Why Information Systems Modelling Is Difficult

HANNU JAAKKOLA, Tampere University of Technology, Finland
JAAK HENNO, Tallinn University of Technology, Estonia
TATJANA WELZER DRUŽOVEC, University of Maribor, Slovenia
BERNHARD THALHEIM, Christian Albrechts University Kiel, Germany
JUKKA MÄKELÄ, University of Lapland, Finland

The purpose of Information Systems (IS) modelling is to support the development process through all phases. On the one hand, models represent the real-world phenomena – processes and structures – in the Information System world and, on the other hand, they transfer design knowledge between team members and between development phases. According to several studies there are reasons for failed software projects in very early phases, mostly in bad quality software requirements` acquisition and analyze, as well as in lacking design. The costs of errors are also growing fast along the software life cycle. Errors made in software requirements analyze are increasing costs by the multiplying factor 3 in each phase. This means that the effort needed to correct them in the design phase is 3 times, in the implementation phase 9 times and in system tests 27 times more expensive than if they would be corrected at the error source; that means in the software requirements analyze. This also points out the importance of inspections and tests. Because the reasons for errors in the requirements phase are in lacking requirements (acquisition, analyze) which are the basis of IS modelling, our aim in this paper is to open the discussion on the question "Why is Information Systems modelling difficult?". The paper is based on teachers' experiences in Software Engineering (SE) classes. The paper focuses on the modelling problems on the general level. The aim is to provide means for the reader to take these into account in the teaching of IS modelling.

Categories and Subject Descriptors: **D [Software]**; **D.2 [Software Engineering]**; **D 2.1 [Requirements / Specifications]**; **D 2.9 [Management]**; **H [Information Systems]**; **H.1 [Models and Principles]**; **H.1.0 [General]**

General Terms: Software Engineering; Teaching Software Engineering, Information Systems, Modelling

Additional Key Words and Phrases: Software, Program, Software development

1. INTRODUCTION

The purpose of Information Systems (IS) modelling is to establish a joint view of the system under development; this should cover the needs of all relevant interest groups and all evolution steps of the system. The modelling covers two aspects related to the system under development – static and dynamic. A conceptual model is the first step in static modelling; it is completed by the operations describing the functionality of the system. These are, along the development life cycles, cultivated further to represent the view needed to describe the decisions made in every evolution step from recognizing the business needs until the final system tests and deployment. The conceptual model represents the relevant concepts and their dependences in the terms of the real-world. Further, these concepts are transferred to IS concepts on different levels.

The paper first focuses in the basic principles related to IS modelling. The topics selected are based on our findings in teaching IS modelling. The list of topics covers the aspects that we have seen as difficult to understand by the students. The following aspects are covered: Variety of roles and communication (Section 2), big picture of Information Systems development (Section 3), role of abstractions and views (Section 4), characteristics of the development steps and processes (Section 5), varying concept of concept (Section 6) and need for restructuring and refactoring after IS deployment (Section 7). Section 8 concludes the paper.

These different points of view give – at least partial – answers to our research problems: Why Information Systems modelling is difficult to teach? Why this topic is important to handle? In our

Authors' addresses: Hannu Jaakkola, Tampere University of Technology, Pori, Finland, hannu.jaakkola@tut.fi; Jaak Henno, Tallinn University of Technology, Tallinn, Estonia, jaak.henno@ttu.ee; Tatjana Welzer-Družovec, University of Maribor, Maribor, Slovenia, tatjana.welzer@um.si; Jukka Mäkelä, University of Lapland, Rovaniemi, Finland,jukka.makela@ulapland.fi; Bernhard Thalheim, Christian Albrechts University, Kiel, Germany, thalheim@is.informatik.uni-kiel.de.

In: Z. Budimac, Z. Horváth, T. Kozsik (eds.): Proceedings of the SQAMIA 2016: 5th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Budapest, Hungary, 29.-31.08.2016. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073).

work we recognized problems in learning the principles of Information Systems modelling. If these problems are not understood, the software engineers' skills are not at the appropriate level in industry. The paper could also be understood as a short version of the main lessons in Software Engineering (SE).

2. UNDERSTANDING THE ROLES AND COMMUNICATION

Software development is based on communication intensive collaboration. The communication covers a variety of aspects: Communication between development team members in the same development phase, communication between development teams in the transfer from one development phase to the next one, and communication between a (wide) variety of interest groups. The authors have handled the problems related to collaboration in their paper [Jaakkola et al. 2015]. Figure 1 is adopted from this paper.

