

LATVIJAS UNIVERSITĀTE

Baiba Apine

**Programmatūras izstrādes procesa
diagnosticēšana un attīstīšana**

SAISTĪTĀS PUBLIKĀCIJAS

(promocijas darba pielikums)



Rīga - 2003

Zinātniskās publikācijas:

Izdevumos no LZP vispāratzīto recenzējamo zinātnisko izdevumu saraksta:

1. Baiba Apine. *"Software Development Risk Management Survey"*. Information Systems Development. Advances in Methodologies, Components, and Management, 2002, Kluwer Academic Publishers, USA, lpp. 241-251.
2. Baiba Apine. *"Measurements and risks Based Method to Support Software Development Process Planning"*. Databases and Information Systems II, selected papers of BalticDB&IS 2002, 2002, Kluwer Academic Publishers, The Netherlands, lpp. 187-199.
3. Baiba Apine. *"Measurements and risks Based Method to Support Software Development Process Planning"*. Proceedings of the Fifth International Baltic Conference, BalticDB&IS 2002, 2002, Tallinn, Estonia, lpp. 3-14.
4. Baiba Apine, Martins Gills, Janis Plume. *"Software Development Process Improvement through Measurements and Requirements Traceability"*. Proceedings of the Fifth International Baltic Conference, BalticDB&IS 2002, 2002, Tallinn, Estonia, lpp. 65-76.
5. Baiba Apine, Ilgvars Apinis, Ojars Krasts, Uldis Sukovskis. *"Meta-model Based and Component Based Approach for Information Systems Design"*. Proceedings of the 4th IEEE International Baltic Workshop: Baltic DB&IS '2000, 2000, Vilnius, Lithuania, lpp. 78-83.
6. Baiba Apine, Uldis Smilts, Uldis Sukovskis. *"Software Measurement Practice to Address Customer Satisfaction"*. Scientific Proceedings of Riga Technical University, 2000, Applied Computer Systems. – 1st thematic issue, lpp. 11 – 18.
7. Baiba Apine. *"Practitioner's approach to software cost estimation"*. Abstracts of the international conference: DB & IS '98, 1998, Riga, Latvia, lpp. 134-140.
8. Baiba Apine, Arnis Kleins, Ojars Krasts, Uldis Sukovskis, Artis Teilans, Vita Zviedre. *"Modelling methodology and tool for business systems: Registrar"*. Abstracts of the international conference: Simulation, Gaming, Training and Business Process Reengineering in Operations, 1996, Riga, Latvia, lpp. 44-45.

Pārējos izdevumos:

9. Baiba Apine. *"A Visualisation and Analysis of Experimentally Gathered Results in an User-Friendly Mode"*. Informatica, 1995, Vol.6, No.4., lpp. 387-396.

SOFTWARE DEVELOPMENT RISK MANAGEMENT SURVEY

Baiba Apine*

1. INTRODUCTION

Software development is rather complex process consisting of different activities. It is dependent on skills' level of different specialists as well as on usage of different technologies. One of the activities supporting software development process is risk management. Risk management requires knowledge and experience from people involved. This paper addresses software development risk management. The survey among software development experts was performed to find the risks, which are live to software developers in Latvia. This paper summarizes the results of the survey.

2. PROBLEM

There are two activities within risk management process requiring special expertise and experience: risk identification and finding activities for risk mitigation. The task was to identify software development process risks for software developers in Latvia and activities for risk mitigation.

3. BACKGROUND

3.1 Definition of Risk

The definition of risk is very simple: the possibility of loss, injury, disadvantage or destruction, as it is defined in Webster's dictionary¹.

* Baiba Apine, Riga Information Technology Institute, Kuldigas iela 45, LV-1083 Riga, Latvia

Risk related to information systems is defined in² as the potential that a given threat will exploit vulnerabilities of an asset or group of assets to cause loss of/or damage to the assets.

In the context of software engineering and development, risk can be defined as the possibility of suffering a diminished level of success within a software-dependent development program³.

Risk is usually measured by a combination of impact and probability of occurrence². Consider a sample, where a potential threat to software development project is leaving of the leading programmer at the peak of the development process. The probability of realizing of this threat is 25%. Let's try to calculate the impact of the threat. We have to arrange new leading programmer from the development team, it will cause the lagging behind the project schedule and paying penalties for \$2000, hire new programmer and train him/her for \$1200. The total impact of the threat is \$3200. The risk is calculated as follow: 25% from \$3200 gives \$800. This is a quantitative risk assessment. Sometimes it is not possible to assess the impact of risk quantitatively. Then qualitative risk assessment is used, where probability and impact are assessed using terms like low, medium, high etc.

There are a lot of risk management methods, which define when and how to identify threats, how to assess and prioritize them, etc.

3.2 Risk Management Methods

Risk management is a practice with processes, methods and tools for managing risks in a project. It provides a disciplined environment for proactive decision making to assess continuously⁴:

1. What could go wrong (risks).
2. Determine which risks are important to deal with (prioritise risks).
3. Implement strategies to deal with those risks.

Risk management is used in the software development process if it is necessary⁵. Risk management process is iterative process, consisting of three basic activities:

1. Planing of the risk management^{6, 5}, when potential threats are identified, risk assessment method chosen, responsibility of risk assessment and monitoring assigned, frequency of reassessing risks defined etc.

The identification of potential threats is an activity, which requires expertise and experience. It could be said, that this is the state of the art.

2. Risk analysis, when the probability and impact of each threat is assessed, risks are prioritized according the results of the assessment and preventative, detective and corrective actions planned⁶ (in⁵ this activity is part of planning).

The most complex part of the risk analysis is identification of preventative, detective and corrective actions. It could be also said, that this is the state of the art requiring knowledge of management and experience.

3. Risk mitigation and monitoring, when developers and managers follow the activities for risk mitigation and responsible managers monitor continuously a situation with the potential threat^{6, 5}.

There are a lot of risk management methods available, based on standards, but concentrating more on some risk management aspects. For instance, RiskIt method, based

on CMM, concentrating on clear and structured definition of risk⁷, or method concentrating on maximum involvement of customer in planing of risk management process⁸.

It is possible to choose any convenient method, but identification of threat and activities to prevent, detect or correct the risk (or mitigate) will require expertise and experience anyway.

The survey was organized to find out risks, which are actual to software developers in Latvia, and activities for risk mitigation.

METHOD USED FOR IDENTIFICATION OF RISKS

Expert polling method was used to find risks, which are important to software developers in Latvia. The "Delphi" method^{9, 10} was chosen. "Delphi" is an iterative decision making method. The method has three steps:

1. Forming the group of respondents – the expert group in the terms of the method.
2. Forming of the questionnaire and spreading among the experts.
3. Analysis of the results.

The second and the third steps are performed iteratively, while group of experts has agreed on the topic.

4.1 Group of Experts

Group of experts should consist of 10 to 20 experts having the same level of expertise¹⁰. Thirteen experts were chosen. Experts where chosen from software development companies (87% of respondents) and from software development units serving other business units within the same company (13% of respondents). Each expert suited the following requirements to ensure the same level of expertise for all respondents within the group of experts.

1. Expert is currently working as software development project manager.
2. Expert has the position of project manager for at least 2 and no more than 8 years.
3. Expert has managed at least 2 software development projects.

4.2 Questionnaire

According to the method used for survey, the first step is to prepare the list of properties experts must agree on. Experts were asked to provide lists of risks having some impact on the software development process. After analysis of the lists of risks, the 2 major risks were highlighted (see Table 1).

Table 1. List of risks

No.	Risk
Customer risks	
1.	Lack of hardware on the customer side
2.	Difficult communication with customer
Requirements risks	
3.	Low quality of software requirements
4.	Unstable software requirements
Project management risks	
5.	Unrealistic schedules and budgets
6.	Weak project management
Developers risks	
7.	Software development environment bugs
8.	Lack of developers motivation
9.	Lack of hardware on developers side
10.	Change of qualified personal
11.	Lack of knowledge in software development technologies and environment
12.	Difficult communication among developers

Table 2. Risk relevance matrix

		Frequency											
		Rarely								Often			
Expert viewpoint		1	2	3	4	5	6	7	8	9	10	11	12
		Low impact	1	1	1	1	1	2	2	2	2	3	3
	2	1	1	1	1	2	2	2	2	3	3	3	3
	3	1	1	1	1	2	2	2	2	3	3	3	3
	4	1	1	1	1	2	2	2	2	3	3	3	3
	5	4	4	4	4	5	5	5	5	6	6	6	6
	6	4	4	4	4	5	5	5	5	6	6	6	6
	7	4	4	4	4	5	5	5	5	6	6	6	6
	8	4	4	4	4	5	5	5	5	6	6	6	6
	9	7	7	7	7	8	8	8	8	9	9	9	9
	10	7	7	7	7	8	8	8	8	9	9	9	9
	11	7	7	7	7	8	8	8	8	9	9	9	9
Heavy impact	12	7	7	7	7	8	8	8	8	9	9	9	9

The next step was to put all the risks from the list in order of decreasing frequency (see Table 4 in “8 Appendix”) and decreasing impact (see Table 5 in “8 Appendix”).

