSE'06*, 2006, USA, pp. 152–162.

[20] Exception Management Architecture Guide ver 1.0. Microsoft Patterns & Practices, 2003, Available at: http://www.usol.com/~joe/Exception%20Management%20-%20EntLib.pdf.

[21] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, Technical Report, CMU/SEI-90-TR-21, SEI, CMU, Pittsburgh, 1990.

[22] J. Yang and S. Jarzabek, Applying a Generative Technique for Enhanced Reuse on JEE Platform, *4th Int. Conf. on Generative Programming and Component Engineering*, *GPCE'05*, 2005, Tallinn, pp. 237–255.

[23] M. Kim, V. Sazawai, D. Notkin, and G. Murphy, An Ethnographic Study of Code Clone Genealogies, *Proc. Euro. Soft. Eng. Conf. and Int. Symp. Foundations of Soft. Eng.*, 2005, Portugal, pp. 187–196.

[24] H.A. Basit and S. Jarzabek, A Data Mining Approach for Detecting Higher-Level Clones in Software, *IEEE Trans. Softw. Eng.*, 35(4), 2009, pp. 497–514.

[25] T. Kamiya, S. Kusumoto, and K. Inoue, CCFinder: A Multi-linguistic Token-based Code Clone Detection System for Large Scale Source Code, *IEEE Trans. Soft. Eng*., 28(7), 2002, pp. 654–670.

[26] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, USA, 2000.
[27] R. Prieto-Diaz, Domain Analysis for Reusability, *Int. Comp., Soft. & Appl. Conf., COMPSAC'87*, 1987, Tokyo, Japan, pp. 23–29.
[28] M. Shaw and D. Garlan, *Software Architecture: Perspectives on Emerging Discipline*, Prentice Hall, USA, 1996.
[29] I. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, Clone Detection using Abstract Syntax Trees, *Proc. Int. Conf. on Software Maintenance, ICSM98*, 1998, pp. 368–377.
[30] M. Fowler, *Refactoring - Improving the Design of Existing Code*, Addison-Wesley, USA, 1999.

III. Embedded and Web Systems

Chapter 9

Experimental Real-Time Arinc 653 Based Pitch Angle Control Application

1. Introduction

Modern airliners increasingly use computer systems to improve operational features of almost all onboard systems. Gradually, avionic computer programs are executed under real-time operating systems (RTOS) [6, 7, 8, 15, 16]. This involves some new software development techniques which have to be applied into aviation.

Recently, due to progress in engineering, airborne real-time systems have been evolving from the so-called “federated structure” to something new - Integrated Module Avionics (IMA) [2, 3, 4, 5, 9, 13]. The IMA concept has been introduced through European research projects: PAMELA, NEVADA and VICTORIA. The result was the first generation of IMA (IMA1G), currently onboard A380, A400M and B787 aircraft. Following the IMA concept, modern onboard avionic subsystems (software applications) should be grouped into a limited set of standard microprocessor units. The microprocessor units and other electronic devices should communicate via standard network interface - Avionics Full Duplex Switched Ethernet (AFDX) [2, 3]. So far physically and logically separated (federated) avionic units were going to be converted into groups of real-time applications controlled by real-time operating systems. The current implementation of IMA covers a limited range of aircraft functions but shows that it may bring some significant benefits: aircraft weight reduction and lowered maintenance costs.

Next step of development of integrated avionics was to define a scalable, reconfigurable fault-tolerant driven and secure new avionics platform, called Distributed Modular Electronics (DME) which announces IMA(2G) family of devices. But before it happened it had had to be tested and confirmed if DME units could have been effectively used as hard real-time applications platforms. The research reported in this paper, done within the European SCARLETT program [20], involved the new DME units evaluation as a hardware platform for real time applications The preliminary research results acquired in this area were published in [14, 15, 16]. Next sections of the paper are organized as follows. Firstly, Distributed Modular Electronics concept as well as the ARINC specification 653 are briefly presented. Next, an experimental aircraft Pitch Angle Control Application (PACA) is introduced (an illustrative example of the ARINC 653 based hard real-time avionic control system). The final part of the paper includes some application development remarks as well as final system tests reports.

2. Distributed Modular Electronic concept

Distributed Modular Electronics (DME) goes directly from IMA concept and has been developed very dynamically during last years. DME should improve the following features of avionics systems:

- Scalability, portability and adaptability,
- Fault tolerance and reconfiguration capabilities,
- Number of standardized electronic module types could be minimized,
- The full range of avionics function should be supported.

DME oriented avionics provides few basics computer units as platforms for real-time software:

- CPM (Core Processing Module) offers generic computation capability, an AFDX or other communication,
- REU (Remote Electronics Unit) is an electronics box dedicated to a specific task and is mounted geographically close to where this task occurs. REUs are not generic units and therefore not part of the IMA perimeter. However, the interfaces of REUs to the IMA world shall be standardized,
- RDC (Remote Data Concentrator) is a type of equipment which supports the exchange of information between sensors/actuators (digital, discrete and analogue data) and aircraft digital communication networks (ADCN). The RDCs are located in pressurized areas close to sensors and effectors, which may be potentially remote from the associated processing resources rather than in the avionics bay,
- RPC (Remote Power Controller) is a power switching "unit based on SSPCs (Solid State Power Controllers). They are able to control the switching of loads with current limitation, self-test, leakage protection, etc. so they are not strictly equivalent to relays."

Although the individual DME hardware devices may communicate with some external devices using a wide set of interfaces, such as CAN, RS or Field Bus, the main medium of the inter-DME units' communication should be AFDX [2, 3] - a redundant and reliable Ethernet network developed and standardized by the European constructors of the avionics which equip the Airbus A380. The switches of the AFDX network are star couples, provide packet forwarding, and they also have additional features to guarantee the timing and bandwidth allocation of the entire network, but they only use multicast addresses (Virtual Links). The AFDX internal message format should follow ARINC specification 664 [3]. The software modules executed on the hardware units should be developed according to ARINC specification 653 [4].

Within the SCARLETT project, the aforementioned set of hardware modules was configured into several laboratory demonstrators. Their task was to assess whether the developed DME units might be effectively used in the next generation of IMA.

3. ARINC Specification 653P1-2

One of the main IMA concepts assumes the reduction of the number of the individual microprocessor units installed on-board. However, this also entails a new paradigm in avionics development. The group of federated applications which have been executed up to now on separate microprocessor units (communicating by means of ARINC standard 429 based devices [1], for example) must become a set of real-time processes executed on one microprocessor. This is managed by a specialized real-time operating system and communication by means of a specialized Ethernet computer network. Provided that the operating system offers a standard API and fulfills safety requirements, this proposed solution significantly broadens the portability of avionic applications and makes it possible to develop and certify hardware and software independently.

Partitions

The IMA assumes that a set of time-critical and safety-critical real-time applications (avionics units) may be executed on one microprocessor module.

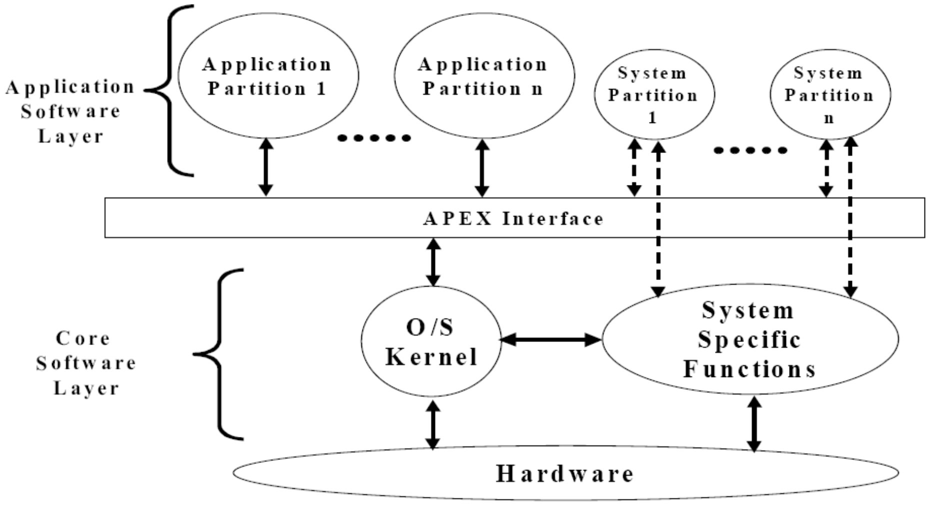


Figure 1. Logical real-time operating system structure created according to ARINC 653P1-2 specification[4].

To cope with this level of criticality, new real-time operating system architecture was suggested. ARINC 653P1-2 [4] defines the generic system structure. Figure 1 shows the RTOS logical structure which was suggested for avionic systems.

The key concept introduced of the specification is partition. It constitutes a kind of container for an application and guarantees that the execution of the application is both spatially and temporally isolated. The partitions have been divided into 2 categories: application partition and system partition. The application partitions are dedicated to executing avionics applications. They can exchange data with the environment by means of specific interface – APEX (APplication/EXecutive). The system partitions are optional and their main role is to provide services that had not been predicted in APEX, such as device drivers or fault management.

Hardware-Software Module Architecture

The ARINC 653P1-2 also includes some recommendations regarding the microprocessor module architecture for the specialized real-time operating system. The general schema of the architecture is presented in figure 2.

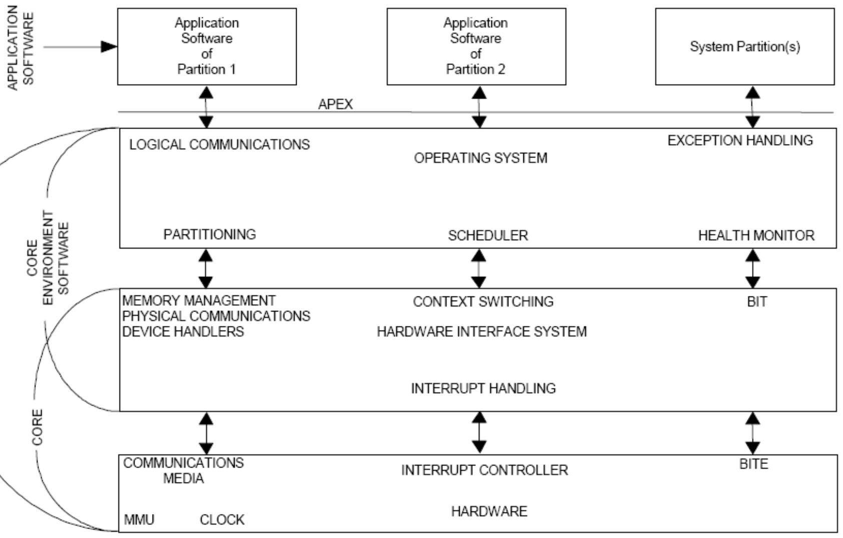


Figure 2. Logical real-time operating system structure created according to ARINC 653P1-2 specification[4].

Each module may include one or more microprocessors. The hardware structure may require some core operating system modification but not the APEX interface. All the processes that belong to one application partition (real-time tasks) must be executed on one microprocessor and it is forbidden to allocate them to different microprocessors within the module or between modules. The application program should be portable between processors within the module and between modules without any modifications to the interface of the operating system core. The processes that belong to one partition may be executed concurrently. A separate partition-level scheduling algorithm should be responsible for this. Inter-application (partition) communication is based on a ports and channels concept. The applications do not have the information about which partition the receiver of data is being executed on. They send and receive data via ports. The ports are virtually connected by channels which are defined in a separate level of system development.