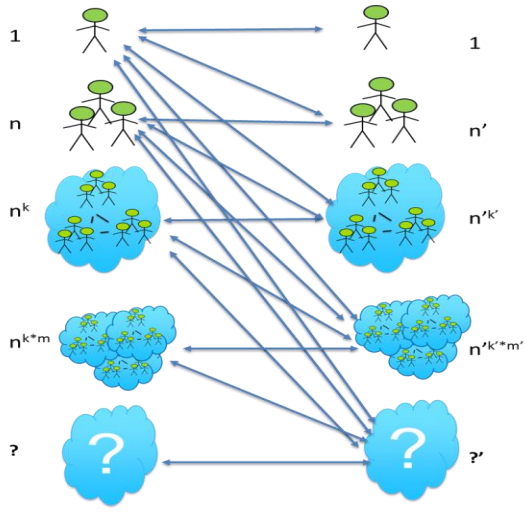


Fig. 1. Degrees of collaboration complexity [Jaakkola et al. 2015].

The elements in Fig. 1 cover different collaboration parties (individual, team, collaborative teams (in the cloud), collaboration between collaborative teams (cloud of clouds) and unknown collaboration party (question mark cloud). The collaboration situations are marked with bidirectional arrows. Without going into the details (of the earlier paper) the main message of the figure is the fast growing complexity in collaboration situations (1*1; 1*n'; nk*n'k'*m'). In increasing amounts there are also unknown parties (question mark cloud; e.g. in IS development for global web use), which increases the complexity. The explicit or implicit (expected needs of unknown parties) communication is based on messages transferred between parties. Interpretation of the message is context-sensitive (i.e., in different contexts the interpretation may vary). The message itself is a construction of concepts. The conceptual model represents the structure of concepts from an individual collaborator's point of view. An important source of misunderstanding and problems in collaboration is an inability to interact with conceptual models.

In this paper we concentrate on two important roles – the Systems Analysts and the customer (variety of roles). The starting point is that the Systems Analysts are educated in ICT Curricula and they should have a deep understanding of the opportunities provided by ICT in business processes. The customer should present the deep understanding of the application area instead, and they are not expected to be ICT experts. What about the Systems Analyst – should he/she also be expert in ICT

applications? We will leave the exact answer to this question open. Our opinion is that, first and foremost, the Systems Analyst should be a model builder who is filtering the customer's needs and, based on abstractions, finally establishes a baseline as a joint view – from the point of view of all interest groups - to the system under development. The joint view is based on communication between different parties. The Standish Group has reported communication problems between Systems Analysts and users - lack of user involvement – to be one of the important sources of IS project failures (Chaos report [Standish Group 2016]).

3. UNDERSTANDING THE BIG PICTURE OF MODELLING

Information System development is based on two different views, the static one and the dynamic one, having a parallel evolution path. All this must be recognized as a whole already at the beginning, including the evolution of requirements through the development life cycle. Figure 2 illustrates flow in the "big picture" of modelling. In the upper level of IS development the approach always follows the principles of a "plan driven" approach, even in the cases where the final work is based on Agile or lean development.

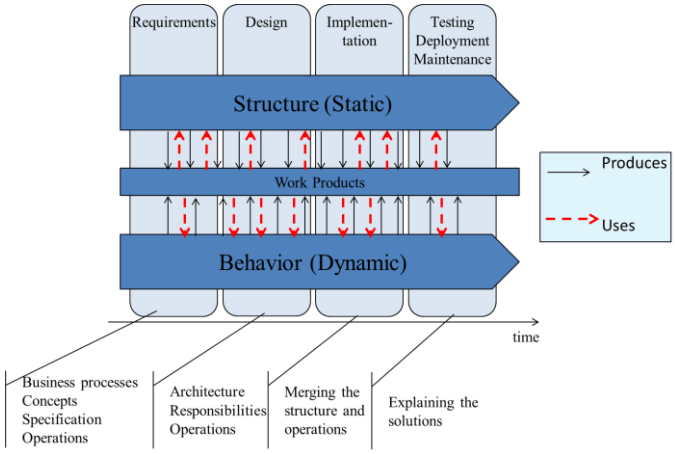


Fig. 2. Static and dynamic evolution path in Information System modelling.

