Proceedings of the 10th WSEAS International Conference on COMPUTERS, Vouliagmeni, Athens, Greece, July 13-15, 2006 (pp818-825)

On Software Design and Development Supporting Requirements Formulation

DENISS KUMLANDER
Department of Informatics
Tallinn University of Technology
Raja St.15, 12617 Tallinn
ESTONIA

Abstract: The software engineering is evolving on the permanent base. This evolution highlights some issues that were hidden so far or appeared with new techniques. In this paper an idea of supporting software design is proposed and extended to the whole software development cycle. Thereafter communication gaps are examined, which are important to avoid for successful implementation of the proposed technique. Several methods to avoid such gaps are introduced. Finally two companies' cases are described where the proposed methodology is applied. Those companies are quite typical therefore advices can be used in other companies having the same troubles.

Key-Words: software engineering, software design, requirements gathering, supporting software design

1 Introduction

The ultimate goal of developing any software is to provide customers with tools that will help them run their business in a better way. Nowadays increasing competition and globalisation of business demands much higher quality of the released software, much shorter development cycle and increased flexibility of defining requirements. The proper software implementation over the low quality one provides benefits for both projects sides – mostly because of saving resources, quicker applying the software that business needs to have, better imago for software providers and some others. Unfortunately many software projects are far from the described goals and the number of failing projects is still high.

The goals of software engineering are to make the software development simpler and the resultant software better [3]. There are a lot of software development principles that are more or less similar and are basing on some common for all techniques. The software industry advanced a lot by applying those technologies and solving major problems we had in the past. At the same time it also brings forward problems that so far looked to be easy to avoid. In this paper we are going to propose a methodology allowing stabilising the software development process by increase the quality and decreasing the implementation time. The paper extends our previous researches [4, 5] to the full software development cycle. The paper also examines problems arising because of bad communication between different team members and demonstrates how it can be solved.

The paper is organised as follows. The section 2 briefly describes the software development work cycle. The following section introduces the supporting software design principle and discusses its different aspects. The question of communication gaps is examined in the section 4. Some project cases where the proposed methodology is revised in the section 5. The last section concludes the paper.

2 Classical Work Cycles

2.1 Software development

There are a lot of models for software development and some of them are quite basic ones like the waterfall or spiral software development [1] methodologies. It is a bit hard to build a model that could adequately reflect all those on a general level, but the following very simple one should demonstrate basic principles from most of them.

The model contains three parts: requirements, then design, thereafter software (program code) and this can result in developing new requirements.

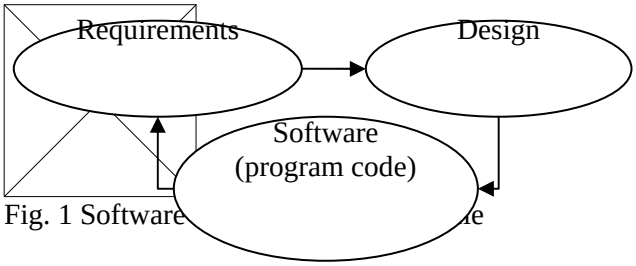

Fig. 1 Software

Of course this model could include more steps, could be presented as a model of phases, activities

etc., but it still demonstrates the software development process quite adequately. A new software development usually starts from the requirements formulation. Those could be formulated by special persons called business analysts, could be provided by customers or could be a result of requirements gathering process on the customer sites, i.e. collaboration between business analysts and customers. Anyway it is a set of documents describing what and how should be done from the customer/business/functionality point of view. The next step is developing a design that defines what could be done and how it will be done from the technical point of view, how processes will be connected and implemented in the future software package. The design is build basing on the requirements document, which can be seen as an input parameter for the design formulation. The design is implemented in a form of the software programming code, which is shipped to the customer and contains all required features. New requirements could arise after users started to use the implemented software and a new cycle will start.

2.2 Reengineering

The software reengineering process is very similar to the previously described one and also starts from the requirements formulation. The main difference is that requirements are formulated basing on an existing software package for rebuilding or reengineering that. Therefore the existing software is usually a starting point of the development cycle. The diagram contains the same steps and is just transformed by rotating.