An important element of the module should also be a Health Monitor (HM). It is an operating system component that ought to monitor hardware as well as the operating system and application faults and failures. Its main task is to isolate faults and prevent failure propagation. As an example, the HM is permitted to restart a partition when it detects an application fault.

The temporal isolation of each partition has been defined as follows, a major time frame is defined for each module. It is activated periodically. Each partition receives one or more time partition windows to be executed within this major time frame.

Generally, time partition windows constitute a static cyclic executive [6]. Real-time tasks executed within the partition can be scheduled locally according to a priority-based policy. The order of the partition windows is defined in a separate configuration record of the system.

In general, the applications that reside within partitions may be developed by separate application providers. Thereafter a separate role in the IMA system development process has been proposed. This person or organization, “an integrator”, has to collect the data regarding resources, timing constraints, communication ports and exceptions defined in each partition. Then the collected data is transferred into configuration records. The configuration record for each module is an XML document interpreted during compilation and consolidation of the software.

APEX Interface

The main part of ARINC 653P1-2 is the APEX interface definition. The APEX makes it possible to create platform-independent software that fulfils ARINC 653 requirements. The three main components of the interface are: real-time application creation and maintenance; partition management; intra- and inter-partition communication.

The application may be constructed as a set of (soft or hard) real-time processes, which are scheduled according to their priorities. It is possible to develop both an event- and a time-driven process activation policy.

The APEX interface provides a separate set of functions that enables the user to determine the actual partition mode and change it. The application may start the partition after the creation of all the application components or monitor the actual partition mode. It is also possible to restart the partition.

The synchronization of processes that belong to one partition may be achieved by the appropriate application of counting semaphores and events. The inter-process communication within the partition (intra partition communication) is made possible by means of APEX buffers (shared message queues) and APEX blackboards (shared variables). Inter partition (application) communication is based on queuing port and sampling port communication units. The queuing port provides an inter-partition message queue, whereas the sampling port makes it possible to share variables between the ports. During system integration, the ports are connected by means of channels defined in the system configuration tables. The ports may be applied for communication with: other partitions, device drivers within the module or to exchange data between modules (by means of AFDX network interfaces).

4. Pitch Control Hard Real-Time Application

The paper reports some author’s work under aviation software module which satisfies ARINC 653, ARINC 664 requirements and is intended to run on DME modules. The application was created as a piece of software realizing selected control functions of an autopilot system. It demonstrates that software prepared according to ARINC 653, ARINC 664 rules could be used to control aircraft flight. Control algorithm of pitch angle control channel has been selected as an example and finally coded inside the application.

General Application Specification

The application ought to demonstrate whether it is possible to use IMA philosophy to control an aircraft. The pitch angle control channel was selected as a sample. The application tests if it is possible to use two synchronized actuators deflecting an aircraft elevator's surfaces. A flight controller computes the position of the elevator on the basics of flight parameters, pilot input, and implemented control procedures. The application communicates with actuator controllers and units which collect data from indicators via ADFX bus. Figure 3 shows an example of the distribution of the application between hardware modules.

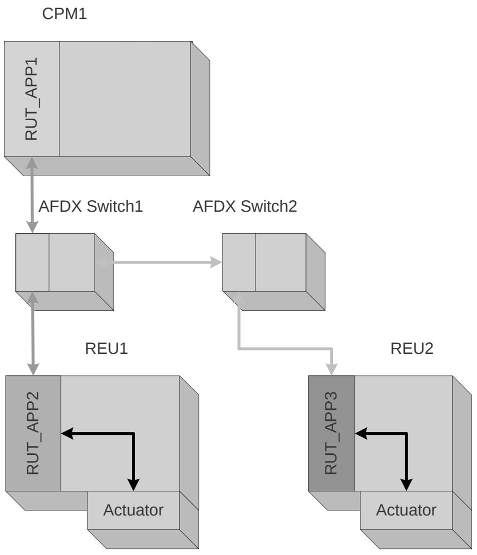


Figure 3. Example application distribution scenario.

A part of the control application is executed on a CPM module. Two other parts are allocated to two REUs. The REUs are directly connected to elevator actuators. The control application modules communicate via AFDX.

Control System Project

To fulfill the application requirements the following control system architecture was proposed. The control application would be a Pitch Angle Control Application (PACA) controlling two actuators (brushless motors) loaded externally (an elevator). Each actuator is controlled by a separate cascade of controllers, as in fig. 4. The single actuator control system includes an internal current control loop, a velocity control loop, and a position control loop. The Flight Control Algorithm is a superior module that generates the position demand signal for both actuator control subsystems. It collects signals from the aircraft simulator, pilot simulator and actuator. The real-time aircraft simulator makes it possible to adjust the dynamics of the PACA to realistic values. It reflects typical airliner behavior.

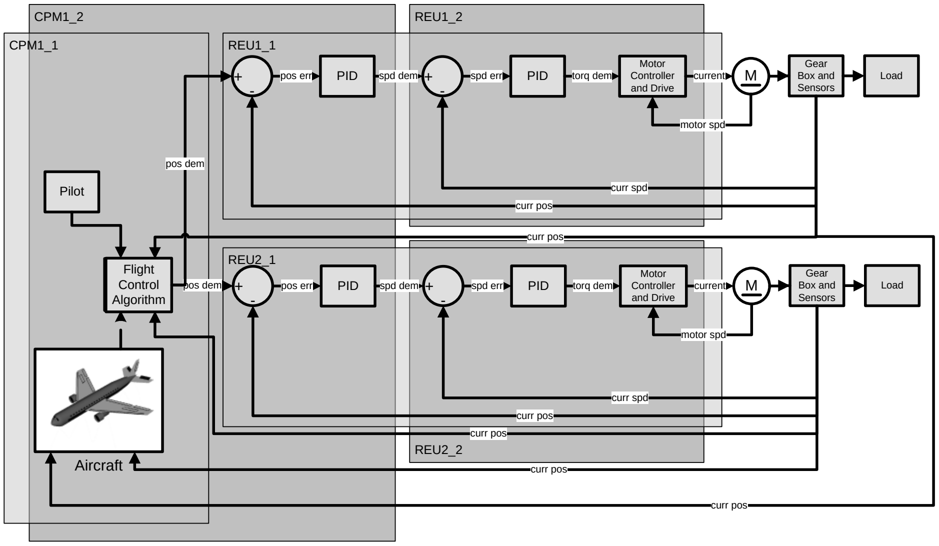


Figure 4. Pitch Angle Control System Architecture.

Control Application Distribution Scenarios

One of the most important goals of reported research was to evaluate the quality of distributed control applications, where some parts of the application are housed on different devices. Therefore the Pitch Angle Control Application was developed with the intention of distributing some of its parts to separate hardware modules. Figure 4 shows the possible control application distribution variants proposed by the demonstrator's developers. In the first variant there are CPM1_2 + REU1_2 + REU2_2 units, the Pilot, Aircraft, Flight Control Algorithm, and two position controllers housed on the CPM module, whereas control algorithms controlling the rate of the elevator are housed on separate REUs (compare fig. 4). In the second variant (CPM1_1 + REU1_1 + REU2_1), the CPM module executes the superior part of the control system (Pilot, Flight Control Algorithm, and Aircraft), with the position and velocity controllers housed on REUs (compare fig. 4).

Control Application Configuration Assumptions

Bearing in mind application specification and the likelihood of different allocation scenarios of the control application components, there are following detailed assumptions regarding the developed application taken:

- The position controller must be movable. It should be possible to install it both on CPM or REU hardware modules. Therefore it should belong to a separate ARINC 653 partition.

- All data packages sent between partitions should be in accordance with ARINC 664P7 [3]. This way, the partition application does not have to be changed even if some of the partitions will be moved to other hardware units. ARINC 664P7 messages will be ready to be sent via the AFDX network.
- Some verification procedures should be built into the application software. They should provide information about the quality of control system and the soundness of the system structure.
- The application should be developed according to ARINC 653P1-2 for 2 target operating systems: VxWorks 653 [17, 18, 19] and PikeOS [10, 11, 12].

Pitch Angle Control Application

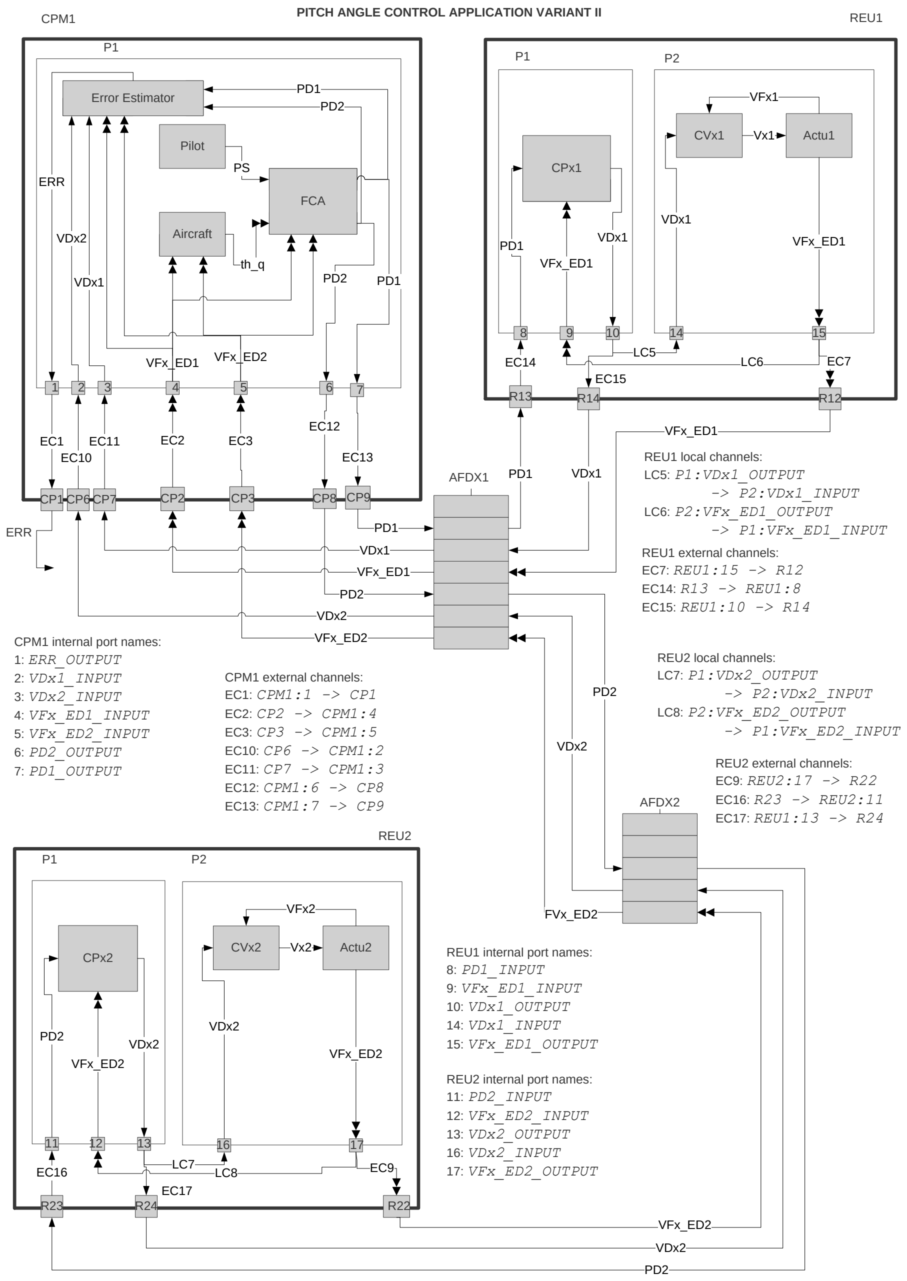
The PACA software consist of both controllers and real-time simulators of the hardware units which were finally replaced by real devices. Figure 5 includes this structure. Figure 5 also depicts the PACA structure from figure 4 in a schema based on ARINC 653. The second variant of module allocation has been chosen (CPM1_1 + REU1_1 + REU2_1; compare sec. 4.3 and fig. 4). The application is hosted on three DME units: one CPM and two REUs. P1 partition of CPM1 module includes some real-time simulators of Pilot and Aircraft. It also includes a Flight Control Algorithm (FCA) block that collects signals from Pilot, Aircraft and actuator modules and produces the desired pitch angle signals for controllers. The last module built into the CPM1's P1 partition is an Error Estimator. It makes it possible to monitor both communication channels and the quality of control system during the system run-time. The Error Estimator, Pilot, Aircraft and FCA modules are separate real-time tasks.