In this paper we do not focus on the discussion of the current trends in software development models. The traditional plan-driven (waterfall model based) approach is used. It is an illustrative way to concretize the basic principles of the constructive approach in software development. The same principles fit in all approaches, from plan-driven (waterfall based) to agile, lean, component based, software reuse based etc. approaches. According to Figure 2 the Information System development has its roots in business processes (understanding and modelling). Business processes represent the *dynamic approach* to the system development, but also provide the means for the *preliminary concept recognition* and the operations needed to handle them. *The conceptual model* is a *static structure* describing the essential concepts and their relationships. The Information System development continues further by the specification of the *system properties* (to define the system borders in the form of external dependencies) and transfers the real-world concepts first into the requirement level, and further to the architecture and implementation level concepts. Separation of the structure and behavior is not always easy; people are used to describing behavior by static terms (concepts) and static state by dynamic terms (concepts).

The role of "work product repository" is not always recognized. The development flow produces necessary work products, which are used by other parts of the development flow. *Conformity* between work products must be guaranteed, but is not always understood clearly. Conformity problems, both

in the horizontal (evolution path of work products) and vertical (dynamic vs. static properties) direction are typical.

4. UNDERSTANDING THE ROLE OF ABSTRACTIONS AND VIEWS

The IS development is based on abstractions – finding the essence of the system under development. Figure 3 illustrates the role of abstractions in Information Systems modelling.

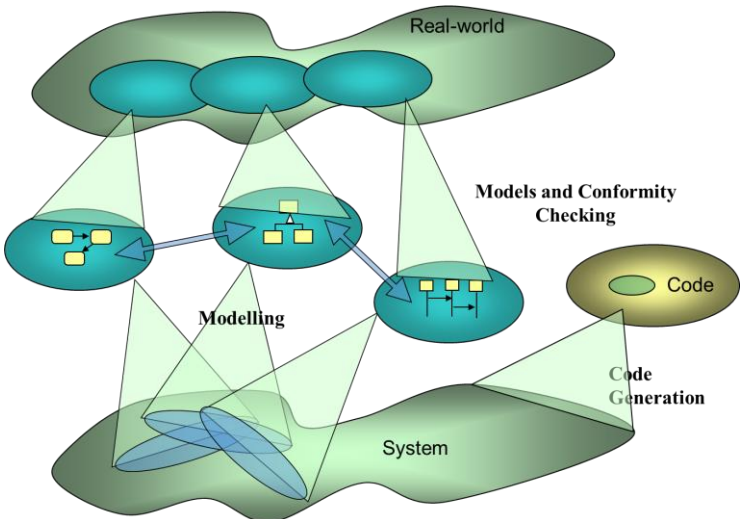


Fig. 3. The role of abstractions [Koskimies 2000; modified by the authors].

The Information System is the representative of the real-world (business) processes in the "system world". The model (set) of Information System describes the real-world from different points of view (*viewpoint*) and a single model (in the terms of UML: Class diagram, state diagram, sequence diagram, …) provides a single *view* to certain system properties. Information System is an abstraction of the real-orld covering such structure and functionality that fills the requirements set to the Information System. Such real-world properties that are not included in the Information System are represented by the external connections of it or excluded from the system implementation (based on abstraction). As seen in Figure 3, the starting point of the model is in the real-world processes, which are partially modelled (abstraction) according to the selected modelling principles; both the static and dynamic parts are covered. The individual models are overlapping, as well as the properties in the real-world (processes). This establishes a need for checking the *conformity between individual models*; this is not easy to recognize. An additional problem related to abstractions is to find the answer to the question "What should be modelled?" and "How to fill the gaps not included in the models?". No clear answer can be given. However, usually the problems in Information Systems relate more to the features that are not modelled than to those that are included in the models. Models make things visible, even in the case that they include some lacks and errors (which are also becoming visible this way).

The Information System development covers a variety of viewpoints to the system under development. Structuring the viewpoints helps to manage all the details of the Information System related data as well as the dependences between these. In this context, we satisfy by referring to the widely used *4+1 View model* introduced originally by Kruchten [Kruchten 1995], because it is referred to widely and was also adopted by the Rational Unified Process specification.