Risk frequency and impact given by each expert was consolidated using risk frequency and impact matrix (see Table 2). This matrix is prepared using qualitative risk assessment, according to the project of cabinet law¹¹. Consolidated frequency and impact forms the risk relevance. The final risk relevance is given in the Table 3.

Table 3. Risk relevance

Risks	Experts													Average relevance
	1	2	3	4	5	6	7	8	9	10	11	12	13	
Difficult communication among developers	2	1	8	2	5	1	5	8	1	5	5	4	5	4
Difficult communication with customer	8	9	9	9	9	9	9	3	1	2	6	9	5	7
Change of qualified personal	7	5	1	3	4	5	1	9	1	8	5	5	5	5
Lack of developers motivation	4	2	1	4	1	1	5	9	1	1	1	2	5	3
Lack of knowledge in software development technologies and environment	3	5	6	4	5	7	5	5	2	6	5	5	5	5
Lack of hardware on developers side	2	1	1	1	1	3	1	9	1	2	1	1	2	2
Software development environment bugs	4	4	5	6	5	2	1	9	7	5	5	5	5	5
Unrealistic schedules and budgets	9	6	9	8	9	9	9	6	6	9	9	8	9	8
Low quality of software requirements	6	8	9	7	9	5	9	4	8	9	8	9	8	8
Lack of hardware on the customer side	2	3	4	2	1	4	1	9	5	2	2	5	5	3
Unstable software requirements	6	9	5	9	9	9	9	4	8	6	9	9	8	8
Weak project management	7	7	2	5	2	5	5	1	5	9	5	5	5	5

Finally, the concordance is calculated on the risk relevance according to the "Delphi" method. Concordance rate is between 0 and 1. If the rate is close to 1, it means that experts agree about the topic. If the concordance is closer to 0, then the survey is corrected, updated, the reasons for disagreement among experts are discussed and survey is spread among experts once more. This survey was spread twice and finally has concordance rate 0.91. It means that experts have agreed on risks actual for software developers.

If comparing software development risks identified by our experts and those found in different sources, we can see that mostly risks are the same: unstable or low quality software requirements, unrealistic schedules and budgets, lack of knowledge in software development technologies and environment, change of qualified personal^{12, 13, 14, 5, 8}.

Risks, which are listed in the expert list, but not often found in sources about software development risks, are:

1. Lack of appropriate hardware on the customer side as well as on the developer's side.
2. Difficult communication with customer as well as among developers.

5. RISK MITIGATION ACTIVITIES

Experts were asked to provide activities for each risk mitigation.

5.1 Unstable software requirements

Unstable software requirements might be the king of the risks for software development as it is named in many sources (see above). Nevertheless some creep of requirements is normal during software development process. T. C. Jones comments, that monthly rate of change after the requirements are first identified runs from 1% to more than 3% per month during the subsequent design and coding stages is considered as normal¹³.

To prevent creeping requirements, the effective action is contracting of a sliding scale making the implementation of changes financial disadvantageous later in the software development life cycle¹³. Only one expert approved this as a preventative action.

The most popular preventative action is the establishment of project change request board consisting of customers and developers. All experts mentioned this in the questionnaire. Change request board conduct meetings on regular basis. Changes approved by project change request board are implemented only.

Using the iterative software development life cycle with some administratively limited time period of "freezing requirements" is another preventative activity popular among the experts. Dealing with unstable software requirements is the main advantage of the iterative software development life cycle introduced by B. Boehm^{15, 16}.

5.2 Unrealistic schedules and budgets

The common problem in the software industry is that of intense but artificial schedule pressure applied to the programmers by their managers and customers^{13, 17}. This is the significant risk mentioned by the experts as well.

There are four preventative actions proposed by the experts:

1. Use of formal methods for software development cost estimation before the development starts. There are a lot of formal software development cost and schedule estimation models available and the supporting tools. The most popular is COCOMO¹⁸. This as a preventative action is proposed by T. C. Jones¹³ as well.
2. Use the advantages of technology (automated tools for configuration management, project management etc.).
3. Plan the software architecture so that it is possible to use previously developed and tested components.
4. Review of the software development plans, whether all is correct.

The corrective actions proposed were:

1. Negotiate the schedule with customer or executives in order to set the priorities for deliverables or extend the schedule.
2. Increase the workload for experienced team members, which are able to generate original solutions. This gives the result rather quickly, but is not a solution for long term¹⁷.
3. Change inexperienced team members with the experienced ones working in the similar problem area. This is the alternative to adding the extra staff, giving no expected results¹⁹.

5.3 Difficult communication with customer

Preventative actions:

1. Regular meetings on the project management level. It would be better to conduct these meetings on the customer site. If it isn't possible to meet, customer has to be informed about project development by phone or via e-mail. All the experts agreed, that this is the most effective preventative action.
2. More than half of the experts assume that cause of the communication problems is customers' lack of knowledge about software development life cycle. In this case the only action must be taken is education of the customer.

The only corrective action provided was to change of the contact person on the customer side to the person having more procurement in the customer's company and knowing the business area.

5.4 Low quality of software requirements

Preventative actions:

1. Don't cut time for software requirements specification. This job must end with mutually agreed (signed) software requirements specification.
2. Build prototype of the system under development.

The only corrective action provided was to find out more about requirements informally. It could be done by finding informal requirements pioneers on the customer side as well as on the developers' side.

5.5 Other Risks and Mitigation Activities

Lack of hardware on the customer side. Lack of hardware on customer side is one of the risks, which is specific for software developers in Latvia. This risk must be considered carefully during planning, hardware specification must be provided to customer as early as possible and these aspects must be negotiated carefully. All the experts agreed, that this is the most effective preventative action.

If corrective action is necessary, there are two possibilities:

1. Buy or rent hardware specified. This is the most effective corrective action giving the results immediately.

2. Optimization of the software. Half of the experts agreed that this would help. Another half said that this would never help and this was rather risky way, because new bugs would be introduced during optimization.

Software development environment bugs. Preventative actions:

1. Don't use software development environments developers are not familiar with.
2. Don't use new environment versions entering the market before the benchmarking information is available.
3. Establish company wide benchmarking bulletin and motivate developers post there information about problems highlighted during the software development process.
4. Building and using unified components where it is possible.

There are two corrective actions recommended by experts:

1. Find roundabouts on Internet or contact vendors.
2. Change the software development environment and train the developers in using new environment.

Weak project management. The first activity coming in mind is change of the project manager. Experts are rather cautious about change, saying that it may give the expected result as well as aggravate the situation in project. It is the last thing should be done. The other corrective actions proposed are:

Provide the experienced assistant to the project manager covering the areas in the project management field, where project manager is not so successful.

Encourage and assist to project manager in deeper analysis of the situation in the project helping to find out the most painful areas in development process and concentrate on them.

Lack of developers motivation. All the experts agree, that material benefits are important, but it is not enough. It is important, that developers see the result of their job.

Lack of hardware on developers side. This is an issue of planning. The only way is to purchase, rent or use customers' equipment.

Change of qualified personal. Preventative action proposed by experts is:

1. Assign responsibility about development of software component, module, function etc. to two developers always.
2. Document everything during software development process, even if customer doesn't request it.

These two as preventative actions of change of the personal is also recommended in ².

Lack of knowledge in software development technologies and environment. The preventative action is developers' training. All experts mentioned this. There were two groups within experts' group regarding corrective actions:

1. More than half of experts suggested involving of consultants – software developers, who are able to communicate with the rest of the group and assist during the development process.
2. Another group didn't advise involvement of external experts. They suggested finding a developer or group of developers within software development team, who are able to self-education.

There were three experts cautioning of developers, who claimed of being pioneers.

Difficult communication among developers. This risk has very high probability in the case, when development of some software components is outsourced to the third party. The preventative actions suggested by experts are as follows:

1. Regular meetings of the developers, weekly or twice a week, discussing problems during software development process.
2. Organise small development teams.
3. Define responsibilities.
4. Organize off-hour meetings, sports etc.

6. CONCLUSIONS

Software development practitioners have agreed, that software development risk management is important activity. Top twelve risks are identified and mitigation activities are highlighted. Software development managers could use the identified risks as a checklist for initial risk analysis. The preventative and corrective actions proposed by experts could be used as guidelines for planning software development risk mitigation activities.

There are two very important steps in software development risk management, which could be considered as a state of art:

1. Identification of risk.
2. Finding the appropriate preventative or corrective action.

Communication among software developers as well as communication between customer and developer is very important for successful software development. The further research is needed to provide effective methods for accumulation and appliance of software managers' experience in risk identification and mitigation.

7. REFERENCES

1. G.,P. Babcock, Editor. Webster's Third New International Dictionary: Unabridged (MA: Merriam-Webster, Springfield, 1981).
2. Information Systems Audit and Control Association (2002 CISA Review Manual, 2002)
3. Software Engineering Institute. "The SEI Approach to Managing Software Technical Risks." Bridge (October 1992), p.19-21
4. Carnegie Mellon Software Engineering Institute, Software Engineering Risk Management FAQ. (21st of March, 2001); <http://www.sei.cmu.edu/publications>.
5. C.Mark, B.Curtis, M.B.Chrissis, and C.V. Weber, *Capability Maturity Model for Software, Version 1.1*, (Software Engineering Institute, CMU/SEI-93-TR-24, February (1993)).