The first (P1) REUs' partitions include the position controller algorithms (CPx1 and CPx2), running as a separate real-time task. The second (P2) partitions of REUs include the velocity controller modules joined with actuators attached to the first (CVx1+Actu1) and second (CVx2+Actu2) control loops.

For the intra-partition communication ARINC 653 blackboards were applied, whereas the inter-partition communication is based on ARINC 653 sampling ports and channels (compare section 3.3). Figure 5 includes both ARINC 653 port names and communication channels defined. The port names are the only reference to the application communication interface. The same port names exist in the application configuration record and this makes it possible to connect these ports by means of the channels.

The Pitch Angle Control Application was developed to fulfill the timing constraints shown in fig. 6. Major application frame and partition windows were defined according to ARINC 653 and encoded in the application's XML configuration file. Table 1 includes all the PACA tasks' ARINC 653 real-time parameters. These timing constraints originate from control engineering needs. The CPM's major time frame includes a set of regions dedicated to applications which are executed on CPM1 apart from P1 partition. The HARD attribute attached to each of the real-time tasks instructs the ARINC 653 Health Monitor (which is built into the operating system structure) that if any task misses its deadline, the core operating system must be informed about it. This in consequence, imposes the operating system to take an appropriate action. The

Health Monitor procedures may even reload the whole partition that signals the missing timing constraints event.



CPM1 internal port names:
1: *ERR_OUTPUT*
2: *VDx1_INPUT*
3: *VDx2_INPUT*
4: *VFx_ED1_INPUT*
5: *VFx_ED2_INPUT*
6: *PD2_OUTPUT*
7: *PD1_OUTPUT*

CPM1 external channels:
EC1: *CPM1:1 -> CP1*
EC2: *CP2 -> CPM1:4*
EC3: *CP3 -> CPM1:5*
EC10: *CP6 -> CPM1:2*
EC11: *CP7 -> CPM1:3*
EC12: *CPM1:6 -> CP8*
EC13: *CPM1:7 -> CP9*

REU1 local channels:
LC5: *P1:VDx1_OUTPUT -> P2:VDx1_INPUT*
LC6: *P2:VFx_ED1_OUTPUT -> P1:VFx_ED1_INPUT*

REU1 external channels:
EC7: *REU1:15 -> R12*
EC14: *R13 -> REU1:8*
EC15: *REU1:10 -> R14*

REU2 local channels:
LC7: *P1:VDx2_OUTPUT -> P2:VDx2_INPUT*
LC8: *P2:VFx_ED2_OUTPUT -> P1:VFx_ED2_INPUT*

REU2 external channels:
EC9: *REU2:17 -> R22*
EC16: *R23 -> REU2:11*
EC17: *REU1:13 -> R24*

REU1 internal port names:
8: *PD1_INPUT*
9: *VFx_ED1_INPUT*
10: *VDx1_OUTPUT*
14: *VDx1_INPUT*
15: *VFx_ED1_OUTPUT*

REU2 internal port names:
11: *PD2_INPUT*
12: *VFx_ED2_INPUT*
13: *VDx2_OUTPUT*
16: *VDx2_INPUT*
17: *VFx_ED2_OUTPUT*

Figure 5. Pitch Angle Control Application ARINC 653 based structure.

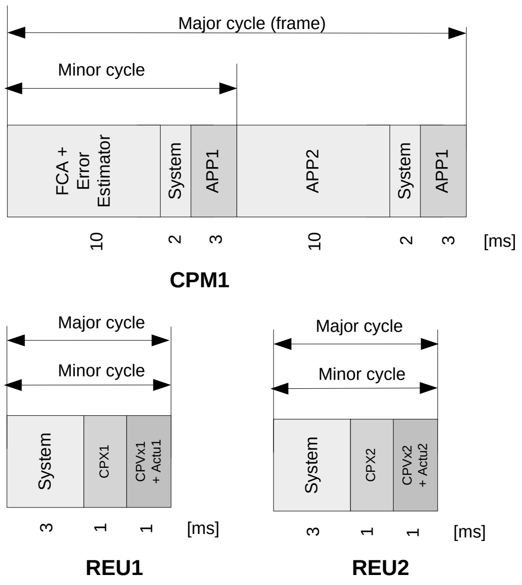


Figure 6. Pitch Angle Control System Partition Timing

PACA Real-Time Analysis

As it was mentioned before, the PACA timing restrictions were forced by control engineers which specified the system. P1 and P2 partitions at REUs acquired 2 ms time frames for their computations and are activated every 5 ms. This timing constraints guarantee sufficient frequency (200Hz) of PID algorithm repetition which in consequence guarantees the sufficient quality of control. P1 partition on CPM1 module is 4 times “slower” than others. To authors knowledge algorithms computed in this partition may produce the results in such rate without any effect on the control system quality. This makes it possible to save some computational time for other application that will be installed on the same hardware module.

All of algorithms applied in the PACA are controllers or simplified numerical procedures which solve some differential equations. During the system development the worst case computation time analysis for each of the algorithm was conducted [6, 7]. It was proved and experimentally checked that the algorithms can meet the aforementioned timing constraints.

The real-time task parameters of all of the PACA is depicted in tab. 1. The local real-time tasks priorities (which were defined within the partitions) reflect the order of the computations the task should follow. This approach is essential especially in P1 partition. It is expected that the Pilot and Aircraft real-time tasks should finish their computations and produce their results before the FCA task starts.

All the communication mechanisms applied in the PACA are both shared variables and monitors. This solves the mutual exclusion problem. The shared variable access (at the operating system level) is conducted according to the priority inheritance protocol

[6, 7]. It is easily to notice that the developed PACA communication structure prevents form the deadlock phenomena, too.

Table 1. PACA ARINC 653 Real-Time Tasks Parameters

Tasks	Real-Time Tasks Parameters				
	Stack Size	**Base Priority**	**Period [ms]**	**Time Capacity [ms]**	**Deadline**
Pilot	4096	120	20	2	HARD
Aircraft	4096	119	20	4	HARD
FCA	4096	118	20	2	HARD
ERROR ESTIM.	4096	117	20	2	HARD
CPx1	4096	110	5	2	HARD
CPx2	4096	110	5	2	HARD
VPx1	4096	109	5	1	HARD
VPx2	4096	109	5	1	HARD
Actu1	4096	108	5	1	HARD
Actu2	4096	108	5	1	HARD

Built-in Self-Testing Procedures

According to application specification, the PACA should provide a set of test procedures informing the user about the quality of the system before or during runtime. Therefore the following extensions of the basic PACA structure depicted in fig. 4 were applied.

- A separate error (ERR) port was included in the CPM1s' P1 partition structure.
- A new Error Estimator real-time task was introduced in the CPM1s' P1 partition.
- It has been decided that the quality of the PACA's subsystem service would be run simultaneously with the control procedures.
- The quality of service procedures has been divided into two subsystems:
    - The channel connection detector permanently monitors the channels of the system and indicates whether all links are properly connected. This subsystem guarantees that all system components send and receive data from the proper ports and software modules. The channel connection detector checks whether all channels are configured according to the assumed structure. During the runtime of the application, apart from control application data, a separate set of values is sent via the channels. Some additional procedures included in application control blocks make it possible to detect whether the application's ports receive data from the assumed sources. The detector also makes it possible to reveal data transmission faults. It produces a bit word, where each bit value means the correctness of the related channel. If the bit value equals 0 the channel works properly. If not, the channel is badly established or the function block connected to the channel produces incorrectly formulated data packages.

- The control system error detector signals to the system operator that the quality of control is below the assumed acceptable level. It may reveal some problems with communication or may suggest that the control system's parameters should be refined. Each single actuator control system is monitored by a separate control procedure built into the Error Estimator block. This procedure collects all the possible signals from the PACA modules and assesses the quality of control by: 1) detection of the actuator's angular velocity oscillations, 2) signalization of the elevator's position error that exceeds the assumed threshold value.

Finally, the control system error detector produces 2 major and 4 minor values. The major values describe the quality of control of the appropriate actuator. In general, if they both have the value of 0, both control loops work correctly. If they have values between 1 and 3, they include the error code of the monitored control loop. Table 2 includes the error codes' interpretation.

Table 2. Error codes signalized by PACA diagnostic system

Error Code	Error Code Interpretation
0	Control system is working correctly
1	Deflection velocity oscillations in occurs
2	Tracking error occurs
3	Both tracking error and control system oscillations occurs

PACA Tests

The complete PACA system was finally integrated with new DME IMA2G modules developed within SCARLETT program. Figure 7 shows the mapping between the partitions and new hardware DME modules.

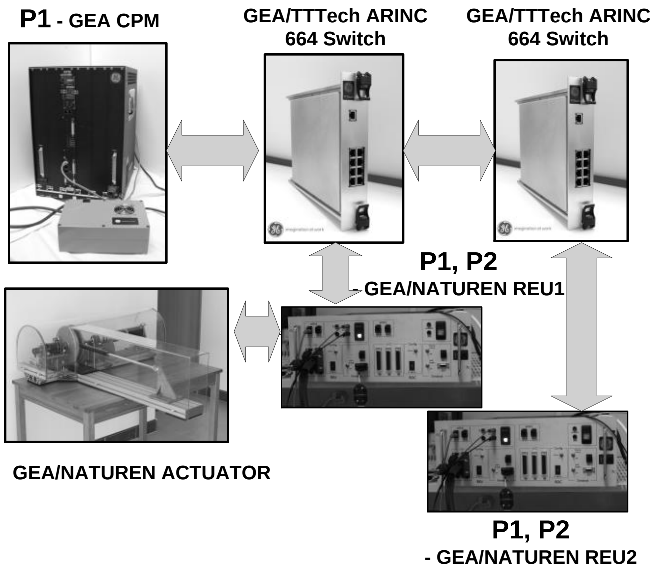

P1 - GEA CPM
GEA/TTTech ARINC 664 Switch
GEA/TTTech ARINC 664 Switch
P1, P2 - GEA/NATUREN REU1
GEA/NATUREN ACTUATOR
P1, P2 - GEA/NATUREN REU2

Figure 7. DME modules developed where prototype Pitch Angle Control Application was integrated with.