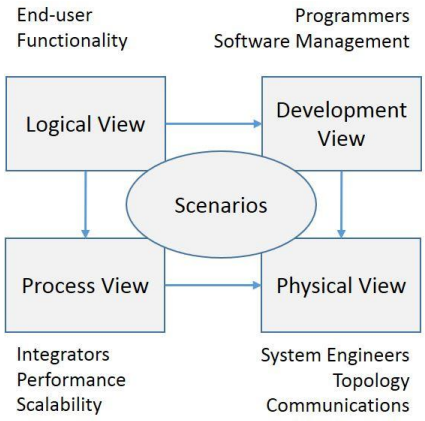


Fig. 4. 4+1 architectural view model (Kruchten 1995; Wikipedia 2016]

The aim of the 4+1 view model (Figure 4) is to simplify the complexity related to the different views needed to cover all the aspects in Information Systems` development; the relations between different views are not always clear. Views serve different needs: A *logical view* provides necessary information for a variety of interest groups, a *development view* for the software developers, a *physical view* for the system engineers transferring the software to the platforms used in implementation, and the *process view* to the variety of roles responsible for the final software implementation. Managing the conformity between the variety of views (models) is challenging. Again, to concretize the role of views in Information Systems modelling, we will bind them to UML (static path related) specifications: Logical view – the main artefact is a class diagram; development view – the main artefact is a component diagram; physical view – the main artefact is a deployment diagram; process view - the artefacts cover a variety of communication and timing diagrams. Dynamic path decisions are specified by a variety of specifications, like state charts, activity diagrams, sequence diagrams and timing descriptions.

One detail not discussed above is the role of non-functional (quality) properties, assumptions and limitations. Without going to the details, we state that they are changing along the development work to functionality, system architecture, a part of the development process, or stay as they are to be verified and validated in qualitative manner.

5. UNDERSTANDING THE CHARACTERISTICS OF THE DEVELOPMENT PATH AND PROCESSES

The purpose of the Information Systems development life cycle models is to make the development flow visible and to provide rational steps to the developer to follow in systems development. There exists a wide variety of life cycle models – from the waterfall model (from the 1960s) as the original one to the different variants of it (iterative – e.g. Boehm's spiral model), incremental, V-model and, further, to the approaches following different development philosophies (e.g. Agile, Lean); see e.g. [Sommerville 2016]. As already noted above, our aim is not to go in detailed discussion of development models. All of them represent in their own way a model of constructive problem solving, having a more or less similar kernel with different application principles.

We selected the V-model to illustrate the development path for two reasons. The origin of the V-model is in the middle of 1980s. In the same issue, both Rook [Rook 1986] and Wingrove [Wingrove1986] published its first version, which has since been adopted by the software industry as the main process model for traditional (plan-driven) software development. Firstly, it separates clearly the *decomposition* part (top-down design) and *composition* part (bottom-up design) in the system evolution, and, secondly, it shows dependences between the early (design) and late (test) steps. An additional feature, discussed in the next Section, relates to the evolution of the concept of concept along the development path.

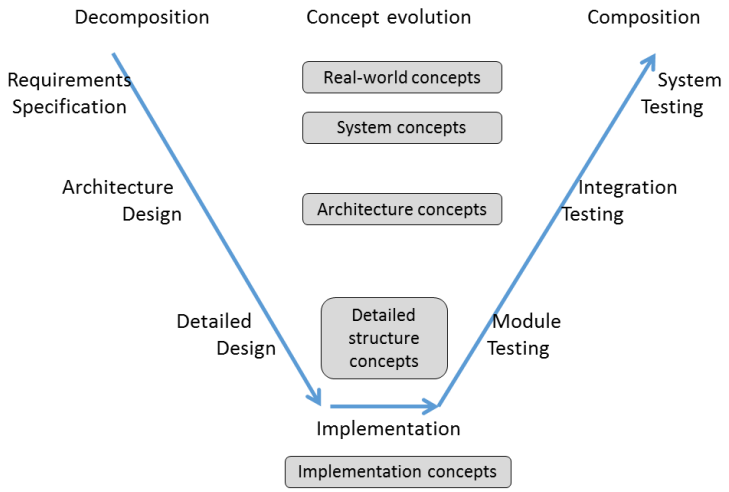


Fig. 5. The V-model of Information System development.

The development activity starts (Figure 5; see also Figure 2) from business use cases (processes) that are further cultivated towards user requirements (functionality) and the corresponding static structure. In the top down direction (left side) the system structure evolution starts from conceptual modelling *in the terms of the real-world*. These are transferred further to the structures representing the requirements set to the Information System (*in terms of the requirements specification*). Architecture design modifies this structure to fill the requirements of the selected architecture (*in terms of the architecture*) having focus especially on the external interfaces of the system. The detailed design reflects the implementation principles, including interfaces between system components and their internal responsibilities. Implementation ends the *top-down design* part of the system development and starts the *bottom-up design*. The goal of the bottom-up design is to collect the individual system elements and transfer them to the *higher level abstractions*, first to *components* (collection of closely related individual elements – in terms of the UML classes) and further to the *nodes*, which are deployable sub-systems executed by the networked devices. The bottom-up modelling includes the *sketching* and *finalizing* phases. An additional degree of difficulty in this "from top-down to bottom-up" elaboration is its iterative character; the progress is not straightforward, but iterative, and includes both directions in turn.