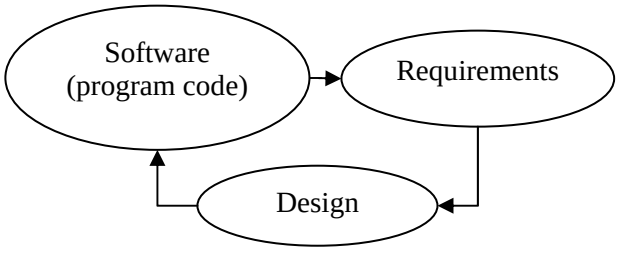

Fig. 2 Software reengineering work-cycle

In this software development case the source system is studied at the first step to find out existing processes, bottlenecks, number and kind of documents in use and so forth. New requirements are also gathered as it was done in the new software development process described above and then mixed up with the existing logic to produce new system requirement, then design and thereafter a new system.

2.3 Special methodologies for different software companies types

Previously described work cycles are general and demonstrate overall software development procedure and steps connections. The quick development of the software industry and increasing level of globalisation, which results in the higher competition between software providers, makes software companies to look for specific segments to operate in. Such segments are very different and therefore require sometimes different approaches to the software development process. The described work-cycles' steps are basic guidelines from which different variations are derived. Each specific methodology depends on a software company (software department) profile and parameters like type of projects, type of the company etc.

A company that implements products for external customers usually uses a work cycle typical for the agility type methodologies. It contains iterations of the fixed size, which results in a complete software package ready to be installed and used by the customer. The planning for the next iterations is a bit light, as the company doesn't know in advance what the customer will want to do afterwards. The customer can freeze the product; the next release could be less profitable than some other customers' software projects etc. It is very similar to building a wall from bricks from the project planning point of view: all bricks (projects iterations) have the same size and are independent, so instead of continuing the previous one the software company could start another. One more option is to move iterations between different teams. The independency between blocks is reached by making iterations that are complete and ready to be shipped.

Another typical type of a software company is a software department inside a large company that uses IT solutions to support the main business. The main target of that software department is to optimise with their IT solutions the overall company efficiency, so the department efficiency is not important by itself. The department should provide high level services to other departments and usually deals either with one or with a limited number of software projects. Besides the customer of the department locates close and this simplify requirements formulation process, software reviewing process and so forth. There are a set of methodologies and development framework that suits perfectly to such software developers. Those include different analysis processes starting from the business logic and ending with infrastructure, networking etc analyses that are deep allowing

increasing the performance in all elements of the developed software.

3 Supporting software design

3.1 Essence of the supporting software design

The classical software development work-cycle shows that any next step of the software development is done only if the previous one is completed. For example, the design can be started only after the requirements are formulated and provided to software designers. There are several basic ways of gathering requirements: customers could provide those; the software company business analyst will formulate or the first and the second one will be combined. Unfortunately the real world is not perfect and requirements are not perfect and complete as those should be from the theoretical approach point of view. There could be a loss of information, which is described in the later chapter of this article. The customer could formulate requirements wrong – sometimes they don't know exactly what they would like to have and this will be detected only when the developed software is reviewed etc. That is why design and programming are done sometimes despite requirements, programming despite design etc. It happens since some number of errors is found in the previous steps documents that makes impossible to do the following steps without correcting those, and usually this correction happens on those next steps instead of returning back. Notice also that nowadays business world is changing very fast forcing companies to be flexible and may be change requirements during the development cycle. The globalisation adds pressure to software companies forcing them to compete with many other software developers and provide highest services. Therefore sometimes there is no time to make the full cycle before errors will be fixed especially if those were detected on early project stages. There is a need for the software development approach that could include efficient feedback information flow for each cycle's step. The central idea of the supporting design is to use design to verify requirements, design to get all information and requirements from customers including information on the requirements uncertainty and so forth. The approach proposes to see the design as an additional tool that could help in formulating requirements and breaks the rule that requirements should be completely formulated first and only then the design; having those steps in parallel. This principle could be extended to other steps also and should be defined as using each step to help doing correctly the previous one. The idea is in making different steps team members to collaborate, provide feedback and then the quality level of the product will increase.