P1 partition software module is integrated with the CPM (Core Processing Module) developer by General Electric Aviation – GEA. REUs' P1 and P2 partitions software is being integrated with REUs (Remote Electronic Unit) modules developed by GEA and NATUREN. ARINC 664-compatible switches developed by TTTech and GEA were responsible for inter-module communication. One of the REU modules was being integrated with a laboratory set including a brushless motor, a load and a power controller developed by NATUREN. The integrated system (the demonstrator) was being applied to the evaluation of both hardware and software module prototypes.

There were three groups of tests conducted with PACA application. The first group of tests assessed the application from a control engineering point of view. The second group of tests covered execution of built-in self-test procedures of the PACA. The third group of test included the application of VxWorks 653 analysis tools for PACA timing and communication evaluation.

A control engineering-based application evaluation was conducted as follows, the application's goal was to perform in an AFDX environment and to control actuators using data incoming from the network. To guarantee more realistic test conditions, the actuator controller's software was put into a simulated flight control environment. A real-time aircraft simulator was used to obtain realistic values from the application. It also produced a reference signal for the Flight Control Algorithm module (fig. 4 and 5).

Figure 8 includes the result of the pitch control system evaluation. A position tracking by the motor as well as its velocity were examined during experiments. As a result, it was stated that the system met the typical requirements of position tracking control systems. The system latency was below 40 [ms]. Both CPM and REU hardware module were able to effectively serve the software applications loaded on them.

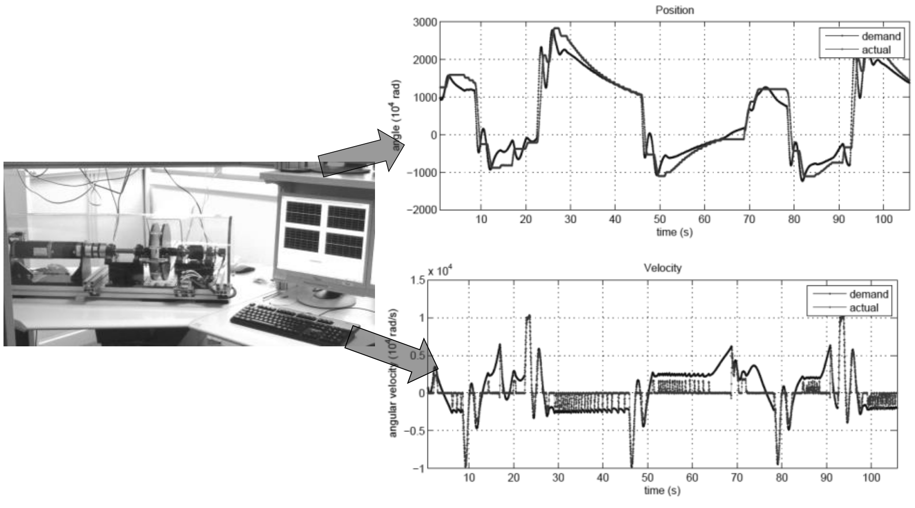


Figure 8. Position and velocity tracking tests of prototype Pitch Angle Control Application.

A built-in self-test subsystem evaluation was performed as follows, both the channel connection detector subsystem and control system error detector were tested in detail during long-term tests of the PACA. All possible channel malfunctions were simulated and properly detected. Similarly, a low quality (from the control engineering point of view) PACA was executed. The control system error detector successfully signaled the lower quality control signals.

A VxWorks 653 analysis tools-based system evaluation was conducted as follows, during the real-time PACA execution, the a System Viewer toolset was applied to collect standard timing and communication events that occurred during the monitoring session. Thorough analysis of the System Viewer data made it possible to confirm that the timing and communication requirements were fulfilled.

The final report of SCARLETT program concluded that the hardware modules developed within the program could be potentially applied as a control system applications platform.

5. Conclusions

An experimental hard real-time avionic control and ARINC 653-compatible application development was a main subject of the paper. The paper mentions the general objectives of Distributed Modular Electronics and specification ARINC 653. The main part of the paper includes a report of the PACA development and evaluation. It covers system specification and the most important project development perspectives: control system structure, ARINC 653-based application structure, real-time timing parameters, and built-in self-testing procedures.

The experimental PACA application has been successfully integrated with a new DME modules. It proved that hard real-time control application can be effectively loaded into new DME modules. Currently the DME modules are tested and certified for the use in a future aircrafts. The IMA concept is going to be introduced in broaden aircraft developers.

The future author’s work will concentrate on the development and evaluation of new self-testing procedures for ARINC 664/653 based applications, mentioned in section 4.7. The research done during the software tests as well as the experience exchanged between the researches and engineers showed that the ARINC 653 software should be equipped with some more effective software modules which would supervise the control system state and modify the system behavior during the malfunction detection.

Acknowledgments

Research reported in the paper is funded by SCARLETT 7th European Framework Project, Grant Agreement No. FP7-AAT-2007-RTD-1-211439. Some of hardware components used in the research published within this paper were financed by the European Union Operational Program - Development of Eastern Poland, Project No. POPW.01.03.00-18-012/09. Ralph McGaw assisted in proofreading the English version of this text.

References

[1] *ARINC 429*: Mark 33 Digital Information Transfer Systems (DITS), 1996.
[2] *AFDX: The Next Generation Interconnect for Avionics Subsystems*, Avionics Magazine Tech. Report, 2008.
[3] *Aircraft Data Network Part 7 - Avionics Full Duplex Switched Ethernet (AFDX) Network*, ARINC Specification 664p7 2005.
[4] *Avionics Application Software Standard Interface Part 1-2, ARINC Specification 653p1-2*, 2005.
[5] P. Bieber, E. Noulard, C. Pagetti, T. Planche, F. Vialard, Preliminary Design of Future Reconfigurable IMA Platforms, *ACM SIGBED Review - Special Issue on the 2nd International Workshop on Adaptive and Reconfigurable Embedded Systems (APRES'09)*, Volume 6 Issue 3, October 2009.
[6] A. Burns, A. Wellings, *Real-Time Systems and Programming Languages*, Addison Wesley, 2001.
[7] G. Buttazzo, *Hard Real-Time Computing Systems – Predictable Scheduling Algorithms and Applications*, Kluwer, 2002
[8] B. Dołega, G. Kopecki, Fault Detection and Isolation in Attitude and Heading Reference Systems For Fly-By-Wire Control System For General Aviation Aircraft, *SAE Paper*, No. 2006-01-2415, 2006.
[9] P. Parkinson, L. Kinnan, *Safety-Critical Software Development for Integrated Modular Avionics*, Wind River White Paper, 2007.
[10] *PikeOS Fundamentals*, Sysgo AG, 2009.
[11] *PikeOS Tutorials*, Sysgo AG, 2009.
[12] *PikeOS Personality Manual:APEX*, Sysgo, 2009.
[13] J. W. Ramsey, Integrated Modular Avionics: Less is More Approaches to IMA will save weight, improve reliability of A380 and B787 avionics, *Avionics Magazine*, 2007. http://www.aviationtoday.com/av/categories/commercial/8420.html.
[14] T. Rogalski, A Conception of Voice Guided General Aviation Aircraft, *Aircraft Engineering And Aerospace Technology, An International Journal*, Vol 80 No. 6 2008, Emerald Group Publishing Limited, 2008.
[15] T. Rogalski, S.Samolej, A. Tomczyk: ARINC 653 Based Time-Critical Application for European SCARLETT Project, *AIAA Guidance, Navigation, and Control Conference,* 08 - 11 August 2011, Portland, Oregon, USA, paper number: AIAA 2011-6684, available on www.aiaa.org website.
[16] S. Samolej, A. Tomczyk, J. Pieniążek, G. Kopecki, T. Rogalski, L. Rolka, VxWorks 653 based Pitch Control System Prototype, *Development Methods and Applications of Real-Time Systems,* L. Trybus, S. Samolej eds., WKŁ 2010, (in. Polish).
[17] *VxWorks 653 Configuration and Build Guide 2.2*, Wind River Systems, Inc. 2007.
[18] *VxWorks 653 Configuration and Build Reference, 2.2,* Wind River Systems, Inc. 2007.
[19] *VxWorks 653 Programmer's Guide 2.2*, Wind River Systems, Inc. 2007.
[20] http://www.scarlettproject.eu

Chapter 10

Improving Dependability of Embedded Software Systems using Fault Bypass Modeling (FBM)

1. Introduction

Embedded software plays today a very significant role in our daily lives. Everything from our mobile devices, telecommunications infrastructure, satellites to home appliances and automotive products depend heavily on the embedded software to provide functionality and services. Over the last two decades, there has been an enormous increase in the complexity of embedded software, shorter innovation cycle times, while the demands for their reliability and dependability have anything but grown [1].

Due to requirements of real time behavior and stringent demands for quality and dependability, embedded software are much more complex than their counterparts in IT applications or desktop software due to real-time and interface constraints [2]. Also given that the late defect correction costs are higher in embedded software development and testing of software after code completion costs about 30-50% of all resources [2], verification and validation holds special significance in this domain. Model driven or model based development (MBD) is now widely adopted within the domains of embedded software/systems development. A good overview on embedded software development and model based development therein can be found in [3], [4], [5], while [2] provides important facts, figures and the expected future for embedded software.

The problem and challenges related to verification and validation of models, specifically how to verify, validate, and test the behavioral/implementation models that are used for code generation is also well recognized within the research community of model driven engineering [6]. The predominant form of testing within embedded software using MBD is done using test cases and test scenarios. Model based testing (MBT) attempts to use data models to generate tests where data models intend to capture the requirements and input configurations [7]. While test automation and MBT provide considerable reduction in cost of test generation, the importance of real-time issue and the need for testing using continuous signals calls for reactive or closed loop testing.

Closed loop testing offers many advantages for testing systems which depend or interact closely with their environment. By modeling the environment and letting the system under test interact with its environment through controlled interfaces - provides the possibility of reactive testing, identification/generation of multiple system-environment test cases/scenarios automatically, and tests the system for its real time characteristics.

Fault injection techniques can further enhance the effectiveness of closed loop testing and thus help in evaluating and increasing the dependability of a system in its early stage of development, but injecting faults into a system (in a closed loop configuration) may lead to unrealistic system behavior, which is hence unreliable for making analysis or testing hypothesis. The problem occurs mainly due to dependencies between the system and its environment and feedback loops between them. In this paper we highlight the problem and discuss how a framework referred as fault bypass modeling can be used as a potential solution to this problem.

2. Related work

The idea of fault bypass modeling presented here is introduced by the authors of this paper in [8], with a case study using a behavioral model of an anti-lock braking system in Simulink. In this paper we highlight the need of closed loop testing and evaluate the applicability of the fault bypass principle to a case of simulating an autonomous vehicle.

Using fault injection techniques for dependability evaluation of behavioral/functional models are on the rise, Svenningsson et al. [9] introduce the tool called MODIFI which can be used to apply fault injection methodology on functional/behavioral models in Simulink. The tool is capable of injecting single or multiple faults into the signals of a given system to evaluate the fault propagation properties and analyze the effectiveness of fault tolerance mechanism of the system under test. But as described in our earlier work [8], using such tools in closed loop mode needs careful consideration to fault bypass principle to ensure that the system output is realistic and reliable. Such a modelling is important even for other domains, e.g. a measurement system or software modelling in general [10, 11].