6. IEEE P1540/D11.0, "Draft Standard for Software Life Cycle Processes – Risk Management", (IEEE Standards Department, 2000).
7. R.Basili, J.Kontio, "Riskit: Increasing Confidence in Risk Management" (21st of April 2001); <http://satc.gsfc.nasa.gov/support>.
8. B.W. Boehm, "Software Risk Management: Principles and Practices" (IEEE Software, Jan 1991), pp. 32-41.
9. Л.В.Ницецкий, Л.П.Новицкий "Применение методов экспертного опроса для оценки качества диалоговых обучающих систем". Методы и средства кибернетики в управлении учебным процессом высшей школы. Сборник научных трудов, (Рига РПИ, 1986).
10. "Теория прогнозирования и принятия решений". Под ред. С.А. Саркисяна. (М.: Высшая школа, 1977).
11. Informācijas sistēmu riska analīzes metodika (23rd of February 2002); <http://www.lddk.lv>.
12. K.Lockyer, J.Gordon, *Project Management and Project Network Techniques* (Bell and Bain Ltd, 1996) pp. 49-51.
13. T. Capers Jones. *Estimating Software Costs*. McGraw-Hill, USA, 1998.
14. B.Hetzel, *Making Software Measurement Work*. John Wiley & Sons, Inc., 1993, 290 p.
15. B. Boehm "A Spiral Model of Software Development and Enhancement" (IEEE Computer, vol.21, #5, May 1988), pp 61-72.
16. G. Holt, "Software Risk Management – the Practical Approach" (Mei Technology Corporation, 2000, #2)
17. E.Yourdon, *DEATH MARCH. The complete Software Developer's Guide to Surviving 'Mission Impossible' Projects* (Prentice Hall, 1997) 218 p.
18. C.Abts, B.Boehm, B.Clark, S.Devnani-Chulani. *COCOMO II Model Definition Manual*_(University of Southern California) 68 p.
19. R. S. Pressmann, *Software engineering, a practitioner's approach* (McGraw-Hill, 1992).

8. APPENDIX

Table 4. Risks ordered by frequency (12 - the most frequently, 1 - the least frequently)

Risk	Expert													Average
	1	2	3	4	5	6	7	8	9	10	11	12	13	
Difficult communication among developers	8	3	5	6	5	1	7	8	3	6	5	4	5	5
Difficult communication with customer	6	12	10	10	10	9	10	11	2	6	9	10	8	9
Change of qualified personal	4	5	4	9	4	6	4	12	1	8	6	6	6	6
Lack of developers motivation	3	6	3	2	3	2	5	12	3	4	4	5	5	4
Lack of knowledge in software development technologies and environment	9	8	9	4	7	3	8	8	5	9	8	7	7	7
Lack of hardware on developers side	5	1	1	1	1	11	1	12	3	5	4	4	6	4
Software development environment bugs	1	4	7	11	8	5	3	12	1	7	6	7	6	6
Unrealistic schedules and budgets	10	11	11	8	9	12	9	11	9	10	10	7	9	10
Low quality of software requirements	12	7	12	3	12	7	12	3	6	9	8	11	7	8
Lack of hardware on the customer side	7	10	2	7	2	4	2	12	7	5	6	6	6	6
Unstable software requirements	11	9	8	12	11	10	11	3	6	9	9	9	8	9
Weak project management	2	2	6	5	6	8	6	3	6	10	5	5	7	5

Table 5. Risks ordered by impact (12 - heavy impact, 1 - very little impact)

Risks	Expert													Average
	1	2	3	4	5	6	7	8	9	10	11	12	13	
Difficult communication among developers	1	2	9	4	8	2	7	10	3	5	5	6	6	5
Difficult communication with customer	10	12	12	9	10	12	10	3	1	4	8	10	8	8
Change of qualified personal	9	6	3	3	7	5	4	12	3	10	6	6	6	6
Lack of developers motivation	5	3	1	6	1	3	5	12	3	1	4	3	5	4
Lack of knowledge in software development technologies and environment	4	5	7	8	6	9	8	8	3	7	7	7	7	7
Lack of hardware on developers side	3	4	2	1	2	4	1	12	3	3	4	4	4	4
Software development environment bugs	7	8	5	7	5	1	3	12	11	6	7	7	6	7
Unrealistic schedules and budgets	12	7	11	10	9	10	9	8	5	12	9	9	10	9
Low quality of software requirements	8	11	10	12	12	7	12	8	10	9	10	11	10	10
Lack of hardware on the customer side	2	1	8	2	3	8	2	11	5	2	4	5	5	4
Unstable software requirements	6	10	6	11	11	11	11	8	10	8	9	10	9	9
Weak project management	11	9	4	5	4	6	6	3	7	11	7	6	6	7

MEASUREMENTS AND RISKS BASED METHOD TO SUPPORT SOFTWARE DEVELOPMENT PROCESS PLANNING

Baiba Apine

*Riga Information Technology Institute, Kuldigas iela 45, LV-1083 Riga, Latvia
Baiba.Apine@dati.lv*

Abstract This paper describes the way in which software development process measurement data together with results of risk analysis are used for software development project planning. The Measurements and Risks Based Method (MERIME) to support software development process planning is proposed. The first results of using this method are announced.

Keywords: software development process planning, software development process measurements, risk analysis.

1. Introduction

Software development is rather complex process, consisting of different activities. It is often said that software development is very hard to manage and projects are out of schedule and out of budget. Planning of the development process is very important activity. The software development process is dependent on skills level of different specialists as well as on usage of different technologies. A lot of various factors must be kept in mind while planning the software development process. Therefore any information submitted to support full-cycle software development process planning as well as short-term planning is helpful.

There are two software development supporting processes providing useful information for planning: risk analysis and measuring. Several methods are used in software development companies for planning using the information produced by these two processes. For instance, analytical

software development cost estimation methods like COCOMO II [1], Experience Pro [4], which use statistical data about many software development projects probably developed in different companies. The results are used to forecast effort and schedule for software development projects. It is recommended to adjust the results given by formal analytical methods using software development process measurement information collected within the company [4], as formal results are might be too general and might not meet the specific project needs.

We have six-year experience of using COCOMO II method for software development effort and schedule planning. As well as the Measurement program is implemented as a set of activities on the company level to measure software development process. Gathered measurement data are used to adjust results of formal methods to our software development projects for more than two years. We often face the following drawbacks:

1. Analytical software development process effort and schedule estimation methods available are used to estimate the whole software development process only. It would be very useful for project short-time planning to forecast, for instance, the distribution of effort or any other project characteristic.
2. It is quite easy to gather the measurement data, but it worth nothing if not appropriately used. The effective usage of software development measurement data is a key success factor of Measurement program [3, 6]. It has to be shown continuously that measurement data are analysed properly and used efficiently.

To find the way how to use software development process measurement data for software project short-term planning, we started to use them together with results of risk analysis. This paper describes Measurements and Risks Based Method (MERIME) developed to support software development process planning. The method formalizes the experience of software managers and developers in such a way it could be spread among other software developers and used for software development planning.

Initially lets have a look at the method as a "black box" (see *Figure 1*) and discuss results expected from the usage of MERIME method for planning and input data necessary to apply the method.

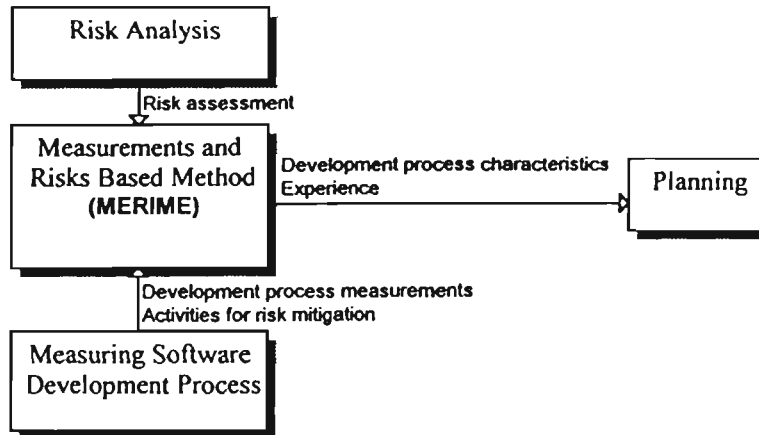


Figure 1. Measurements and Risks Based Method relationships with other processes

2. Output of the Method

There are two results from using MERIME method:

- Development process characteristics for planning: the percentage of effort had to be spent on developments as well as on different activities within software development process for next months. For instance, the result could be the following forecast: work amount spent on project documentation for the project with unstable software requirements will be 6% of total project development work amount for next month.
- Tips based on experience from other projects of software development risk mitigation activities. For instance, consider that the live risk for the project under development is instability of software requirements. The tip could be: "Establish project level control structure: change request board consisting of two developers and two customers".

These results could be used for project short-term planning. Nevertheless it must be kept in mind that these are forecasted characteristics and recommendations only, so these results must be treated carefully.

3. Input of the Method

There are two software development support processes creating input data for MERIME method: software development risk analysis and software development process measurement.