6. UNDERSTANDING THE VARYING CONCEPT OF CONCEPT

Along the development path the abstraction level of the system is changing. This reflects also in the used terminology. This is illustrated in Figure 5`s middle part – concept evolution. In the beginning of the development work the modelling is based on the *real-world concepts* (conceptual model); this terminology is also used in communication between the Systems Analyst and different interest groups. As a part of requirements specification these concepts are transferred to fill the needs of system requirement specification. The terminology (concepts used) represents the requirements level concepts, which do not have (necessarily) 1-1 relation. In architecture design the concepts related to architecture decisions become dominant – i.e. the role of *design patterns* and *architecture style* become important. This may also mean that, instead of single concept elements, the communication is based on *compound concepts*. In practice this may mean that, instead of single elementary concepts (class diagram elements), it becomes more relevant to communicate in the terms of design patterns (observer-triangle, proxy triangle, mediator pair, factory pair, etc.) or in the terms of architecture style (MVC solution, layers, client-server solution, data repository solution). The implementation phase

brings the need for programing level concepts (idioms, reusable assets, etc.). To summarize the discussion, the communication is based on different concepts in different parts of the development life cycle – we call it the *evolution of concepts*.

7. PROACTIVIVE MODELLING - STRUCTURAL AND CONCEPTUAL REFACTORING

Programs model real-life systems and are designed for real, currently existing computer hardware. But our real-life – our customs, habits, business practices and hardware are changing rapidly and our computerized systems should reflect these changes in order to perform their tasks better. Thus, software development is never finished – software should be modified and improved constantly and, therefore, should be designed in order to allow changes in the future. Because of that the design should take into account the need for future changes in a *proactive manner*; otherwise the changes become expensive and difficult to implement and cause quality problems. Proactive modelling is based on the use of interfaces instead of fixed structures, modifiable patterns in design, generalized concepts and inheritance instead of fixed concepts, the use of loose dependencies instead of strong ones, extra complexity in concept to concept relations, etc.

The most common are changes in program structure - structural refactoring, applying a series of (generally small) transformations, which all preserve a program's functionality, but improve the program`s design structure and make it easier to read and understand. Programers` folklore has many names and indices for program sub-structures (design smells), which should be reorganized or removed: Object abusers (incomplete or incorrect application of object-oriented programing principles), bloaters (overspecification of code with features which nobody uses, e.g. Microsoft code has often been called 'bloatware' or 'crapware'), code knots (code which depends on many other places of code elsewhere, so that if something should be changed in one place in your code you have to make many changes in other places too, so that program maintenance becomes much more complicated and expensive). Structural refactoring generally does not change programs` conceptual meaning, thus, in principle, it may be done (half)-automatically and many methods and tools have been developed for structural refactoring [Fowler 1999; Kerievsky 2004; Martin 2008].

Cases of conceptual refactoring are much more complicated. Our habits and behavior patterns change constantly: we are using new technology that was not used commonly at the time of program design, i.e. when the conceptual model was created; increased competition is forcing new business practices; etc. All these changes should also be reflected in already introduced programs and, generally, they also require re-conceptualization of the programs or some parts of them. We will clarify this in the following examples below.

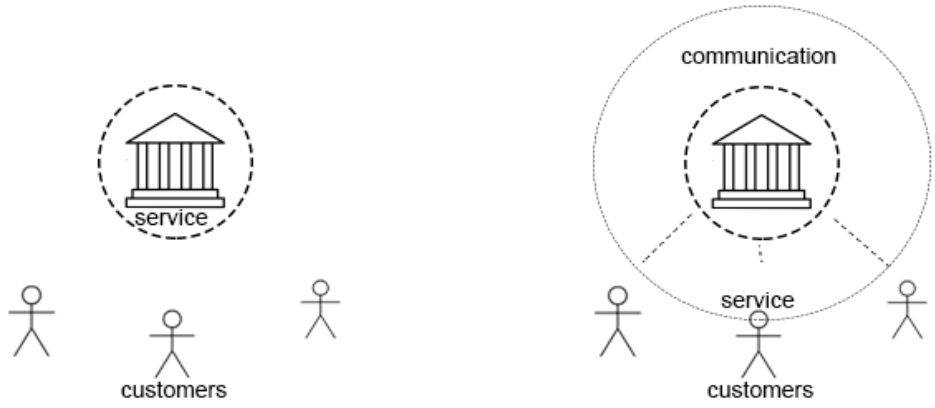
Microsoft, who have often been accused of coupling useful programs (e.g. the Windows OS) with bloatware and crapware, introduced in 2012 a special new service "Signature Upgrade" for "cleaning" up a new PC – you bring your Windows PC to a Microsoft retail store and for $99 Microsoft technicians remove the junk – a new twist in the Microsoft business model.