Trawczynski et al. [12] presented an approach for modelling software systems in cars using closed loops in the context of security engineering. Their approach provides another example from a similar domain.

2.1. **Need for closed loop testing**

The main form of traditional software testing is open loop testing using test case and scenarios. But in a number of industrial domains and types of applications where the program/system under test is non-deterministic or where the behavior of given function/system depends on its operational environment – open loop testing is not an effective approach. In such cases the problem of test case generation even using MBT or automated test generation tools is much more complex than for deterministic type of applications [13].

Stockmann et al. [14] document the need for closed loop testing in the automotive industry. Focusing on the domain of electric vehicles and testing electronic control units (ECUs), the authors propose a methodology and tool chain for simulating virtual ECUs to enable functional testing under different conditions. The requirement of using closed loop testing for testing model based development in automotive domain due to real time issues behavior and need for using continuously changing signals is also expressed in [15]. The need for testing in the virtual space and thus, in a closed loop configuration due to advancements in autonomous driving, vehicle to infrastructure, and vehicle to vehicle communication is further established in [16]. Further need for such testing in automotive domain is highlighted in study by Matinnejad et al. [17] that explored the Model-In-Loop testing of highly configurable continuous controllers.

The problem of non-deterministic factors of testing and the need for closed loop testing in the area of medical devices is explained and highlighted in [13]. The authors refer to implantable devices increasing complexity as a factor leading to large amount of device recalls. The safety critical nature of such systems calls for more rigorous testing, one way of increasing test coverage is by using physically relevant model for test case generation for such devices using a closed loop testing environment. For example in pacemakers the capability and effectiveness of the approach is demonstrated by the system's ability to test for common and complex heart conditions for different pacemaker models.

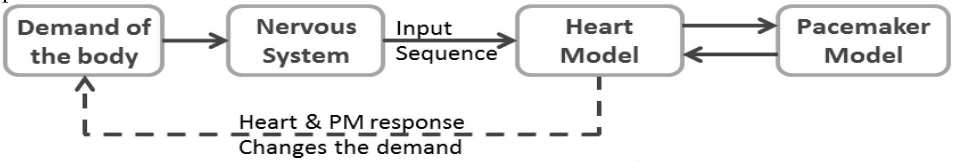


Figure 1. Simulated heart and pacemaker model in closed loop configuration as presented in [13].

A common approach to test systems with human elements in closed loop is to couple the human subject to the simulated system. Using virtual human models as a cheaper alternative is an area of active research [18]. Thus, closed loop testing is also important in areas with a man-machine interface, which constitutes a large part of day-to-day products with embedded software.

2.2. **Fault Injection**

Fault injection has been used with good results for verification of dependability attributes of hardware and software systems [19]. Fault injection is widely used to identify bottlenecks related to dependability, study the behavior of a system under faulty operating conditions, and examining the coverage of fault tolerance or error detection and recovery mechanisms within software systems.

For applications, which are safety, mission, or business critical, dependability evaluation is especially important activity. Fault injection techniques have been studied much and used for safety critical applications development. SCADE or Safety-Critical Application Development Environment is a modeling language developed to simulate hardware failure scenarios. SCADE have been used in projects such as ESACS and ISAAC for identification of fault combinations leading to safety case violations. A plug-in called FISCADE [20] have also been developed for SCADE language for introducing faults into using the SCADE simulator.

3. Improving closed loop testing using fault injection

As described in 2.1, there is a high need for using closed loop testing for a number of domains and applications. Closed loop testing can be achieved by developing/modeling the environment, with which the system interacts, and simulating the system and environment coupled through interfaces in a virtual environment. Using MBD and MBT approaches in conjunction can be used to generate tests for system in closed loop configuration which works well under normal (specified) working conditions.

But in order to achieve dependability evaluation of a system; for example running fault based scenarios, we need to go one step further to the closed loop testing. This could be easily done by injecting faults into the system. Using such approach many scenarios can be created, for instance a system with inputs from n sensors may run scenarios with individual failure of x $(0 < x < n)$ sensors input and their combinations. Different types of sensor/input failure modes could be modeled and so does the failure related to reading parameters and system dependencies onto other system which simulates reading, writing or memory errors. All these fault operating conditions can be used to identify failure modes under which system output is unacceptable and test cases/scenarios generated to ensure that final implementation code have error handling or tolerance capabilities to avoid such scenarios.

Thus by coupling fault injection techniques with close loop testing, the efficiency and effectiveness of testing real-time systems with non-deterministic or environmental dependent properties can be enhanced significantly. Model based development and closed loop configuration allows for running the system automatically against a large number of normal and fault scenarios, which would not be possible in an open loop configuration or by using manually crafted test cases.

But the main challenge in using fault injection in a close loop configuration is to differentiate between correct system's behaviors from the system failure under fault mode. The problem is described in the next section using a simple case study, while detailed description using a behavioral Simulink model is also available in [21].

4. Case study: problem description and proposed solution

In this section we describe the challenge when using fault injection in a closed loop configuration. We use the miniature vehicle and its environment model as described in [16].

The implementation of system-environment model for the autonomous miniature car and its environment is presented in Figure 2. The modules named monitor, lanedetector and driver are the parts constituting the system within the car, while the vehicle, camgen, and irus forms the simulation for the environment model. The simulation can take inputs from a scenario modeling GUI, which gives flexibility of designing and running test scenarios.

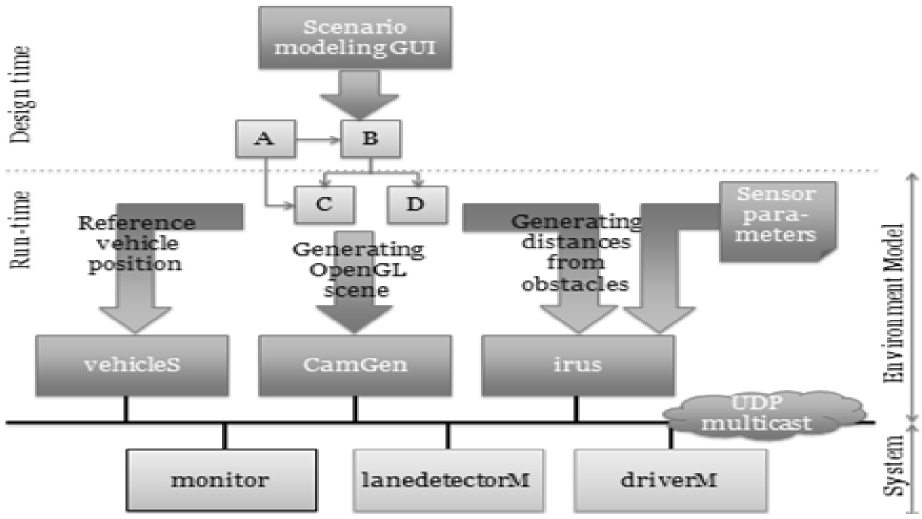


Figure 2. Representation of model-based system-environment model capable of simulating vehicle-environment model in virtual space, as presented in [16].

When simulating the autonomous miniature car in the virtual environment, the lanedetectorM module takes input from environment simulator module CamGen, which is producing virtual image data similar to a camera input during on-road conditions. Using data from CamGen and controlling commands provided by the user in the virtual environment or using the test scenario model, the driverM module determines the current vehicle position. The driverM module also calculates the demand velocity (V_d) and the desired steering wheel angle (θ_d) to be applied using input from lanedetectorM and driving instructions.

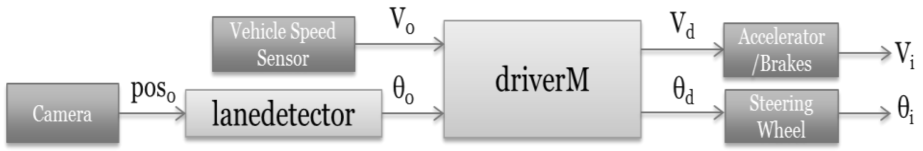


Figure 3. Miniature vehicle running in open-loop condition.

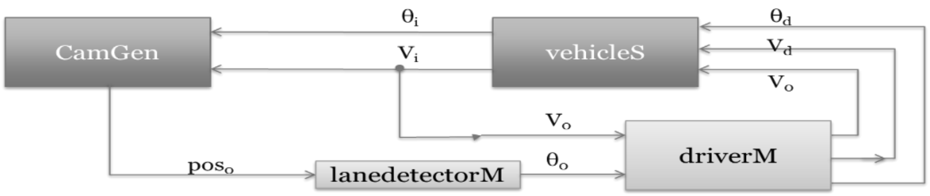


Figure 4. Vehicle in the virtual simulation mode under closed loop operation.

The output of the driver module is used to control the vehicle movement in case of on-track mode, or it is fed back to the vehicleS module in the simulation mode to calculate the new vehicle position using linear bicycle model. The new position from the vehicleS module is then used by CamGen to generate new image data and irus to re-calculate the obstacles distance to be used by lanedetectorM and driverM modules. Figure 3 and Figure 4 represent the working modes in on-track and virtual simulation mode.

4.1. **Injecting fault into the system**

Now we consider a simple scenario, where we simulate how the vehicle would act in the case of a faulty speed sensor (sensor output is zero). In the real vehicle on the track, even though the speed sensor has failed at t=t0, we can reasonably assume that vehicle would continue in motion with its initial velocity v0 and process the observed camera images to navigate the lane according to lanedetectorM input. Although due to an assumed failure in the vehicle speed sensor, the vehicle speed would be assumed by driverM (system) as zero and thus demand maximum speed resulting in full throttle leading to vehicle accelerating and continuing operation in full speed mode.

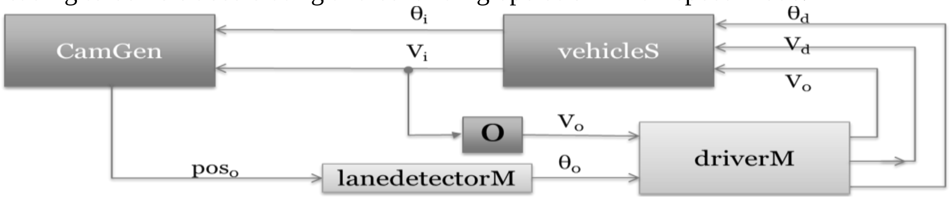


Figure 5. Vehicle in the virtual test environment mode with fault injected.

If we simulate the same condition in a virtual test environment, the fault condition of vehicle speed 0 would be interpreted in the manner as described above (like in real case) by the vehicleS model to simulate a condition with full acceleration demand. The wrong signal (zero vehicle speed) will make incorrect new vehicle speed and thus also the distance traveled from the point of fault injection leading to faulty position interpretation by CamGen and thus the vehicle speed and trajectory in simulated case will not reflect the actual behavior and thus unreliable to make analysis.

Using a simplified 1D model, the current velocity and distance can be calculated using Newton's law of motion,

$$v = v_0 + at$$
$$S = S_0 + v_0 t + \frac{1}{2}at^2$$

In case of the actual vehicle on the track, due to the faulty vehicle velocity input (v = 0 m/s) the driverM module will demand maximum acceleration (assumed here $a_{max} = 5\ m/s^2$), but the initial velocity irrespective of the state (working or faulty) will be v_0 (assumed to be 40m/s below) will follow the laws of motion. Also the observations from camera unit will be normal and thus the vehicle would be able to navigate the obstacles and follow the lane.