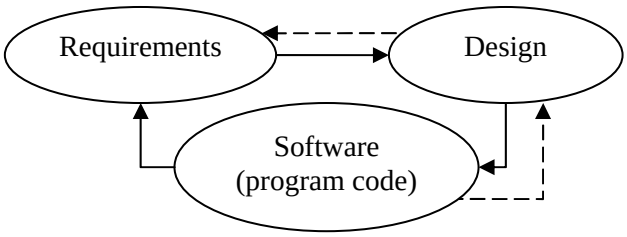

Fig. 3 Applying supporting software design principles to the software development work-cycle

The question of how different team members can collaborate is described in the following subchapters and now only general level principles will be provided. Notice that the collaboration process helps to find errors since in many cases next steps are done more "precisely", i.e. include much more details and therefore could be used to verify previous steps. Ideally the collaboration step should be done using common documents, i.e. documents that are used by collaborating team members, although sometimes it is not possible. Anyway the feedback could be provided either directly if a problem is identified by the next step team members or indirectly by establishing the reviewing process. During this activity team members of the previous step review the work done by the next step team members to ensure that their thoughts are understood and implemented correctly so far. Notice the "so far" part of the sentence. The efficiency of the reviewing process will increase if it is done more times than just once after the task to be reviewed is completed. It will allow finding and fixing errors much earlier saving a lot of resources. Each team could define their own period of recurring reviews depending on the project, team and other properties and goals. A balance should be found between doing reviews and providing feedback too much and too less. Each activity should optimise the overall performance of the software development process.

Finally the prototyping process has to be mentioned. Classically the prototyping means developing a quite restricted model (sometimes it does include only the user interface) for verifying requirements and design. The proposed methodology is much wider and could be described as extending prototyping to each step and to each type of activity/document to be produced, i.e. prototyping of requirements, prototyping of design and so forth. Now we are about to study close each

connection of the previously described software development work-cycle.

3.2 Design and requirements formulation

The first connection between activities to be reviewed is the requirements formulation and the design phase followed by that. The ideal requirements document contains all necessary information about what and how it should be done and the design transforms it into the technical description including programming languages syntax for all features that can be done using one or another technology. Unfortunately requirements are not perfect and usually uncertain because of errors, missed information and sometimes could be changed during the project because of new regulations that customer should adopt etc. This demonstrates that there is a gap between requirements and design and sometimes designers decide how the system will look like by themselves. This could produce many errors as such design could be invalid. Therefore the collaboration between the requirements gathering team and the design team is needed to do required features right. This allows moving new information on later design stages to the design team and having a feedback on errors and later corrections. If the collaborative team is created then requirements will be both correct and correspond to real customers needs instead of been list of one person wrongful suggestion on what is needed. The design construction process always requires describing the future system on more detailed level than it used to be in the requirements documents and therefore verifies the requirements. Sometimes the designer also have a deeper knowledge about previous releases and could have a broader vision about what places of the product could be affected by new features and provides to the requirements formulating team this information if they have missed something. The close work of those teams will ensure the correctness of a new system. The central idea of the supporting software design is to help verify requirements and reach customers' goals using design; do design so that it will contain all information we have in requirements including information on those uncertainty.

3.3 Development and design

Another process to be reviewed is the design and the development phases' connection. The ideal design, as it is described the literature usually, should be so deep that developers could program by it just converting the designed functionality into a programming language. The design should contain all elements like the business logic, a description of objects and functions allowing them to cooperate and much more. Moreover there are a lot of ideas of automating converting the design directly into code [6] that eliminates developers from the software development process. Unfortunately this situation is an ideal one and a lot of software companies have troubles applying these plans. The first problem is that the development of design on a very low level, which is needed for the automatic code generation, requires a lot of efforts. It is the same complex from the amount of work point of view as the actual programming. Notice also that it is easier to find developers than to find the same number of designers. Therefore design usually contains less information that is required for the automatic generation of the ready to ship project and developers are completing the programming phase. Another reason why developers are still presented is that they are able to identify a lot of problems in the design and they do that. It can be seen as a perfect testing method for the design completeness, since developers are writing their code on the lowest level for the project and always have to consider all cases - like all branches of the conditional structures. If any code's branch is missing then developers see that much better than designers. Skilled developers are constantly doing the "what-if" analysis writing the code and this brings a lot of designer errors up. The automatic generation programs are not intellectual enough so far to detect a missing code and do the "what–if" analyses.