There are no special restrictions how to organize the software development process risk analysis. It is possible to use any software development risk analysis method, for instance, standards IEEE P1540/D11.0

[7] or CMMI [10] for software development process risk analysis. Nevertheless the results of risk analysis must satisfy the following criteria:

1. Results of the risk analysis must be summarized regularly. For instance, monthly.
2. Each identified risk must be graded. For instance, using grades from 1 (the lowest mark) to 5 (the highest mark) according to grading scale given in *Table 1*, which is used in our company.

Table 1. Risk grading scale

Risk Grade	Criteria for Choosing Risk Grade
1	Risk is not applicable to this project.
2	The probability that the risk will have negative influence to the development process is very low. It is not necessary to plan any activities for risk mitigation.
3	The probability that the risk will have negative influence to the development process is quite low. It is necessary to think about special activities for risk mitigation.
4	The probability that the risk will have negative influence to the development process is high. It is necessary to plan special activities for risk mitigation and monitor the risk continuously.
5	The risk will have negative influence to the development process. It is necessary to plan special activities for risk mitigation, monitor the risk continuously and involve top management in risk mitigation activities.

There are risk analysis methodologies grading separately risk probability and impact [8, 9] by assigning grades from 1 to 3. Total risk exposure is calculated by multiplying probability and impact. Although it is allowed to choose other risk grading scales, all the development processes, which are using MERIME method for planning, must be graded using one and the same risk grading scale.

MERIME method uses risk sequences as input data. Lets have a look at the sample. Risk analysis results suitable as input information for MERIME method are given in the *Table 2*, where risk analysis grades are chosen according to the criteria given in the *Table 1*.

Another software development supporting process producing information suitable for planning is measuring. Even if the official Measurement program doesn't exist at a company level, there are some traditions in each software development company regarding the software development process measuring. And again, like in the case with risk analysis data, MERIME method doesn't set any limitations on measurement information gathered in the company, the only restriction is that all the processes using MERIME method for planning must be measured according one and the same measurement methodology. This could be ensured by implementing Measurement program at a company level.

Table 2. Sample of risk analysis results (risk sequences)

Risk	Jan-00	Feb-00	Mar-00	Apr-00	May-00
Unstable software requirements	2	3	2	3	4
Schedule shortage	1	1	2	4	4

Lets have a look at the Measurement program sample, which results are suitable for using in MERIME method.

Software development process measurement activities have to be planned along with software development project planning. Each project gathers the following measurement data:

1. The amount of functionality has to be implemented (function points, lines of code etc.).
2. Work amount spent on different activities within software development process.

Each project developer writes down work hours spent on software development at the end of each working day.

3. Planned activities for risk mitigation.

During risk analysis, if the grade for the particular risk exceeds the accepted risk gap, risk mitigation activities are planned. For instance, if the grade for risk "Difficult communication with customer" increases 3, possible activity is "Conduct regular meetings on the project management level. It would be better to conduct these meetings on the customer site. If it isn't possible to meet, customer has to be informed about project development by phone or via e-mail." This is recorded by the project team member whose responsibility is project level risk management.

The head of Measurement program performs initial analysis of the measurement data by forming reports suitable for input data of MERIME method.

It is highly recommended that measurement data about problem reports are gathered and analysed also, but this is out of scope of this paper.

4. How does It Work ?

The idea of the method is to store risk analysis and measurement information (hereinafter risks and measurements) in MERIME information base and use it for new projects' planning (hereinafter current projects). MERIME method searches for relationships between risks and measurements in MERIME information base to find software development projects, which are similar to the current project, and therefore their measurement information could be used as guidelines for current project.

To illustrate how risks and measurements are stored in the MERIME information base, let's have a look at its meta-model [2] (see *Figure 2*):

1. *Project* is software development project, which risk analysis and measurement information is stored in the MERIME information base. *Project* attributes are name, development environment, problem area, etc.
2. *Risk* is, analysed and graded risk. "Unstable software requirements" and "Lack of knowledge in problem area" are samples of *Risk*. Name is an attribute of the *Risk*.
3. *Measurement* is project development process measurement information. "Percentage of work amount spent on documentation", "Average response time to problem report (days)" are samples of *Measurement*. Name is an attribute of the *Measurement*.
4. *Activity* is activity for risk mitigation. "Establish change request board", "Close project" are samples of *Activity*. Name is an attribute of the *Activity*.
5. *Sequence* is a sequence of measurement information or risk grades. For sample see *Table 2*. Name of the risk, month, for which risk is graded, grade and length of the sequence are attributes of the *Sequence*.

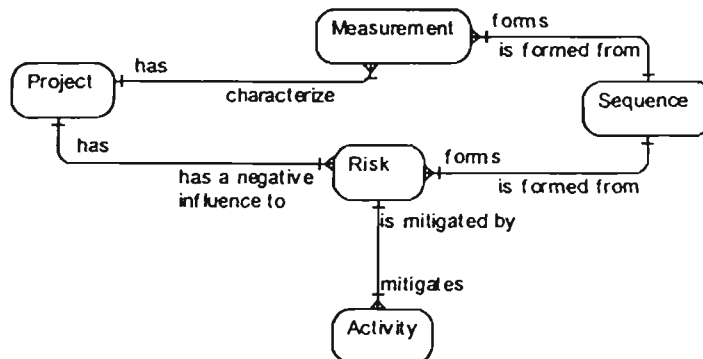


Figure 2. Meta-model of the MERIME method information base

Process diagram in *Figure 3* illustrates steps of using MERIME method for project development planning.

The first task is **Store risk analysis results**. To use MERIME method for new project planning (for current project), the following activities must be carried out:

1. Perform current project risk analysis and store results in MERIME information base for at least three months following the guidelines given in chapter "3 Input".
2. Perform current project risk analysis for the next time period. For instance, if MERIME method is used for planning for next two months, risk analysis must be performed for next two months also.

The results of the task **Store risk analysis results** are stored in the MERIME information base.

Task Store measurement information shows, that measurement data for new project must be gathered, summarised and stored in the MERIME information base for at least three months. The result of **Store measurement information** is stored in the MERIME information base.

Storing risk analysis and measurements information for current projects guarantees, that MERIME information base is updated continuously.

The next task is **Search for similar projects**, where all the projects satisfying the following criteria are found in MERIME information base:

1. Approximately the same amount of functionality must be implemented. The amount of functionality could be measured in function points, lines of code etc.
2. Software development projects follow the same software development life-cycle.
3. Have at least one risk sequence similar to the risk sequence of the current project.

In MERIME method two risks' sequences are considered as similar, if they are of equal length and there are no corresponding risk grades difference more than one. For instance, risk sequence $V1=1,2,3,3,2$ and $V2=1,3,3,2,1$, are similar. Sequence $V3=3,2,3,3,2$ is similar to neither $V1$ nor $V2$, because the first element of $V3$ differs from the first element of $V1$ and $V2$ for two.

The result of the **Search for similar projects** is *Risk and measurement sequence for sample project* satisfying the criteria defined above. The sample projects are considered as similar to the current project and will be used to generate guidelines for current project planning.

Example in *Figure 4* illustrates how to apply the criteria to find the sample projects. PROJ_4 is current project, but PROJ_1, PROJ_2 and PROJ_3 are sample projects found in the MERIME information base. The PROJ_4 satisfies the following requirements:

1. The same amount of functionality must be implemented.
2. Software development projects follow the same software development life-cycle.
3. PROJ_4 risks' sequence of length 5 is similar to PROJ_1, PROJ_2 and PROJ_3 appropriate risks' sequences.

The next task is **Process characteristics forecast**. Process characteristics are forecasted to keep the measurement sequences for the current project the same as for sample projects. *Process characteristics* are the result of the usage of the MERIME method.

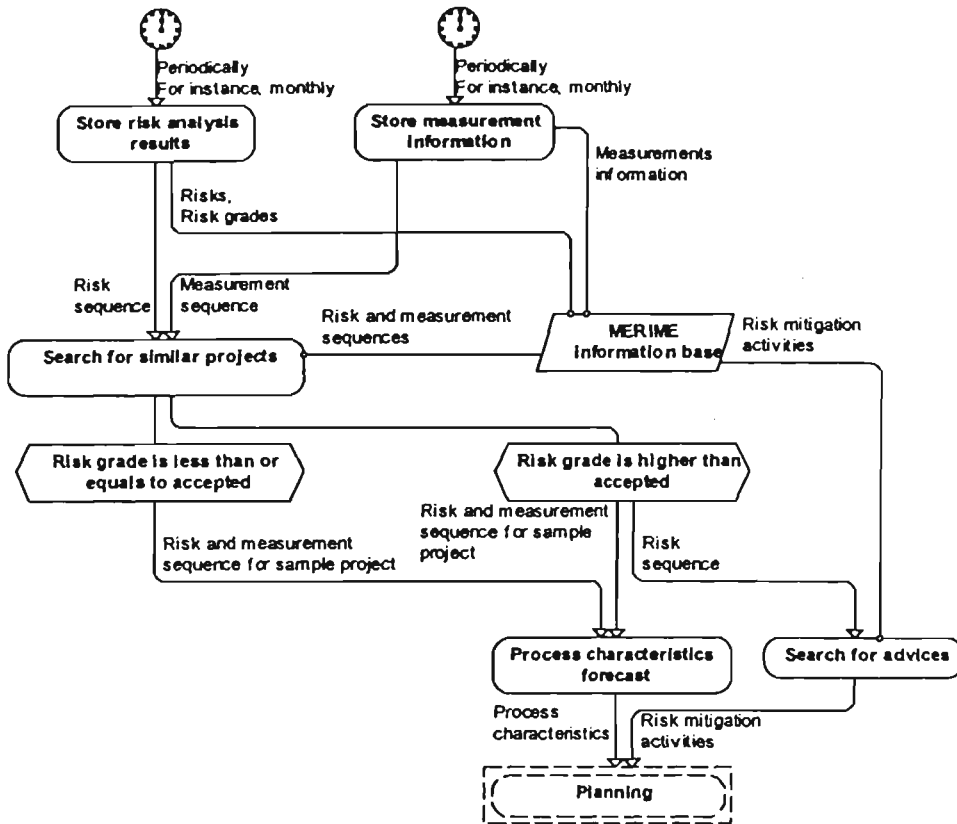


Figure 3. Business process of MERIME method. Business process is described using GRAPES/BM business modelling language [5].