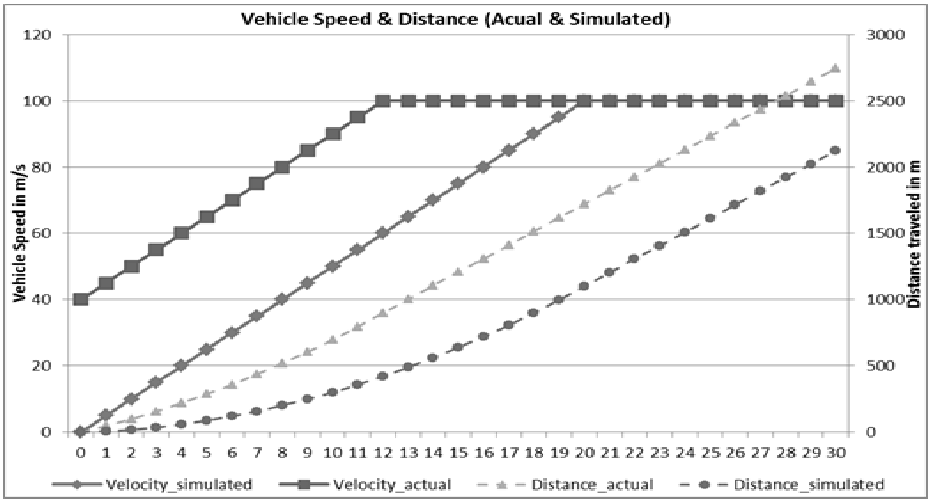


Fig 6: Velocity and distance traveled (actual and simulated).

While in case of the virtual test environment, the initial velocity will be wrongly taken to be 0 and although the driverM module will demand similar condition of maximum acceleration, in this case the simulated velocity and distance traveled would be wrongly calculated. And since in the simulated case the module CamGen is used instead of real camera, the generated image based on wrong position data from the vehicle module will result in faulty image generation, and hence the vehicle would not navigate or follow lane correctly. Fig 6 shows the difference between velocity and distance in actual and simulated case.

Such inconsistencies occur *due to dependencies and superficial feedback loops between the system and its environment where a system state/signal is used to calculate/control a natural parameter which in normal circumstances would not depend on that signal/ state of the system* [21]. In the given case study, the problem occurs due to the virtual vehicle dynamics simulator (vehicleS) that will take wrong input of current velocity (as zero) in the fault scenario, which is used to calculate the new velocity and new vehicle position, which is further used to generate the virtual image data by CamGen and thus producing incorrect simulated outcome.

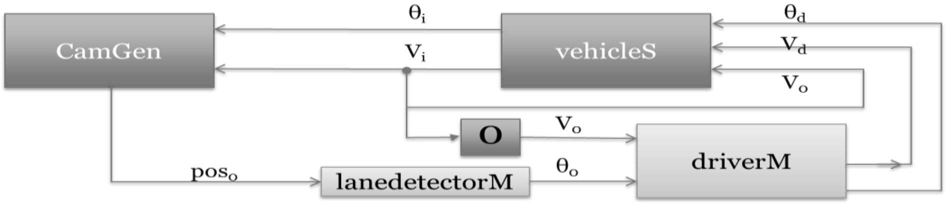


Figure 7. Vehicle simulation closed loop testing using FBM.

The solution for such problems is easily achieved by using the principle of fault bypass modeling where the part of the signal or its derivative, which is used to calculate/control an environment parameter (in this case correct initial velocity) is made fault free to break the unrealistic feedback loop. In the above mentioned case when FBM principle is applied, the initial velocity of moving vehicle is a parameter independent of the injected fault. Thus when we simulate the given fault scenario, fault (v=0) needs to be provided to the driver module, but the fault free current value of initial velocity should be passed by to the simulated environment (vehicleS module) so that the new velocity and position data is correctly generated and thus the output of CamGen (generating the virtual image data) as well. The implementation of FBM in the given case is represented in Figure 7.

This is a simple example but for many embedded systems that require closed loop testing, transient properties are important or even critical. Consider testing if the vehicle stops safely under a scenario of failed brakes or how the pacemaker or some implantable device would react to an intermittent discharge from the battery. Using the fault injection methodology to test for these fault scenarios under closed loop strictly depends on ensuring that the system-environment simulated output is reliable and reflects the realistic behavior of system under test. Thus, the FBM principle outlined here can be useful for closed loop testing of dependability of non-deterministic systems and systems with high dependence on their environment.

5. Conclusion

We established that there is significant need for using closed loop testing of embedded software systems in many domains and applications. It is also discussed that fault injection can be used to enhance the effectiveness of closed loop testing by making it possible to do dependability evaluation of the system in early development stages. But injecting faults into closed loop configurations can generate outputs that are unreliable and unrealistic. To overcome this problem, a framework referred to as fault bypass modeling is demonstrated with a simple case study. Although the example discussed here is very simple, the use of closed loop testing is most often needed for testing of safety critical applications where dependability and reliability is of utmost importance; thus, FBM can prove to be a useful tool in ensuring dependability of embedded systems.

Acknowledgments

The work presented here has been funded by Vinnova and Volvo Cars jointly under the FFI programme (VISEE, Project No: DIARIENR: 2011-04438).

References

[1] P. Liggesmeyer and M. Trapp, "Trends in embedded software engineering," *Software, IEEE,* vol. 26, pp. 19-25, 2009.

[2] C. Ebert and C. Jones, "Embedded software: Facts, figures, and future," *Computer,* pp. 42-52, 2009.

[3] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty, "Model-integrated development of embedded software," *Proceedings of the IEEE,* vol. 91, pp. 145-164, 2003.

[4] B. Graaf, M. Lormans, and H. Toetenel, "Embedded software engineering: the state of the practice," *Software, IEEE*, vol. 20, pp. 61-69, 2003.

[5] G. Buttazzo, "Research trends in real-time computing for embedded systems," *ACM SIGBED Review*, vol. 3, pp. 1-10, 2006.

[6] R. Van Der Straeten, T. Mens, and S. Van Baelen, "Challenges in model-driven software engineering," in *Models in Software Engineering*, ed: Springer, 2009, pp. 35-47.

[7] S. R. Dalal, A. Jain, N. Karunanithi, J. Leaton, C. M. Lott, G. C. Patton, *et al.*, "Model-based testing in practice," in *Proceedings of the 21st international conference on Software engineering*, 1999, pp. 285-294.

[8] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, "Improving Fault Injection in Automotive Model Based Development using Fault Bypass Modeling," in *GI-Jahrestagung*, 2013, pp. 2577-2591.

[9] R. Svenningsson, J. Vinter, H. Eriksson, and M. Törngren, *MODIFI: a MODel-implemented fault injection tool*: Springer, 2010.

[10] L. Kuzniarz and M. Staron, "On Practical Usage of Stereotypes in UML-Based Software Development," in *Forum on Design and Specification Languages*, Marseille, 2002, pp. 262-270.

[11] M. Staron and W. Meding, "Using Models to Develop Measurement Systems: A Method and Its Industrial Use," presented at the Software Process and Product Measurement, Amsterdam, NL, 2009.

[12] D. Trawczynski, J. Zalewski, and J. Sosnowski, "Design of Reactive Security Mechanisms in Time-Triggered Embedded Systems," *SAE International Journal of Passenger Cars-Electronic and Electrical Systems*, vol. 7, pp. 527-535, 2014.

[13] Z. Jiang, M. Pajic, and R. Mangharam, "Model-based closed-loop testing of implantable pacemakers," in *Proceedings of the 2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*, 2011, pp. 131-140.

[14] L. Stockmann, D. Holler, and D. Spenneberg, "Early simulation and testing of virtual ECUs for electric vehicles," in *International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium (EVS26)*, 2012.

[15] E. Bringmann and A. Kramer, "Model-based testing of automotive systems," in *Software Testing, Verification, and Validation, 2008 1st International Conference on*, 2008, pp. 485-493.

[16] C. Berger, M. Chaudron, R. Heldal, O. Landsiedel, and E. M. Schiller, "Model-based, composable simulation for the development of autonomous miniature vehicles," in *Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS Integrative M&S Symposium*, 2013, p. 17.

[17] Matinnejad, Reza, et al. "MiL testing of highly configurable continuous controllers: scalable search using surrogate models." Proceedings of the 29th ACM/IEEE international conference on Automated software engineering. ACM, 2014.

[18] W. F. Van Der Vegte and I. Horváth, "Achieving closed-loop control simulation of human-artefact interaction: a comparative review," *Modelling and Simulation in Engineering*, vol. 2011, p. 24, 2011.

[19] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, *et al.*, "Fault injection for dependability validation: A methodology and some applications," *Software Engineering, IEEE Transactions on*, vol. 16, pp. 166-182, 1990.

[20] J. Vinter, L. Bromander, P. Raistrick, and H. Edler, "Fiscade-a fault injection tool for scade models," in *Automotive Electronics, 2007 3rd Institution of Engineering and Technology Conference on*, 2007, pp. 1-9.

[21] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, and F. Törner, "Increasing efficiency of iso 26262 verification and validation by combining fault injection and mutation testing with model based development," in *8th International Joint Conference on Software Technologies-ICSOFT-EA, Reykjavík, Iceland, July 2013*, 2013, pp. 251-257.

Chapter 11

From Academic Project to Production Software Based on Java Web-tier CMS Application

1. Introduction

The Content Management System (CMS) described in this article is a content management system based on Java and HTML technology, and is tailor designed for requirements of Lodz University of Technology (TUL) dealing with heavy load as a production software, content creation, editorial workflow and publishing for TUL. First version of the system was designed in 2005-2006 at doctoral studies and later on the author had to treat it as legacy Struts application and continue improvement to obtain measurable performance effects not using state-of-the-art Java adds on.

Web CMS system consists of frontend and backend. The front is what we see and the backend is hidden within its architecture and logic to obtain functional and performance purposes. This article shows a little behind the curtain of authors original CMS project which emerged within 2005-2013. This CMS is multi-tierd application based on web tier of Java Enterprise Edition (JEE) platform with Model View Controller (MVC) framework, Java Server Pages (JSP), Java Standard Tag Library (JSTL), Expression-Language (EL), Struts 1.2, Object Relational Mapping (ORM) Hibernate and MySql.

As far as the design rationale is concerned author has choosen this solution because in 2005 it was hardly to find open source mature CMS solution which would fulfil requirement of all Steakholdes. Author concentrates on functional solutions of multi hierarchy, multi-domain operability in app, etc. and examples of performance gain practices applied in web-tier CMS application of TUL changing the academic project into production web aplication.

Some technical solutions cases are shown as examples to explain ways on how the web app was improved from academic project to production software, scaled based on own experiences on the research [1][2][3][4][5][6][7][8][9][10][11][12][13] and engineering projects, and making at the same time practice, science and algorithms.

This web system was custom designed for demands of administation of Lodz University of Technology and was refactored to service the emerging increasing quantity of incoming http traffic year by year becoming production software. In year 2006 the CMS introduced decentralization of responsibility for the information which was put to the web by administrative departments of Lodz University of Technology. Each administrative department started operating its web content. There were multi hierarchy, multi-domain operability, multi lang versions features of the system implemented in one application context.