That some arguments having developers still in the project team and therefore collaboration between software designers and developers should also be carefully considered. The first important question is the design grain. It is quite hard to find a level of the design details that will illustrate completely the designed logic and will not be too complex to produce. There are much more developers than designers and they could and should support the design process by implementing their code and producing an efficient feedback to the designers about design errors. Besides involving into the design a set of new persons could help to find new ideas for the design. This is also a very good motivation factor for developers to stay in the company as they can see and feel that they could affect the project with good ideas. Ideas that are not accepted by designers are also not a problem but rather an advantage for the project. First of all such close work between designers and developers enables identifying incorrect understanding of features and business logic by developers on early

stages – before a lot of resources will be used to program a wrong code. That is why such collaboration is worth to have. Besides ideas that are not accepted by designers allows developers better following the business logic of the problem, the designers ideas and makes development much more efficient.

4 Avoiding Communication Gaps

4.1 Communication gaps description

Problems in communication between members of a project are a very serious danger for the project and could lead to loosing a lot of time and efforts. The efficient feedback cannot be organised without a smooth communication. Moreover the same problem could exist in the classical approach since those affects any communications un-regarding their direction [8]. In this paper the designer-business analyst and designer-developer communications are analysed, although specialists in other areas could extend this analysis to communications between other software development team members like testers, installers etc.

The communication gap can be defined as some kind of a problem in the communication process that makes the transferred information to be either lost or deformed [5]. There are different reasons of communication gaps existence and main of them are listed below. Notice that some communication gaps are specific to a certain communication place (i.e. specific to communication between certain team members) and some of them are general.

One reason of the communication gap could be a physical distance between a requirements definer and a designer workplace. The designer in this situation cannot just walk, for example, into the customer's office and talk face-to-face or ask to review the design/gathered requirements or do other things the designer needs to be done. Besides such a distance force them also to communicate in a "none-visual" manner that usually makes the communication between two different people much more problematic. Researches prove that the "visual" feedback is very important part of the communication process. That field works demonstrate that it provides from 20% to 40% of information [2, 7]. So, lacking of "visual" feedback of an opponent reaction makes the communication problem larger since a lot of important information is hidden. It is also a common problem to organize "enough" meetings with customers since they are usually occupied with their business. The same problem could exist in communication with software company business analysts that are formulating requirements if those are over-occupied with too many projects. Notice that such problems are usually not presented between designers and developers as those are usually grouped into the one team in the same location (so called development department) although it not always true. The globalisation and outsourcing produced now a certain number of teams where designers are located at the main office together with business analysts and developers are located somewhere else. In that scenario designers have the described problems with developers as they cannot continuously force developers to follow the design and developers are lazy enough to ask something over emails. A lot of outsourcing using companies are facing such problems and it is clear that they need to improve the communication between those teams.

Another sources of gaps come from one more general level problem of communication between persons, which is explained by different experience, skills, available information, life's and work's environments and culture backgrounds. This problem occurs on any level of team members' communication since they are on different positions that means are having very different backgrounds and knowledge sets. This problem presents also in the communication to external persons like customers who are interviewed on requirements etc. The interviewer can miss important information that the customer's representative does provide or can miss an area to ask about because of that.

One more common problem is a form of the work documents, like requirements and design documents. The document should be a source for collaboration of team members. Notice also that there is an old saying: "If it is not written then it is not said". Project documents that cannot be correctly understood by all project members involved in the communication are a common source of misunderstanding and communication gaps. Another danger is having a set of document instead of common documents, i.e. if each team member hosts its own one. The problem here lies in the un-synchronisation between those and leads to the losing some information during transferring it from a document to document on different project development levels.

Here the communication gaps that were described so far are recollected:

- Impossibility to collaborate quickly;
- Impossibility to do/force to do something if it is needed;

- Loss of information during a communication because of different experience, available information and so forth;
- Loss of information during a communication because of "none-visual communication;
- Loss of information due inappropriate information presentation in collaboration documents;
- Loss of information due to much number of document to be produced;

All those communication gaps could decrease sufficiently the efficiency of applying the supporting design principles and even lead to a project fail. Therefore those should be closely monitored and eliminated as soon as possible. Those represent risks that lead to loosing time, improper spending of resources and so forth.

4.2 Methodologies to avoid communication gaps

Here we are giving an overview of methods to solve or avoid communication gaps described in the previous subchapter.

- Apply the supporting design principles;

This will ensure moving of information, the collaborative work and detection of many errors with efficient feedback. Try to identify errors as soon as possible using a well established reviewing process.

- If it is possible then make development circles shorter in the iterational development;

Divide your project into a set of steps/iterations, for example, ones a month. An output of each development iteration is a part of software (iteration's features) that should be reviewed by the product manager. Customers and the product manager will fill much more comfortable since they will have a better understanding of the work progress.