Have a look at the sample in the *Figure 4* once more. We shall forecast the work amount spent on documentation for the next month for PROJ_4. The sample projects used are PROJ_1, PROJ_2 and PROJ_3. The work amount for documentation for the next time period for PROJ_4 will be 7.8% from work amount spent on project development (see *Figure 4*). This value is chosen as the closest for keeping measurement sequence for the PROJ_4 within the range of measurement sequences of PROJ_1, PROJ_2 and PROJ_3.

The last task is **Search for advices**. This task is executed only in the case when the risk grade in the sequence is higher than accepted. *Risk sequence* of the current project is used for searching in the MERIME information base. All projects having at least one risk sequence similar to the risk sequence of the current project and having the last risk grade in the sequence quite high are considered as sample projects. For instance, if the scale for risk grading given in *Table 1* is used, high-risk grades are 4 or 5. The result is the list of *Risk mitigation activities* stored in the MERIME information base.

5. Application of the MERIME Method

Although MERIME method is still under development it is already used for planning for three software development projects within one software development company for half a year. The MERIME method information base contains risk analysis and measurement information about seven projects.

This software development company focuses on development of large tailored information systems for customers in Latvia and Western Europe, where short-term planning is very important and might be rather complex activity.

There are already established risk analysis traditions in the company, but risk analysis procedure is not formally defined yet. All the project level risks (unstable requirements, lack of knowledge in the problem area, squeezed development schedule etc.) are discussed weekly on the project level meetings.

There are some measurement traditions established in the company. The following measurement information is captured on the regular basis:

1. All development teams capture information about defects, usually including identifier, description, function and version where defect fixed, defect fix date, severity, current defect status, status fix date etc.
2. Some development teams capture information about time spent on different activities during project development.

There is accepted Software development measurement program in the company concentrating on usage of the existing data captured in different software development projects. These measurement data are gathered and analysed monthly. Standard set of measurement data reports is generated monthly and provided to the software developers. The measurement program is discussed in [3] in more details.

The first step was to fill data into the MERIME information base. Seven software development projects from approximately 40 currently running projects were chosen having the following advantages:

1. Project development teams have captured information about time spent on different activities during software development process.
2. Project manager loyalty to usage of formal methods to support software development planning.

Amount of functionality						
PROJ_1	1221	Function points				
PROJ_2	802	Function points				
PROJ_3	964	Function points				
PROJ_4	1008	Function points				
Month	1	2	3	4	5	6
Life cycle activity						
PROJ_1	Design	Design	Implement.	Implement.	Implement.	Testing
PROJ_2	Design	Implement.	Implement.	Implement.	Implement.	Testing
PROJ_3	Design	Design	Implement.	Implement.	Testing	Testing
PROJ_4	Design	Design	Implement.	Implement.	Implement.	
Risk: unstable software requirements						
PROJ_1	3	2	3	3	3	2
PROJ_2	2	3	2	2	2	3
PROJ_3	2	2	3	4	3	3
PROJ_4	2	2	2	3	2	
Risk: difficult communication with customer						
PROJ_1	2	2	2	1	1	1
PROJ_2	2	1	2	2	2	3
PROJ_3	2	2	1	1	1	1
PROJ_4	1	2	2	1	2	
Risk: lack of knowledge about development environment and technologies used						
PROJ_1	4	4	4	4	4	3
PROJ_2	3	2	2	2	2	3
PROJ_3	4	3	2	2	2	2
PROJ_4	4	4	3	3	3	
Measurement: percentage of work-amount spent on documentation						
PROJ_1	37.8%	20.0%	17.3%	13.3%	3.0%	6.3%
PROJ_2	4.5%	4.3%	5.4%	2.9%	4.8%	1.0%
PROJ_3	20.0%	17.3%	13.3%	10.9%	6.3%	3.5%
PROJ_4	12.2%	11.0%	11.8%	7.8%		

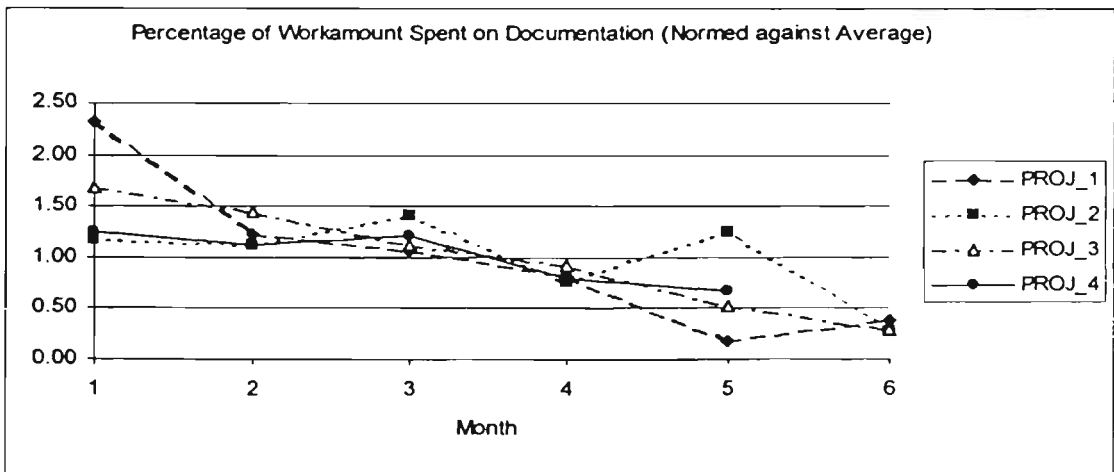


Figure 4. Searching for sample project in the MERIME information base

Questionnaires were provided to these five project managers asking to restore the risk sequences for their projects' risks following the scale given in *Table 1*. One month was chosen as a frequency period for the grading of risks, because measurement information is summarised monthly also. The results of the questionnaires were filled into the MERIME information base.

Measurement information for the initial filling into the MERIME information base was taken from the Measurement program reports. The following measurements information about each project was stored in the MERIME information base:

1. Percentage of work amount spent on development of the concept of operation, requirements specification, design, implementation, testing, deployment, maintenance, documentation, quality assurance, management, and training.
2. Average response time to problem reports generated by customers.
3. Activities for mitigation of risks having grade higher than 3.

The next step was to test the method for the projects under development. Three projects were chosen having the same advantages as those seven projects already stored in MERIME information base. Questionnaires were provided to project managers asking to restore the risk sequences for their projects' level risks following the scale given in *Table 1* from the beginning of the project. The results of the questionnaires for the first 4 months of projects' development were filled into the MERIME information base. MERIME method was used to get the measurement values forecasts for the next two months regarding the work amount spent on different activities during software development. The results were compared with real measurement values of the projects.

6. Conclusions and Further Work

The following conclusions can be made:

1. Method gives rather precise forecasts for the values of development process supporting activities: documentation, quality assurance, management and training. The average error of forecasts is 19.89% with deviation 15.02%.
2. Usage of this method shows that measurement information is used continuously and effectively, which is key success factor of the Measurement program successful implementation.
3. It is possible to use incomplete measurement and risk analysis data. It is very important that the first benefits from gathering measurement and risk analysis data could be obtained after first three months.

4. It is possible to use this method even if there are no data available about large number of software development projects. This is very important for software development companies in small countries like Latvia. Another reason why it is not possible to use large number of software development projects as a source for forecasting is rapid changes in technologies. Even in large companies and countries with high IT development level statistical data gathered becomes obsolete because of rapid changes in software development technologies and skills level of staff.
5. The method is very flexible as it determines neither risk analysis nor measuring process. It is possible to use this method in different software development companies with different software development process traditions.

Acknowledgement

This paper is partly supported by the Latvian Science Council programme No. 02.0002.