In 2010 quantity of visitors increased significantly and improvement of performance was demanded. In on – peak traffic periods the http sessions were from 1k to 10k per day. Rewriting code for decend performance and ability for scalability took consistently till 2013. The refresh of front end was done simultaneously (front end was delivered by other vendor) and in may 2013 the new production and scalability ready web app was deployed at www.p.lodz.pl. The system is ready for operation of 1 to 2k http session at the same time with one server. Nowadays the monthly traffic for p.lodz.pl domain in peak season is 300k http sessions per month what gives circa 1M clics on the web per month.

2. Functional solutions

Functional solutions like multi hierarchy, multi-domain operability in app was implemented because of the requirements of organization. For the first stage the development of CMS project, the aspect of functionality was first priority. Thus the author concentrated on functionality required to operate the administration of Lodz Univeristy of Technology. The so important aspect of performance was an add-on code refactor later on in time.

2.1. To be smarter than infrastructure - multi-domain operability in app

Web-programming model for enterprises called the Java 2 Enterprise Edition (J2EE) extened with architectural framework allowed to build multitierd, here 3-tiered e-business applications.

Author used one of the MVC frameworks called Struts to operate HTTP request [18]. HTTP requests from thin client are view events, Fig. 1, a logic operates the HTTP request and responses through ActionClass and a controller , here ActionServlet directs control to proper views (JSP pages). Detailed analisys of J2EE architecture and code generation from model are described in authors doctoral dissertation [18].

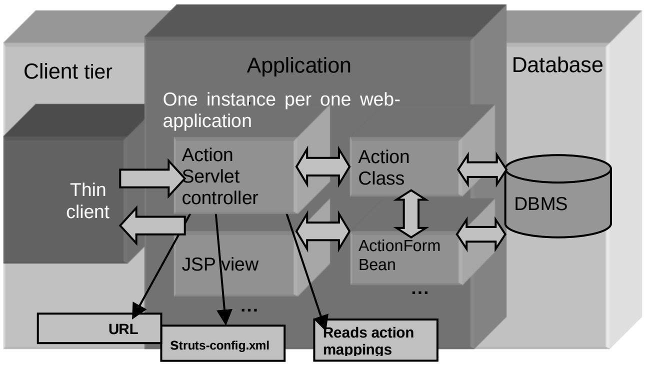


Figure 1. Struts MVC flow of control schema.

Struts 1.x which is used in mentioned CMS do not support multi-domain operability that might operate many domains in one web context. Usually one instance per one web-application.

The author implemented own logic for multi-domain operability in one web context by adding additional flag “main_context” based on url decomposition in ActionServlet container of Struts framework. The code in Listing 1 shows the idea.

Listing 1. Code example of multi-domain operation in one web context block in controller

```
if(request.getAttribute("main_context")==null ){
    //--------------- next domain
    String domena=request.getServerName();
          ...
          if( (domena.endsWith("www.studyinlodz.edu.pl") ||
                    domena.endsWith("studyinlodz.edu.pl")) ){
                      ...//data context
                      Menu m=impl.getDefaultLeafForDomainName(domena);
                      if(m!=null){
                                 int numer=m.getId_kat();
                                 url="/studyinlodz,menu"+numer+",_index.htm";
                      }else{
                      try{
                      re-
                      quest.getRequestDispatcher("/"+"domena_not_operable.htm").forward(req
                      uest,response);
                      return;
                      }catch(Exception ex){ log.debug(ex.toString());}}
          ...
          try {      super.process(request, response);} catch ...
}}}
```

If server name contains domain checked then the data are being fetched to show in the context of this domain. This might be implemented as Struts extension since most Web-tier application frameworks lack this design pattern.

2.2. Multi hierarchy and multi language support

This solution is very flexible and usefull for multi hierarchy support for menu items. The data is encoded with UTF-8 standard thus allowing for multilanguage content for all domain context to be presented in one web context. Thus presenting polish, english version, and Chinese, and russian, and Ukrainian.

The relational schema implies risk on how we collect the hierarchical data. One mistake in the algorithm and it may casue the jam problem. Then only helps the memory dump of the thread with “kill -3 javapid”, detailed analysis of the dump, debug and fixation of code.

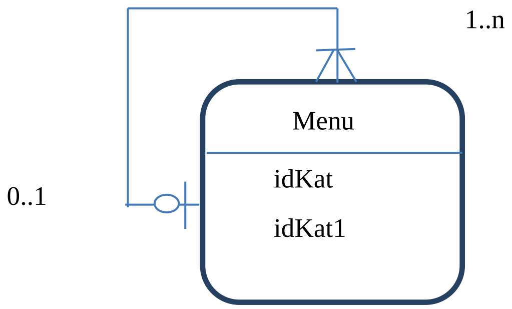


Figure 2. ERD schema of self referencing menu table

2.3. Decentralization of operation of CMS by administrative staff of TUL

The Access Control Lists (ACL) for the CMS allowed for decentralization of operation of CMS. Using a combination of ACLs, permissions, and roles, CMS provides methods for setting and restricting the access available to CMS users.

That means that each organizational unit is able to operate its content by themselves. This feature was deployed in 2007 at the Lodz University of Technology. The remaining functionality is as follows:

- Browser-based interface,
- WYSIWYG editing tool,
- Role-based workflow,
- Permissions model.

2.4. Friendly URLs

The system has been based on url generation with keywords coming from title of the article put to the CMS by the editor. The URL schema was modelled on the basis of the “Google secrets…” [20]. The link structure is plain and wide.

Thanks to the Search Eengine Optimization (SEO) up till now the web page is easily found on top 10 position in Google. For example it keeps top Search Engine Results Page (SERP) position for keyword “Politechniki”. There are circa 30k content urls.

3. Performance solutions

When quantity of visitors of web in 2010 increases the improvement of performance was demanded. Many software designers and developers take the functionality as the most important issue in a product while thinking of performance and scalability as add-on features. Most of them believe that expensive hardware can fix the performance issue. The same was with the authors CMS thus the web application must have evolved from academic project and become production application.

To scale vertically (or scale up) means to add additional CPUs or memory to a single computer [14]. To scale horizontally (or scale out) means to add more nodes to a system, such as adding a new computer to a distributed software application. The author concentrates on architectural approach which touches mostly the aspect of performance and at last the vertical scalablity.

3.1. Avoid the database - make cache

In order to improve performance of web-application we have to take into account many aspects of web-application i.e. server side consists of many aspects in the topic of performance. By analyzing the results obtained during this phase it is possible find bottlenecks, memory leaks or performance problems related to database layer.

In this case the re-architecture and re-code the whole solution is demanded but it cost money and time since obtaining performance is a time consuming work and error prone.

The architecture of the system assumed in the academic project in 2006 (static data) that all files are put to the database (because of the ease of db migration) and when http requests comes they are taken again and again out of the database through all 3-tir layers of the web app. This caused big bottleneck when the http traffic increased. The re-architecture and re-coding the whole solution was done in a way that the uploaded files were not only saved to database but also into directory of web server (as static data) as well. This solution concerning static data improved significantly performance.

In the web apps where every request processing action needs much data to process the memory-caching comes into play. The method applied by the author in the CMS is based on caching objects in AppContext and readdressing them to HttpRequest for every request in the session for the presentation layer of guest user-agent - code list 3 ilustrates that. Avoiding the database to reduce the database reads is sometimes not possible because of the dynamics of the system and e.g. some-point-critical e.g. financial data but in the author's CMS caching and reloading the cache is appropriate solution for servicing data.

In the CMS the http data is cached where there are more frequent reads operation than updates. The cached object is *PrePrezenter.*

Of course there are algorithms for invalidating and remaking the cache whenever the update from the backend system is done. This solution improved the performance significantly. Caching objects in memory when the system is initialized to avoid creating and fetching from persistent tier too many objects when running improves performance.

Listing 2. In app cache making schema

```
PrePrezenter pp1=null;
…
pp1=(PrePrezenter)context.getAttribute("preprezenter_spec_pl");
if(pp1!=null&&pp1.getList().size()>0&&breloadnewsstronaglowna==false){
        Helper.setToRequest(request,pp1,"preprezenter_spec");
}else{
        pp1=Helper.getDocumentsMainPage("stronaglownamain",…params );
        Helper.setToRequest(request,pp1,"preprezenter_spec");
        context.setAttribute("preprezenter_spec_pl", pp1);
        context.removeAttribute("reloadnewsstronaglowna")
//--
}
```

3.2. Switch from Hibernate to JDBC on front end requests

At the beginning of working on CMS the front end as well as back office were designed with a Hibernate - an object-relational mapping framework for the Java language. This framework mapped from Java classes to database tables (and from Java data types to SQL data types) excellent but there was a little performance overhead. Author decided to rewrite the code for front-end http actions to jdbc instead of hibernate leaving the previous framework for a backend.

JDBC (Java DataBase Connectivity) access a database in much quicker time. Of course there is source code overhead instead when writing JDBC logics.

3.2.1. Not leak resources

Close any JDBC instances that weren't explicitly closed during normal code path, not 'leaking' resources. The code listing 3 shows the details of explicit releasing resurces in whatever path of execution of the code.

Listing 3. JDBC closing connection

```
public List getRodzajeMenu(String lang) throws DAOSysException {
        Connection c = null;
        Statement stmt=null;
        ResultSet rs = null,rs2 = null;
        List ret = new ArrayList();
        javax.sql.DataSource ds;
        String f="select identyfikator_menu from dmenu m   where
        m.id_jezyka='"+lang+"'
        group by m.identyfikator_menu order by m.kolejnosc";
   try {
        ds = DBAFactory.getDs();
        c = ds.getConnection();
        stmt = c.createStatement();
        rs=stmt.executeQuery(f);//
        while(rs.next()){
        ret.add(rs.getString("identyfikator_menu"));
        }
        if(rs!=null)rs.close();
        if(rs2!=null)rs2.close();
        stmt.close();
        c.close();
 } catch (SQLException se) {
log.error("Error List  "+se.toString());
throw new DAOSysException("SQLException: " + se.getMessage());
}finally {        if (stmt != null) { try {
                            stmt.close();
                          } catch (SQLException sqlex) {}
                          stmt = null;}
                        if (c != null) {
                          try {
                            c.close();
                          } catch (SQLException sqlex) {}
                          c = null;
                        }}
return ret;
}
```

3.3. Coordination beetwen threads – "synchronized" keyword

"*The primary tool for managing coordination between threads in Java programs is the synchronized keyword. The synchronized keyword will force the scheduler to serialize operations on the synchronized block. If many threads compete for the contended synchronizations, and only one thread is executing a synchronized block, then any other threads waiting to enter that block are stalled. If no other threads are available for execution, then processors may sit idle. In such situations, more CPUs can help little on performance. The JVM has to maintain a queue of threads waiting for that block (and this queue must be synchronized across processors), which means more time spent in the JVM or OS code and less time spent in your program code*" [14][15]. To avoid the hot lock problem, the author made synchronized blocks as short as possible – code listing 4 - moving the thread safe code outside of the synchronized block.