- Define rules, good practices and processes as clear and simple as possible;

The most common problem in many software companies is lack of rules for documenting and reviewing requirements, designs etc. If nobody is responsible for doing that or such responsibility is shared among two or more persons then nobody will do it and errors that are easy to fix on early stages will be hard to fix later and will demand a lot of resources for rebuilding the project. Notice that rules should be simple and clear otherwise nobody will follow those. Therefore it is not enough to establish rules. Those should be followed as well otherwise there is no point to have them. An ideal solution will be to involve workers into formulating rules since they will surely follow rules they made. They will know why those are established and why those looks like those are.

- Regular meetings between designers and product managers, designer and developers handling the list of open issues.

Short recurring meetings with the clear list of follow up for each participant till the next one. Each team could find their own best time scale. Generally daily 30 minutes meetings are recommended to see the work progress and answer questions so that all will know answers. Such broad discussion and answering makes all knowledgeable about different parts of design, requirements etc., so the information will not flow only through several selected persons. Notice again that all clarifications like for example a detailed description of the workflows need to be documented for the future reference.

- Better preparations for each meeting; each meeting participant should be ready to solve problems;
  - Good timing for the meetings with respect of the time difference if the meeting will include persons from different time zones. Persons should be neither too tired nor sleepy otherwise the communication gap will increase instead of decreasing – the manager is sure that informed the worker about an issue, but the worker missed this information because he is too tired;
  - All documents need to be distributed in advance before the meeting. Otherwise documents are not studied and meeting will not be efficient.
  - Everybody should think about goals and review previous meeting notes to find, which issues are pending.
- Motivate team members to ask question and explain how they do understand requirements, design etc.

First of all notice that some persons are not brave enough to say a word and usually those are the weakest part of the team producing communication gaps and incorrect implementation of code, design etc. Thereafter consider that most obvious issues sometimes are not so obvious and team members can realise that only after they have started discuss them. Notice also that nobody can explain something using own words if he doesn't understand what he is going to say. Motivate team members to explain how they have understood the complex task. This will ensure that he has understood all things correctly and will make him to rethink the task.

- Force to underwrite documents by all involved team members;

For example the functional specification's underwriting will mean – by the business analyst that is it complete and correct; by the designer that he does understand it. Of course it should not be too big restriction for later questions and changing, but at least will force all persons to read and understand what is done instead of postponing this process to later phases, when changes and fixes will be much more expensive.

5 Varying Time Intervals Iterational Development

Today the iterational development is widely adopted in the software development and the work-cycle reflects that also by its cyclic structure. Dividing any project into iterations allows releasing features constantly making the progress of the software development visible to the customer. This process is very important from the avoiding communication errors also since enables finding errors after some iteration instead of at the end of the large project after years of programming. The iterational development [9] can also be seen as a part of the supporting design and development as it is nothing else than an ultimate general level feedback.

The main question to be highlighted here is the iterations size from the period point of view. The agility methodology defines that releases have to be done each month and employs this idea for the flexible planning enabling to switch easily between products/versions to be released next as the iterations have the same size. This model suits very well for software companies having quite a long list of customers with different products or to software departments providing different software to a lot of other departments inside a large corporate organisation. In other cases the iteration size can be defined by the software company and some do release new versions each half year, some do it 3 times a year etc. The supporting software design principles allows stabilizing products and such long releases do fit into the methodologies, but the ideal case will be to have iterations of different sizes. If features to be included requires less time to program than the usual one then this release could and should be done earlier to find out incorrect places. The full cycle feedback is quite important. It can be proved by the fact that user acceptant tests, which are normally done before the product is released, detect certain number of mis-modellings and incorrectly produced features and the varying period iterations could minimize time of finding such errors.

6 Cases Study

Here we will review some companies where the proposed approach was applied to decrease number of errors, mistakes and redesign and this way to decrease spent time on projects and increase personnel productivity. Notice that the described approach is not targeted to fit all software companies' development models. There are a lot of cases where requirements, design etc are stable enough or the approach cannot be used because of some restriction, could be incompatible to company policies etc. At the same time a lot of companies could benefit from applying this methodology right. Moreover some of them do it already trying to eliminate the feedback connections at the same time to fit into standard methodologies. The proposed ideas will make the software development process in such companies much more organised.