References

- [1] Abts, C., Boehm, B., Clark, B., Devnani-Chulani, S. COCOMO II Model Definition Manual, University of Southern California, 68 p.
- [2] Apine, B., Apinis, I., Krasts, O., Sukovskis, U. Meta-model Based and Component Based Approach for Information Systems Design, Proceedings of the 4th IEEE International Baltic Workshop: Baltic DB&IS '2000, 2000, Vilnius, Lithuania, pp. 78-83.
- [3] Apine, B., Smilts, U., Sukovskis, U. Software Measurement Practice to Address Customer Satisfaction, Scientific Proceedings of Riga Technical University, Applied Computer Systems. – 1st thematic issue, 2000, pp. 11–18.
- [4] Experience Pro, <http://www.sttf.fi>, 03.02.2002
- [5] GRADE Business Modeling. Language reference. INFOLOGISTIK GmbH., 1998
- [6] Hetzel, B. Making Software Measurement Work. New York: John Wiley & Sons, Inc., 1993.
- [7] IEEE P1540/D11.0, Draft Standard for Software Life Cycle Processes-Risk Management, IEEE Standards Department, 2000.
- [8] Informācijas sistēmu riska analīzes metodika, <http://www.lddk.lv>, 03.02.2002.
- [9] Microsoft, material No: 1516ACP, Principles of Application Development
- [10] Paulk, M.C., Curtis B., Chrissis, M.B., Weber, C.V. Capability Maturity Model for Software, Version 1.1, Software Engineering Institute, CMU/SEI-93-TR-24, February 1993.

MEASUREMENTS AND RISKS BASED METHOD TO SUPPORT SOFTWARE DEVELOPMENT PROCESS PLANNING

Baiba Apine

Riga Information Technology Institute, Kuldigas iela 45, LV-1083 Riga, Latvia

Baiba.Apine@dati.lv

Abstract: This paper describes the way in which software development process measurement data together with results of risk analysis are used for software development project planning. The Measurements and Risks Based Method (MERIME) to support software development process planning is proposed. The first results of using this method are announced.

Keywords: Software development process planning, software development process measurements, risk analysis.

1. Acknowledgement

This paper is partly supported by the Latvian Science Council programme No. 02.0002.

2. Introduction

Software development is rather complex process, consisting of different activities. It is often said that software development is very hard to manage and projects are out of schedule and out of budget. Planning of the development process is very important activity. The software development process is dependent on skills level of different specialists as well as on usage of different technologies. A lot of various factors must be kept in mind while planning the software development process. Therefore any information

submitted to support full-cycle software development process planning as well as short-term planning is helpful.

There are two software development supporting processes providing useful information for planning: risk analysis and measuring. Several methods are used in software development companies for planning using the information produced by these two processes. For instance, analytical software development cost estimation methods like COCOMO II [1], Experience Pro [2], which use statistical data about many software development projects probably developed in different companies. The results are used to forecast effort and schedule for software development projects. It is recommended to adjust the results given by formal analytical methods using software development process measurement information collected within the company [2], as formal results are might be too general and might not meet the specific project needs.

We have six-year experience of using COCOMO II method for software development effort and schedule planning. As well as the Measurement program is implemented as a set of activities on the company level to measure software development process. Gathered measurement data are used to adjust results of formal methods to our software development projects for more than two years. We often face the following drawbacks:

1. Analytical software development process effort and schedule estimation methods available are used to estimate the whole software development process only. It would be very useful for project short-time planning to forecast, for instance, the distribution of effort or any other project characteristic.
2. It is quite easy to gather the measurement data, but it worth nothing if not appropriately used. The effective usage of software development measurement data is a key success factor of Measurement program [3], [4]. It has to be shown continuously that measurement data are analysed properly and used efficiently.

To find the way how to use software development process measurement data for software project short-term planning, we started to use them together with results of risk analysis. This paper describes Measurements and Risks Based Method (MERIME) developed to support software development process planning. The method formalizes the experience of software managers and developers in such a way it could be spread among other software developers and used for software development planning.

Initially lets have a look at the method as a "black box" (see *Figure 1*) and discuss results expected from the usage of MERIME method for planning and input data necessary to apply the method.

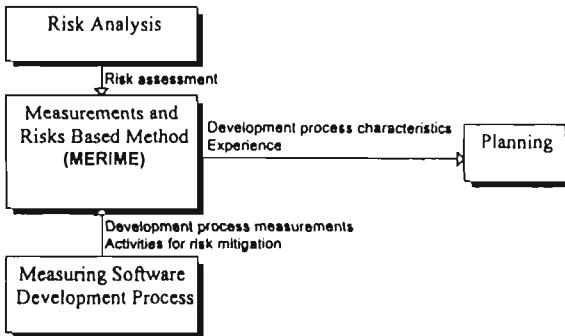


Figure 1. Measurements and Risks Based Method relationships with other processes

3. Output of the Method

There are two results from using MERIME method:

- Development process characteristics for planning: the percentage of effort had to be spent on developments as well as on different activities within software development process for next months. For instance, the result could be the following forecast: work amount spent on project documentation for the project with unstable software requirements will be 6% of total project development work amount for next month.
- Tips based on experience from other projects of software development risk mitigation activities. For instance, consider that the live risk for the project under development is instability of software requirements. The tip could be: "Establish project level control structure: change request board consisting of two developers and two customers".

These results could be used for project short-term planning. Nevertheless it must be kept in mind that these are forecasted characteristics and recommendations only, so these results must be treated carefully.

4. Input of the Method

There are two software development support processes creating input data for MERIME method: software development risk analysis and software development process measurement.

There are no special restrictions how to organize the software development process risk analysis. It is possible to use any software development risk analysis method, for instance, standards IEEE P1540/D11.0 [5] or CMMI [6] for software development process risk analysis. Nevertheless the results of risk analysis must satisfy the following criteria:

1. Results of the risk analysis must be summarized regularly. For instance, monthly.
2. Each identified risk must be graded. For instance, using grades from 1 (the lowest mark) to 5 (the highest mark) according to grading scale given in *Table 1*, which is used in our company.

Table 1. Risk grading scale

Risk Grade	Criteria for Choosing Risk Grade
1	Risk is not applicable to this project.
2	The probability that the risk will have negative influence to the development process is very low. It is not necessary to plan any activities for risk mitigation.
3	The probability that the risk will have negative influence to the development process is quite low. It is necessary to think about special activities for risk mitigation.
4	The probability that the risk will have negative influence to the development process is high. It is necessary to plan special activities for risk mitigation and monitor the risk continuously.
5	The risk will have negative influence to the development process. It is necessary to plan special activities for risk mitigation, monitor the risk continuously and involve top management in risk mitigation activities.

There are risk analysis methodologies grading separately risk probability and impact [7], [8] by assigning grades from 1 to 3. Total risk exposure is calculated by multiplying probability and impact. Although it is allowed to choose other risk grading scales, all the development processes, which are using MERIME method for planning, must be graded using one and the same risk grading scale.

MERIME method uses risk sequences as input data. Lets have a look at the sample. Risk analysis results suitable as input information for MERIME method are given in the *Table 2*, where risk analysis grades are chosen according to the criteria given in the *Table 1*.

Another software development supporting process producing information suitable for planning is measuring. Even if the official Measurement program doesn't exist at a company level, there are some traditions in each software development company regarding the software development process measuring. And again, like in the case with risk analysis data, MERIME method doesn't set any limitations on measurement information gathered in the company, the only restriction is that all the processes using MERIME method for planning must be measured according one and the same

measurement methodology. This could be ensured by implementing Measurement program at a company level.

Table 2. Sample of risk analysis results (risk sequences)

Risk	Jan-00	Feb-00	Mar-00	Apr-00	May-00
Unstable software requirements	2	3	2	3	4
Schedule shortage	1	1	2	4	4

Lets have a look at the Measurement program sample, which results are suitable for using in MERIME method.

Software development process measurement activities have to be planned along with software development project planning. Each project gathers the following measurement data:

1. The amount of functionality has to be implemented (function points, lines of code etc.).
2. Work amount spent on different activities within software development process.

Each project developer writes down work hours spent on software development at the end of each working day.

3. Planned activities for risk mitigation.

During risk analysis, if the grade for the particular risk exceeds the accepted risk gap, risk mitigation activities are planned. For instance, if the grade for risk "Difficult communication with customer" increases 3, possible activity is "Conduct regular meetings on the project management level. It would be better to conduct these meetings on the customer site. If it isn't possible to meet, customer has to be informed about project development by phone or via e-mail." This is recorded by the project team member whose responsibility is project level risk management.

The head of Measurement program performs initial analysis of the measurement data by forming reports suitable for input data of MERIME method.

It is highly recommended that measurement data about problem reports are gathered and analysed also, but this is out of scope of this paper.

5. How does It Work ?

The idea of the method is to store risk analysis and measurement information (hereinafter risks and measurements) in MERIME information base and use it for new projects' planning (hereinafter current projects). MERIME method searches for relationships between risks and measurements in MERIME information base to find software development projects, which

are similar to the current project, and therefore their measurement information could be used as guidelines for current project.

To illustrate how risks and measurements are stored in the MERIME information base, let's have a look at its meta-model [9] (see Figure 2):

1. *Project* is software development project, which risk analysis and measurement information is stored in the MERIME information base. *Project* attributes are name, development environment, problem area, etc.
2. *Risk* is, analysed and graded risk. "Unstable software requirements" and "Lack of knowledge in problem area" are samples of *Risk*. Name is an attribute of the *Risk*.
3. *Measurement* is project development process measurement information. "Percentage of work amount spent on documentation", "Average response time to problem report (days)" are samples of *Measurement*. Name is an attribute of the *Measurement*.
4. *Activity* is activity for risk mitigation. "Establish change request board", "Close project" are samples of *Activity*. Name is an attribute of the *Activity*.
5. *Sequence* is a sequence of measurement information or risk grades. For sample see Table 2. Name of the risk, month, for which risk is graded, grade and length of the sequence are attributes of the *Sequence*.