An even bigger change in the conceptual model of Microsoft's business practices occurred when Microsoft introduced Windows 10. With all the previous versions of the Windows OS Microsoft has been very keen on trying to maximize the income from sales of the program, thus the OS included the subsystem "Genuine Windows" which has to check that the OS is not a pirated copy but a genuine Microsoft product (but quite often also raised the alert "This is not a Genuine Windows! " in absolutely genuine installations). With Windows 10 Microsoft changed by 180^0 the conceptual model of monetizing – it became possible to download and install Windows 10 free of charge! Even more, Microsoft started to foist Windows 10 intensely onto all users of Windows PC and, for this, even changed the commonly accepted functionality of some screen elements: in all applications clicking the small X in windows upper right corner closes the window and its application but, contrary to decades of practice in windowed User Interfaces (UIs) and normal user expectations, Microsoft equated closing the window with approving the scheduled upgrade – this click started the (irreversible) installation of Windows 10. This forced change in the conceptual meaning of a common screen element proved to be wrong and disastrous to Microsoft. A forced Windows 10 upgrade rendered the computer of a

Californian PC user unusable. When the user could not get help from Microsoft's Customer Support, she took the company to court, won the case and received a $10,000 settlement from Microsoft; Microsoft even dropped its appeal [Betanews 2016]. The change in the company's conceptual business policies has created a lot of criticism for Microsoft [Infoword 2016].

Many changes in conceptual models of software are caused by changes in the habits and common practices of clients which, in turn, are caused by the improved technology they use. Once functioning of many public services was based on a living queue – the customer/client arrived, established their place in the queue and waited for his/her turn to be served. In [Robinson 2010] a case of conceptual modelling is described for designing a new hospital; a key question was: "How many consultation rooms are required"? The designer`s approach was based on data from current practice: "Patient arrivals were based on the busiest period of the week – a Monday morning. All patients scheduled to arrive for each clinic, on a typical Monday, arrived into the model at the start of the simulation run, that is, 9.00am. For this model (Fig. 6a) we were not concerned with waiting time, so it was not necessary to model when exactly a patient arrived, only the number that arrived".

This approach of conceptual modelling of a hospital's practice ignores totally the communication possibilities of patients. In most European countries, computers and mobile phones are widespread and used in communication between service providers and service customers, and this communication environment should also be included in the conceptual model of servicing. Nowadays, hospitals and other offices servicing many customers mostly all have on-line reservation systems, which allow customers to reserve a time for visit and not to rush with the requirement to reserve a time for the visit on Monday morning or staying in the living queue. A new attribute, *Reservation,* has been added to the customer object. The current reservation system is illustrated in Fig. 6b.



(a) In 2010 (b) Nowadays

Fig. 6. The conceptual model of mass service (Robinson 2010): (a) In 2010 (Robinson 2010), (b) Nowadays.

Cultural/age differences can cause different variations of the conceptual model of reservation systems. For instance, in Tallinn with its large part of older technically not proficient population (sometimes also non-Estonian, i.e. have language problems) for some other public services (e.g. obtaining/prolonging passports, obtaining of all kind of permissions/licenses) the practice of reserving time has not yet become common. In the Tallinn Passport Office (https://www.politsei.ee/en/) everyone can make a reservation for a suitable time [Reservation System (2016)], but many older persons still appear without one. In the office customers with reservations are served without delay, but those who do not have a reservation are served in order of appearance, which sometimes means hours of waiting. Seeing how quickly customers with reservations are served is a strong lesson for them – here the conceptually (new for them) system of reservations does not only change the practice of office, but also

teaches them new practices, i.e. here, innovation in technology (the Reservation System) also changes the conceptual practices of customers.