6.1 Case 1: a global software company

The fist company to be reviewed here is a global corporation having a software development as one of the main activities. It is a typical telecommunication corporation producing variety of products like telecommunication equipment, software for logistics and warehouses etc. The department we were working with was producing the logistics software. The team size was around 75 persons and was distributed across 2 countries. The business analysts and designers department was located in Western Europe and the development department in India. The major problem the team had when we started to work with them was a low quality of releases and very long development process (up to 2 years) of each software package version. The main reasons of these troubles were identified as: a lot of communications gaps; each version used to be internally released several times, each release took around 6 months and had to be reprogrammed since there were a lot of inaccurately implemented features.

There were two reason of inadequate programming of features: developers were not following design documents because it contained errors and sometimes because documents were hard to understand. The quality of releases was dramatically improved after establishing a collaborative work between designers and developers by selecting several persons from each

side. Designers were surprised a lot by number of ideas developers had and changed their opinion about the developers' department from the "unqualified persons" to the "highly skilled". The quality improvement decreased immediately the time of programming each version up to 43%. Each version was also divided into several iterations of different sizes: first iterations of the smaller size (from included features point of view) and last of the larger size. A lot of inaccurate features' design and implementation were resolved by applying this method on early development stages decreasing the overall time of versions development. A side effect of applying the supporting design principle was the better relationship between project team members and healthier work environment.

6.2 Case 2: an IT department of an insurance company

The second company to be described is a relatively small insurance company's IT department working with variety of different software products designed to meet requirements of different insurance areas. The number of persons working in the department was around 20. The team had to be flexible to meet constantly changing and growing requirements of other departments having quite small resources to implement those. Results of applying the methodology described in this article were much better than we had expected. A lot of requirements were reformulated in the design phase during the collaboration of the persons formulating requirements (usually it was a head of a department that orders software) and designers. Frequent releases allowed stabilizing direction of software packages development. Quick design and development enabled to involved programmers to the formulating requirements and now they see results of they work much earlier and number of needed reprogramming decreased also. The quality of releases before applying the methodology was also very low since developers believed that they had to reprogram each release anyway because of uncertain/incorrect requirements.

7 Conclusion

The central idea of the supporting design is to use design to verify requirements, design to get all information and requirements from customers including information on the requirements uncertainty and so forth. This principle is extended to other software development work cycle steps and is defined as using each step to help doing correctly the previous one. The idea is to make different steps' team members to collaborate, provide feedback. This increases the quality of the product decreasing the time needed to implement it. It is done by building collaboration teams from persons involved into the project on different steps, using shared documents, establishing a reviewing process for each step and using iterational development of varying time intervals. Efficient feedback is one of the methods to avoid certain communication gaps, which are information loosing or deforming during the communication of project team members. Other methods are: shorter development cycles, simple and clear practises and rules, regular well-prepared meetings and the documents underwriting process. The described method has proved its power by applying in several software departments/companies for the real-live projects.

References:

[1] B.W. Boehm, A spiral model of software development and enhancement, *Computer*, Vol. 21, No. 5, 1988, pp. 61-72

[2] K. Hadelich, H. Branigan, M. Pickering, M. Crocker, Alignment in dialogue: Effects of visual versus verbal-feedback, *Proceedings of the 8th Workshop on the Semantics and Pragmatics of Dialogue, Catalog'04*, 2004, pp. 35-40

[3] J.A. Hoffer, J.F. George, J.S. Valacich, *Modern system analyses and design*, Addison Wesley, 1999

[4] D. Kumlander, Providing a correct software design in an environment with some set of restrictions in a communication between product managers and designers, *Proceedings of the Fourteenth International Conference on Information Systems Development: Pre-Conference*, 2005, pp. 1-11

[5] D. Kumlander, Software design by uncertain requirements, *Proceedings of the IASTED International Conference on Software Engineering*, 2006, pp. 224-2296.

[6] Z. Laszlo, T. Sulyan, MOFCOM: A tool for model-based software development, *Proceedings of the IASTED International Conference on Software Engineering*, 2006, pp. 218-223

[7] R. Ludlow, F. Panton, *The essence of effective communication*. Prentice Hall, 1995

[8] M. Rauterberg, O. Strohm, Work organisation and software development, *Annual Review of Automatic Programming*, Vol. 16, 1992, pp 121-128

[9] P.R. Reed, *Developing applications with Visual Basic and UML*, Addison-Wesley, 1999