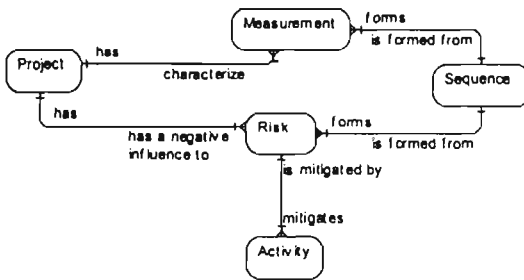


Figure 2. Meta-model of the MERIME method information base

Process diagram in Figure 3 illustrates steps of using MERIME method for project development planning.

The first task is **Store risk analysis results**. To use MERIME method for new project planning (for current project), the following activities must be carried out:

1. Perform current project risk analysis and store results in MERIME information base for at least three months following the guidelines given in chapter "4 INPUT".
2. Perform current project risk analysis for the next time period. For instance, if MERIME method is used for planning for next two months, risk analysis must be performed for next two months also.

The results of the task **Store risk analysis results** are stored in the MERIME information base.

Task **Store measurement information** shows, that measurement data for new project must be gathered, summarised and stored in the MERIME information base for at least three months. The result of **Store measurement information** is stored in the MERIME information base.

Storing risk analysis and measurements information for current projects guarantees, that MERIME information base is updated continuously.

The next task is **Search for similar projects**, where all the projects satisfying the following criteria are found in MERIME information base:

1. Approximately the same amount of functionality must be implemented.
The amount of functionality could be measured in function points, lines of code etc.
2. Software development projects follow the same software development life-cycle.
3. Have at least one risk sequence similar to the risk sequence of the current project.

In MERIME method two risks' sequences are considered as similar, if they are of equal length and there are no corresponding risk grades difference more than one. For instance, risk sequence $V1=1,2,3,3,2$ and $V2=1,3,3,2,1$, are similar. Sequence $V3=3,2,3,3,2$ is similar to neither $V1$ nor $V2$, because the first element of $V3$ differs from the first element of $V1$ and $V2$ for two.

The result of the **Search for similar projects** is *Risk and measurement sequence for sample project* satisfying the criteria defined above. The sample projects are considered as similar to the current project and will be used to generate guidelines for current project planning.

Example in *Figure 4* illustrates how to apply the criteria to find the sample projects. PROJ_4 is current project, but PROJ_1, PROJ_2 and PROJ_3 are sample projects found in the MERIME information base. The PROJ_4 satisfies the following requirements:

1. The same amount of functionality must be implemented.
2. Software development projects follow the same software development life-cycle.
3. PROJ_4 risks' sequence of length 5 is similar to PROJ_1, PROJ_2 and PROJ_3 appropriate risks' sequences.

The next task is **Process characteristics forecast**. Process characteristics are forecasted to keep the measurement sequences for the current project the same as for sample projects. *Process characteristics* are the result of the usage of the MERIME method.

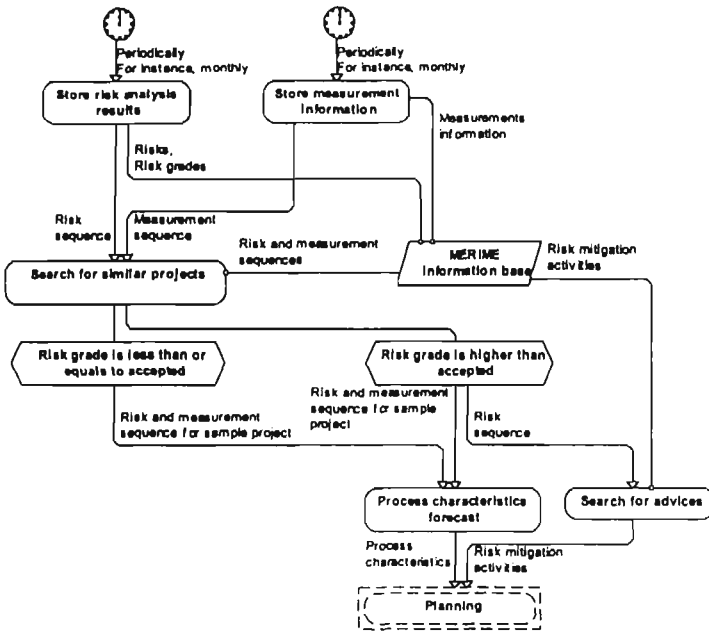


Figure 3. Business process of MERIME method. Business process is described using GRAPES/BM business modelling language [10].

Have a look at the sample in the *Figure 4* once more. We shall forecast the work amount spent on documentation for the next month for PROJ_4. The sample projects used are PROJ_1, PROJ_2 and PROJ_3. The work amount for documentation for the next time period for PROJ_4 will be 7.8% from work amount spent on project development (see *Figure 4*). This value is chosen as the closest for keeping measurement sequence for the PROJ_4 within the range of measurement sequences of PROJ_1, PROJ_2 and PROJ_3.

The last task is **Search for advices**. This task is executed only in the case when the risk grade in the sequence is higher than accepted. *Risk sequence* of the current project is used for searching in the MERIME information base. All projects having at least one risk sequence similar to the risk sequence of the current project and having the last risk grade in the sequence quite high are considered as sample projects. For instance, if the scale for risk grading given in *Table 1* is used, high-risk grades are 4 or 5. The result is the list of *Risk mitigation activities* stored in the MERIME information base.

6. Application of the MERIME method

Although MERIME method is still under development it is already used for planning for three software development projects within one software development company for half a year. The MERIME method information base contains risk analysis and measurement information about seven projects.

This software development company focuses on development of large tailored information systems for customers in Latvia and Western Europe, where short-term planning is very important and might be rather complex activity.

There are already established risk analysis traditions in the company, but risk analysis procedure is not formally defined yet. All the project level risks (unstable requirements, lack of knowledge in the problem area, squeezed development schedule etc.) are discussed weekly on the project level meetings.

There are some measurement traditions established in the company. The following measurement information is captured on the regular basis:

1. All development teams capture information about defects, usually including identifier, description, function and version where defect fixed, defect fix date, severity, current defect status, status fix date etc.
2. Some development teams capture information about time spent on different activities during project development.

There is accepted Software development measurement program in the company concentrating on usage of the existing data captured in different software development projects. These measurement data are gathered and analysed monthly. Standard set of measurement data reports is generated monthly and provided to the software developers. The measurement program is discussed in [4] in more details.

The first step was to fill data into the MERIME information base. Seven software development projects from approximately 40 currently running projects were chosen having the following advantages:

1. Project development teams have captured information about time spent on different activities during software development process.
2. Project manager loyalty to usage of formal methods to support software development planning.

Amount of functionality						
PROJ 1	1221	Function points				
PROJ 2	802	Function points				
PROJ 3	964	Function points				
PROJ 4	1008	Function points				
Month	1	2	3	4	5	6
Life cycle activity						
PROJ 1	Design	Design	Implement.	Implement.	Implement.	Testing
PROJ 2	Design	Implement.	Implement.	Implement.	Implement.	Testing
PROJ 3	Design	Design	Implement.	Implement.	Testing	Testing
PROJ 4	Design	Design	Implement.	Implement.	Implement.	
Risk: unstable software requirements						
PROJ 1	3	2	3	3	3	2
PROJ 2	2	3	2	2	2	3
PROJ 3	2	2	3	4	3	3
PROJ 4	2	2	2	3	2	
Risk: difficult communication with customer						
PROJ 1	2	2	2	1	1	1
PROJ 2	2	1	2	2	2	3
PROJ 3	2	2	1	1	1	1
PROJ 4	1	2	2	1	2	
Risk: lack of knowledge about development environment and technologies used						
PROJ 1	4	4	4	4	4	3
PROJ 2	3	2	2	2	2	3
PROJ 3	4	3	2	2	2	2
PROJ 4	4	4	3	3	3	
Measurement: percentage of work-amount spent on documentation						
PROJ 1	37.8%	20.0%	17.3%	13.3%	3.0%	6.3%
PROJ 2	4.5%	4.3%	5.4%	2.9%	4.8%	1.0%
PROJ 3	20.0%	17.3%	13.3%	10.9%	6.3%	3.5%
PROJ 4	12.2%	11.0%	11.8%	7.8%		

Percentage of Workamount Spent on Documentation (Normed against Average)

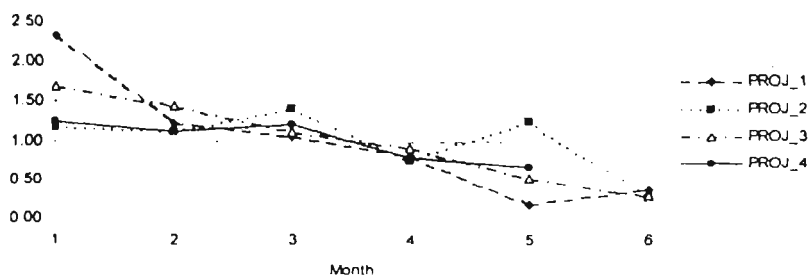


Figure 4. Searching for sample project in the MERIME information base

Questionnaires were provided to these five project managers asking to restore the risk sequences for their projects' risks following the scale given in *Table 1*. One month was chosen as a frequency period for the grading of risks, because measurement information is summarised monthly also. The results of the questionnaires were filled into the MERIME information base.