Practical use of a reservations system sometimes also forces changes to the system itself. For instance, most of the doctors in Estonia, Finland and Slovenia work with reserved times. However, sometimes it happens that a customer who has a reserved time is not able to come. Medical offices require cancellation (some even practice a small fine if cancellation is not done in-time). In order to find a replacement, the office should be able to contact potential customers (who have a reservation for some future time). Thus, two more fields were introduced to the object model of the customer: *Mobile phone number, Minimal time required to appear at the service.* A new functionality was also added to the reservation system: if somebody cancels, the reservation system compiles a list of potential 'replacement' customers, i.e. customers, who have a future reservation and are able to appear at the service provider in time and the office starts calling them in order to agree a new reservation.

8. CONCLUSION

There is a lot of evidence that the most serious mistakes are made in the early phases of software projects. Savolainen [Savolainen 2011] reports in her Thesis and studies (based on the analyze of tens of failed software project data) that, in almost all the studied cases, it was possible to indicate the failure already before the first steps of the software project (pre-phases, in which the base for the project was built in collaboration between the software company and customer organization). The errors made in early phasesare tend to accumulate in later phases and cause a lot of rework. Because of that, the early phase IS models have high importance to guarantee the success of IS projects. The Standish Group Chaos Reports cover a wide (annual) analyze of problems related to software projects. The article of Hastie & Wojewoda [Hastie & Wojewoda 2015] analyzes the figures of the Chaos Report from the year 2015 (Figure 7). The Chaos Report classifies the success of software projects in three categories: Successful, challenged and failed. The share of failed projects (new definition of success factors covers the elements on time, on budget with a satisfactory result) has been stable on the level a bit below 20% (Figure 7, left side). The suitability of the Agile process approach seems also to be one indication for success in all project categories – even in small size projects. The Report has also analyzed the reasons on the background of the success (100 points divided): Executive sponsorship (15), emotional maturity (25), user involvement (15), optimization (15), skilled resources (10), standard architecture (8), agile process (7), modest execution (6), project management expertise (5) and clear business objectives (4).

All projects (modern resolution)					
%	2011	2012	2013	2014	2015
Successful	29 %	27 %	31 %	28 %	29 %
Challenged	49 %	56 %	50 %	55 %	52 %
Failed	22 %	17 %	19 %	17 %	19 %

Agile vs. Waterfall				
Size	Method	Successful	Challenged	Failed
All projects	Agile	39 %	52 %	9 %
	Waterfall	11 %	60 %	29 %
Large size projects	Agile	18 %	59 %	23 %
	Waterfall	3 %	55 %	42 %
Medium size projects	Agile	27 %	62 %	11 %
	Waterfall	7 %	68 %	25 %
Small size Projects	Agile	58 %	38 %	4 %
	Waterfall	44 %	45 %	11 %

Fig. 7. The success of software projects in 2011-2015 [based on Hastie & Wojewoda 2015]).

The Agile vs. Waterfall results are opposite to the ICSE Conference presentation of Barry Boehm (2006; 2006a) related to the software engineering paradigms. According to him, the Agile approach is best suited to small projects that are non-critical and include high dynamics in requirement changes, implemented by skilled people in organizations used to chaos.

The purpose of our paper has been to point out important aspects related to IS modelling. The following aspects are discussed:

- The variety of roles and their responsibilities in IS development;
- Understanding the big picture of the modelling is difficult;
- Understanding the abstractions is difficult;
- Understanding the role of the development path phases and their interrelations is difficult;
- Concepts are varying along the development life cycles;
- Understanding the views and the development flow is difficult.

In teaching IS modelling all this must be taken into account. The experience based realizations of these main problem categories cover at least the following aspects:

- Inadequate skills in using modelling languages – used in an incorrect way (e.g. including dynamic features in static diagrams – functionality in class diagrams; problems to understand what is a class and what is the connection between classes);
- Low motivation to make modelling – preference given to implementation and coding without modelling;
- In a teaching context, we never have the opportunity to solve real modelling problems; small sub-problems instead;
- Difficulties in understanding what is a dynamic and what is a static concept;
- Models are not complete descriptions of the system – what to leave out and what to include; what are essential concepts and what are not; how to fill the gaps;
- Business rules are difficult to understand, because students do not know the real application environment;
- Missing motivation to learn new approaches – "I already know" – syndrome;
- Expectations of the existing skills – in reality reset and relearn is needed because of the "antipattern" type of behavior;
- Models are overlapping and it is difficult to get the different views to conform;
- Models belonging to different abstraction levels are using different concepts – mismatch of conceptual thinking.