Listing 4. Synchronized block

```
package config;
…
public class SessionCounter implements HttpSessionListener {
…
if(se.getSession().isNew()==true){
        synchronized(this){
                activeSessions++;
        }
        …
```

Paying attention to lock granularity is recommended. When we put the "synchronized" keyword on a method, we are locking on "this" object implicitly making lesser granularity. The entire object is locked when calling its method thus we decrease performance and ability to scale. The same is the lock on static methods which means lock on all instances of this class [15]. Programmer may choose the attitude from vast choice of wait-free methods like compare and swap CAS or from java.util.concurrent.atomic package.

3.4. Non-Blocking IO in Tomcat 6

Upgrade of server Tomcat 5 to Tomcat 6 which has embraced non-blocking IO was key factor of better performance of the web application of Lodz University of Technology.

In non-blocking IO, a working thread will not binding to a dedicated request [15]. If one request is blocking due to any reasons, this thread will reuse by other requests, In such way, Glassfish can handle thousands of concurrent users by only tens of working threads.

3.5. Adding more Memory to the Server

Memory is an important resource for your applications. Enough memory is critical to performance especially for database systems. More memory means larger shared memory space and larger data buffers, to enable applications read more data from the memory instead of disks.

"*Too little memory will cause garbage collection to happened too frequently. Enough memory will keep the JVM processing your business logic most of time, instead of collecting garbage. Java garbage collection relieves programmers from the burden of freeing allocated memory, in doing so making programmers more productive. The disadvantage of a garbage-collected heap is that it will halt almost all working threads when garbage is collecting. In addition, programmers in a garbage-collected environment have less control over the scheduling of CPU time devoted to freeing objects that are no longer needed.If one adds Java applications are NOT scalable by given too much memory. In most cases, 3GB memory assigned to Java heap (through "-Xmx" option) is enough.* " *Cited* [14].

This scenario gives the conlucion that Java applications must be well prepared for the scalability, from the system design phase to the implementation phase of the products' life cycle. The scalability is really based on ones programmer vision.

4. Conclusion

In a Web environment concurrent use is measured as simply the number of users making requests at the same time. When the application has decent response time then this aspect is called good performance. Performance refers to the capability of a system to provide a certain response time. It is also software quality metric.

It became crucial for the author of CMS when number of visitors of web page of Lodz University of Technology inceased in 2010.

As we see in this paper the system become production application from academic, focusing in the later stage on the performance increasing teachnique rather then functional. The author realies that the systems are NOT scalabable out-of-the-box and in nearly all cases this is architectural problems.

The system reached decend response time ~1s for 0-2000 http request at the same time. The statistics shows nearly 300.000 http sessions per month in a peak period.

Author suggest premature optimization shoud be done with performance optimization during designing and implantation phase.

Lifecycle APM (Lifecycle Performance Management) and Continuous Performance Management [19], suggest to get all information to know about the scalabilty and performance characteristcs of your application any time. This serves as a basis for deciding when and where to optimize.

"*Concluding we can say that if we want our systems to be scalable we have to take this into consideration right from the beginning of development and also monitor throuhout the lifecycle. If we have to ensure it, we have to monitor it. This means that performance management must then treated equally relevant than the management of functional requirements*". [16]

References

[1] Wojciechowski J., Napieralski A., „Zastosowanie Platformy J2EE w Projekcie Serwisu Internetowego DWZ P.Ł" XI Konferencja „Sieci i Systemy Informatyczne", Łódź, październik 2003, pp. 131-135, ISBN 83-88742-91-4.

[2] Wojciechowski J., Napieralski A. „System wspomagający wykładowcę i studenta przez WWW", Mikroelektronika i informatyka, maj 2004, KTMiI P.Ł. pp. 235-238, ISBN 83-919289-5-0.

[3] Wojciechowski J., Sakowicz B., Dura K., Napieralski A., "MVC model struts framework and file upload issues in web applications based on J2EE platform", TCSET'2004, 24-28 Feb. 2004, Lviv, Ukraine, pp., 342-345 , ISBN 966-553-380-0.

[4] Szymański G., Wojciechowski J.A., Ciota Z. "Design of Web-Based Tutor-Supporting System on The Basis of JAVA Platform" 11th International Conference MIXDES 2004, Szczecin , Poland 24-26 June, pp. 607-610, ISBN 83-919289-7-7.

[5] Wojciechowski J.A., Owczarek M., Napieralski A. "Java Web Services Aplication in University Web System" 11th International Conference MIXDES 2004, Szczecin , Poland 24-26 June, pp. 611-614, ISBN 83-919289-7-7.

[6] Wojciechowski J., Kozłowski M., Napieralski A. "Security Aspects of Web Applications Implemented within J2EE Platform" 11th International Conference MIXDES 2004, Szczecin , Poland 24-26 June, pp. 619-622, ISBN 83-919289-7-7.

[7] Wojciechowski J.,Napieralski A. „System jednolitej autoryzacji w środowisku heterogenicznym opartym o www z zastosowaniem architektury klucza publicznego PKI oraz bazy danych LDAP" International Workshop for Candidates for a Doctor's Degree, 16-19 October, Wisła, pp. 461-464, ISBN 83-915991-8-3.

[8] Wojciechowski J., Murlewski J., Napieralski A. "Pozycjonowanie stron internetowych w serwisach wyszukiwawczych na przykładzie Google", KmiTI Mikorzyn 23-25.09.2005, Mikroelektronika i Informatyka, Prace Naukowe, Łódź 2005, str. 89-94, ISBN 83-922632-0-0

[9] Wojciechowski J., Murlewski J., Sakowicz B., Napieralski A., "Object-relational mapping application in web-based tutor-supporting system", CADSM, Lviv-Polyana, Ukraine, Feb. 23-26, 2005, pp. 307-310, ISBN 966-553-431-9.

[10] Owczarek D., Wojciechowski J., Murlewski J., Sakowicz B.,Napieralski A: „Electronic Document Management System",13th International Conference Mixed Design of Integrated Circuits and Systems MIXDES 2006, 22-24 czerwca 2006, Gdynia,wyd. KMiTI, str. 791-792, ISBN 83-922632-9-1.

[11] Wojciechowski J. „New methodology in designing reactive systems with formal methods based on authorization for hierarchical, component based system with time dependencies", International PhD Workshop for Candidates for a Doctor's Degree OWD 2006, 21-24 X 2006, Wisła, Polska.

[12] Wojciechowski J. „Mapping of Petri net formal model of concurrent system to class model with aspect of polymorphism in object oriented paradigm", Zeszyty Naukowe Katedry Mikroelektroniki i Technik Informatycznych : Mikroelektronika i Informatyka, zeszyt nr 7, Łódź 2007, ss.167-170, ISBN 83-9222632-5-1.

[13] Zięba B., Wojciechowski J., Jabłoński G., Zabierowski W., Napieralski A., :Web-Based Distributed Physic-Based Simulation System of Semiconductor Diode Structure" 10th International Conference Mixed Design of Integrated Circuits and Systems MIXDES 2003, 26-28 June 2003, Łódź, Poland , pp. 690-693, ISBN 83-7283-095-9.

[14] Scaling Your Java EE Applications, By Wang Yu, 01 Jul 2008, TheServerSide.com, http://www.theserverside. com/ news/1363681/Scaling-Your-Java-EE-Applications.

[15] The Top 10 Ways to Botch Enterprise Java Application Scalability and Reliability, Cameron Purdy on Jul 23, 2008, http://www.infoq.com/presentations/10-ways-botch-scalability-reliability.

[16] Performance vs. Scalability September 11, 2008, Alois Reitbauer http://apmblog.dynatrace.com/2008/09/11/performance-vs-scalability/

[17] Wojciechowski J., "From formal methods to implementation based on Petri Nets model of concurrent systems", Pomiary, Automatyka, Kontrola, Vol. 53, No. 5/2007, Maj 2007, pp.132-134, ISSN 0032-4140.

[18] Wojciechowski J. "Translation method of Coloured Petri Nets models towards Java Web application schema based on multi-tier distributed authorization system" Praca doktorska, 2009, Biblioteka Politechniki Łódzkiej.

[19] Compuware APM application lifecycle performance management http://www.compuware.com/en_us/application-performance-management/products/lifecycle-performance-management.html

[20] "Google secrets. How to get a top 10 Ranking on the most important search engine in the world" Blue Moose Webworks Inc., 2003, ISBN 0-9728588-0-6 .

Authors and affiliations

***Marek Majchrzak*– Chapter 1**
Capgemini Polska, Wrocław, Poland
majchmar@gmail.com

***Łukasz Stilger* – Chapter 1**
Capgemini Polska, Wrocław, Poland
lukasz.stilger@capgemini.com

***Aneta Poniszewska-Marańda* – Chapter 2**
Institute of Information Technology, Lodz University of Technology, Poland
aneta.poniszewska-maranda@p.lodz.pl

***Rafał Włodarski* – Chapter 2**
Institute of Information Technology, Łódź University of Technology, Poland
r.wlodarski89@gmail.com

***Mariusz Postol* – Chapter 3**
Institute of Information Technology, Łódź University of Technology, Łódź, Poland, mariusz.postol@p.lodz.pl

***Emilia Mendes* – Chapter 4**
Department of Software Engineering, Blekinge Institute of Technology, Karlskrona, Sweden, emilia.mendes@bth.se

***Krzysztof Wnuk* – Chapter 4**
Department of Software Engineering, Blekinge Institute of Technology, Karlskrona, Sweden, krzysztof.wnuk@bth.se

***Zbigniew Huzar*– Chapter 5**
Faculty of Computer Science and Management, Wrocław University of Technolog, Wrocław, Poland, zbigniew.huzar@pwr.edu.pl

***Małgorzata Sadowska*– Chapter 5**
Faculty of Computer Science and Management, Wrocław University of Technology, Wrocław, Poland, m.sadowska@pwr.edu.pl

***Bogumila Hnatkowska* – Chapter 6**
Wroclaw University of Technology, Wrocław, Poland, bogumila.hnatkowska@pwr.edu.pl

***Dariusz Gall* – Chapter 7**
Wrocław University of Technology, Wroclaw, Poland, dariusz.gall@pwr.edu.pl

***Anita Walkowiak*– Chapter 7**
Wrocław University of Technology, Wroclaw, Poland
anita.walkowiak@pwr.edu.pl

Stan Jarzabek – Chapter 8
Faculty of Computer Science, Bialystok University of Technology, Bałystok, Poland, s.jarzabek@pb.edu.pl

Kuldeep Kumar – Chapter 8
Department of Computer Science, School of Computing
National University of Singapore, Singapore, kuldeepkumar@u.nus.edu

***Tomasz Rogalski* – Chapter 9**
Rzeszow University of Technology, Department of Avionics and Control Systems, Rzeszów, Poland, trogalski@prz.edu.pl

Sławomir Samolej – Chapter 9
Rzeszow University of Technology, Department of Avionics and Control Systems, Rzeszów, Poland, ssamolej@prz.edu.pl

Christian Berger – Chapter 10
Department of Computer Science and Engineering Chalmers, University of Gothenburg, Gothenburg, Sweden, christian.berger@gu.se

***Rakesh Rana* – Chapter 10**
Department of Computer Science and Engineering Chalmers, University of Gothenburg, Gothenburg, Sweden, rakesh.rana@gu.se

Miroslaw Staron – Chapter 10
Department of Computer Science and Engineering Chalmers, University of Gothenburg, Gothenburg, Sweden, miroslaw.staron@gu.se

Jarosław Wojciechowski – Chapter 11
Computer Center - Lodz University Of Technology, Łódź, Poland
jaroslaw.wojciechowski@p.lodz.pl