Our results are in favor with the analyze and studies discussed above. It is important to benchmark the existing studies and to transfer the "lessons learned" into study modules. The problem also is the fact that the IS modelling is a by-product as a part of other teaching topics and, finally, practicing it remains on the artificial level instead of focusing on the modelling of real (large) information systems. Our original aim was also to handle the topic "How to teach IS modelling?", but this will be left for further studies that apply the findings of this paper in curriculum and course implementation design in different educational environments. Nevertheless, the authors, as well as other readers, have an opportunity that, on the basis of the research done introduces some innovative steps and solutions out of the box in their own teaching process what will be, together with the reached experiences, an added value to the present paper for the further studies. We also expect a valuable contribution from the discussion at the conference, while changing of the curricula and its implementation is always a demanding step.

REFERENCES

Betanews (June 26th, 2016), 2016. Microsoft pays out $10,000 for forcing Windows 10 on California woman. http://betanews.com/2016/06/27/microsoft-windows-10-payout/. Retrieved in July 12th, 2016.

Barry Boehm, 2006. A View of 20th and 21st Century Software Engineering. Key note presentation in ICSE 2016 Conference. Presentation slides. ICSE and ACM. http://www.inf.fu-berlin.de/inst/ag-se/teaching/S-BSE/054_20th-and-21st-century-sweng.pdf. Retrieved on July 12th, 2016.

Barry Boehm, 2006a. A View of 20th and 21st Century Software Engineering. Key note paper in ICSE 2016 Conference. ICSE and ACM. https://www.ida.liu.se/~729A40/exam/Barry%20Boehm%20A%20View%20of%2020th%20and%2021st%20Century%20Softw

are%20Engineering.pdf. Retrieved on July 12th, 2016.
M. Fowler, 1999. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional. p 464, ISBN-13: 978-0201485677.
S. Hastie, S. Wojewoda, 2015. Standish Group 2015 Chaos Report – Q&A with Jennifer Lynch. https://www.infoq.com/ articles/ standish-chaos-2015. Retreived in July 12th, 2016.
Infoword (March 14th, 2016), 2016. Microsoft upgraded users to Windows 10 without their OK. http://www.infoworld.com/ article/3043526/Microsoft- windows/microsoft-upgraded-users-to-windows-10-without-their-ok.html. Retrieved on July 12th, 2016.
H. Jaakkola, J. Henno, B. Thalheim, B. and J. Mäkelä, (2015. Collaboration, Distribution and Culture – Challenges for Communication In Biljanovic, P. (Ed.), Proceedings of the of the MIPRO 2015 Conference. Opatija, Mipro and IEEE, 758-765.
J. Kerievsky, 2004. Refactoring to Patterns. Addison-Wesley Professional, p. 400. ISBN-13: 978-0321213358.
K. Koskimies, 2000, Oliokirja. Talentum, Helsinki. ISBN-13: 9789517627207, ISBN-10: 9517627203.
Philippe Kruchten, 1995. Architectural Blueprints — The "4+1" View Model of Software Architecture. IEEE Software 12, 6. (September 1995), 42-50.
R.C. Martin, 2008. Clean Code: A Handbook of Agile Software Craftsmanship. Prencice Hall 2008, p 464, ISBN-13: 978-0132350884
Stewart Robinson, 2010. Conceptual Modelling: Who Needs It? SCS M&S Magazine 1,2 (April 2010). http://www.scs.org/magazines/2010-04/index_file/Articles.htm.Retreived on July 12th, 2016.
Paul Rook, 1986. Controlling software projects. Software Engineering Journal 1,1 (January 1986), 7-16.
Paula Savolainen, 2011), Why do software development projects fail? - Emphasising the supplier's perspective and the project start-up. PhD Thesis, Univcersity of Jyväskylä. Jyväskylä studies in computing (136).
Ian Sommerville, 2016. Software Engineering. Pearson Education Limited. ISBN-13: 978-0133943030; ISBN-10: 0133943038.
Standish Group, 2016. CHAOS Report 2016: The Winning Hand. https://www.standishgroup.com/store/. Retrieved on July 12th , 2016.
Wikipedia 2016. 4+1 View Architectural model. https://en.wikipedia.org/wiki/4%2B1_architectural_view_model. Retrieved on July 12th, 2016.
Alan Wingrove, 1986. The problems of managing software projects. Software Engineering Journal 1,1 (January 1986), 3-6.

4:40 H. Jaakkola, J. Henno, T. Welzer Družovec, J. Mäkelä, B.Thalheim