Measurement information for the initial filling into the MERIME information base was taken from the Measurement program reports. The following measurements information about each project was stored in the MERIME information base:

1. Percentage of work amount spent on development of the concept of operation, requirements specification, design, implementation, testing, deployment, maintenance, documentation, quality assurance, management, and training.
2. Average response time to problem reports generated by customers.
3. Activities for mitigation of risks having grade higher than 3.

The next step was to test the method for the projects under development. Three projects were chosen having the same advantages as those seven projects already stored in MERIME information base. Questionnaires were provided to project managers asking to restore the risk sequences for their projects' level risks following the scale given in *Table 1* from the beginning of the project. The results of the questionnaires for the first 4 months of projects' development were filled into the MERIME information base. MERIME method was used to get the measurement values forecasts for the next two months regarding the work amount spent on different activities during software development. The results were compared with real measurement values of the projects.

7. Conclusions and further work

The following conclusions can be made:

1. Method gives rather precise forecasts for the values of development process supporting activities: documentation, quality assurance, management and training. The average error of forecasts is 19.89% with deviation 15.02%.
2. Usage of this method shows that measurement information is used continuously and effectively, which is key success factor of the Measurement program successful implementation.
3. It is possible to use incomplete measurement and risk analysis data. It is very important that the first benefits from gathering measurement and risk analysis data could be obtained after first three months.

4. It is possible to use this method even if there are no data available about large number of software development projects. This is very important for software development companies in small countries like Latvia. Another reason why it is not possible to use large number of software development projects as a source for forecasting is rapid changes in technologies. Even in large companies and countries with high IT development level statistical data gathered becomes obsolete because of rapid changes in software development technologies and skills level of staff.
5. The method is very flexible as it determines neither risk analysis nor measuring process. It is possible to use this method in different software development companies with different software development process traditions.

References

- [1] C.Abts, B.Boehm, B.Clark, S.Devnani-Chulani. *COCOMO II Model Definition Manual*, University of Southern California, 68 p.
- [2] Experience Pro, <http://www.stf.fi>, 03.02.2002
- [3] Hetzel, B. 1993. *Making Software Measurement Work*. New York: John Wiley & Sons, Inc.
- [4] Baiba Apine, Uldis Smilts, Uldis Sukovskis. *Software Measurement Practice to Address Customer Satisfaction*, Scientific Proceedings of Riga Technical University, 2000, Applied Computer Systems. – 1st thematic issue, 11 – 18
- [5] IEEE P1540/D11.0, *Draft Standard for Software Life Cycle Processes-Risk Management*, IEEE Standards Department, 2000.
- [6] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, *Capability Maturity Model for Software, Version 1.1*, Software Engineering Institute, CMU/SEI-93-TR-24, February (1993).
- [7] Informācijas sistēmu riska analīzes metodika, <http://www.1ddk.lv>, 03.02.2002
- [8] Microsoft, material No: 1516ACP, *Principles of Application Development*
- [9] Baiba Apine, Ilgvars Apinis, Ojars Krasts, Uldis Sukovskis. *Meta-model Based and Component Based Approach for Information Systems Design*, Proceedings of the 4th IEEE International Baltic Workshop: Baltic DB&IS '2000, 2000, Vilnius, Lithuania, 78-83
- [10] 1998. *GRADE Business Modeling. Language reference*. INFOLOGISTIK GmbH.

PATTERNS AS A MEANS FOR MANAGING KNOWLEDGE IN THE INFORMATION SYSTEMS ENGINEERING PROCESS

Per Backlund {per.backlund@ida.his.se}
Ingi Jonasson {ingi.jonasson@ida.his.se}
University of Skövde
P.O. 408, S-541 28 Skövde, Sweden

Abstract: The traditional Information Engineering Process tends to be cumbersome and hard to manage. For example there are problems in managing knowledge and reusing past experiences. This is essentially a knowledge management problem. We propose the use of patterns as a means to resolve these problems. In order to be created, managed and retrieved, patterns must be organised in a repository. We have identified a number of problems that have to be dealt with, some of which are: problems concerning the strategies for the elicitation and evaluation of patterns as well as problems with regard to the documentation, storing, searching, and retrieval of patterns.

Keywords: Knowledge management, patterns, reuse, information systems engineering

1. Background

Information Systems Engineering (ISE) can be defined as an interdisciplinary approach to enable the realisation of successful information systems. The disciplines involved are integrated into a team effort with the goal of providing a quality product that meets both business and technical needs of all stakeholders. The ISE process is a highly knowledge intensive process. For example, developing an e-business application requires that a number of different competencies are coordinated in order to achieve success. Furthermore, the development time has to be shortened and the costs have to be reduced in order to stay in competition. Patterns, commonly described as core solutions to recurring problems, and software components have been

SOFTWARE DEVELOPMENT PROCESS IMPROVEMENT THROUGH MEASUREMENTS AND REQUIREMENTS TRACEABILITY

Baiba Apine, Martins Gills, Janis Plume

**Riga Information Technology Institute, Kuldigas iela 45, LV-1083 Riga, Latvia
{Baiba.Apine, Martins.Gills}@dati.lv*

***IT Alise, Brivibas iela 68, LV-1011 Riga, Latvia
J.Plume@alise.lv*

Abstract: Quality becomes more and more critical aspect of software development - most companies are looking for possibilities to improve quality and begin their improvement initiatives. Authors seek to develop a predictable environment for software development as groundwork for process improvement. The paper outlines a method that comprises quality and document system development, including the integration of measurement program and improved management of project information by means of a traceability tool. Lessons from real-life experience in two software development companies are analysed.

Keywords: Software process improvement, process measurements, quality assurance, traceability.

1. Introduction

Quality becomes more and more critical aspect of software development - most companies are looking for possibilities to improve quality and begin their improvement initiatives. Developers in the two software development companies in Latvia have identified the main problems related to project development. Problems are due to company and project level unawareness of the risks present. The process enhancement actions have to be taken. This paper outlines three aspects of software process improvement (SPI) for different levels:

1. Development of quality management system as a solution at company level (see Chapter 2).

2. Measuring software development process for organisational level as well as at project level (see Chapter 3).

3. Traceability issues and the tool Tracelt at project level (see Chapter 4).

The traceability concern within a project is guided, for example, by the necessity to know the impact of a particular change in a set of project items, or when there is a question of requirement coverage.

2. Software Process Improvement Initiatives at Company Level

Quality management systems are one of the most common ways for a process improvement initiative. It is an integral measure and involves almost all functions of the company.

Most process improvement paradigms impose continuous improvement. That means, after implementation of quality system, companies must go on with their improvement activities. In such phase SPI effort is more implemented in small steps, sometimes referred to as process improvement experiments. Here a short experience on how quality management system was implemented in one company is reported.

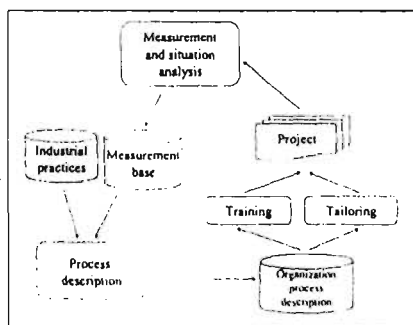


Figure 1. Continuous process improvement

2.1 General approach

Industry practice adoption and learning of the company from its experience are the most evident sources for process improvement. [11] One more widely used practice, as input generator for process improvement is measurement program, which is a particular kind of experience summary. Thus, following general system can be drawn as shown in the Fig. 1.

This system reflects continuous approach for process improvement, emphasised by most common process improvement paradigms (ISO 9001, TQM, EFQM).

There are a lot of problems arising from introduction of such kind of system. Most of them are of cultural nature and must be handled with a great precaution. These aspects are widely discussed in various sources including [11].

2.2 Process training and quality system usage

Development of processes is learning process by itself. Before the processes are put as formal text in a procedure, it is untimely to say that everything is clear in it. One of the most important benefits from process description activities is additional understanding of the process by developer of process documentation.

Other aspect of the process document usage is its role in training of new employees. A very popular way for a newcomer to start in company is to start reading of procedures, instructions, job descriptions and other documents, provided that these documents are reflection of the actual situation.

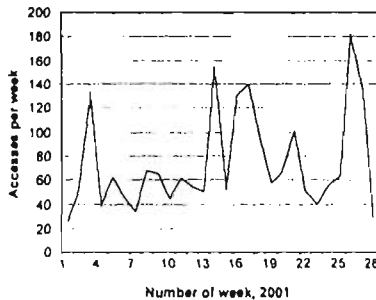


Figure 2. Example of quality document database access intensity

Document control system allows monitoring of intensity of the document usage. Analysis of such information may reveal interesting aspects of employee habits in using the system. Fig. 2 shows number of document access per week.

Number of document access shows peaks. Analysis of such peaks showed two reasons of additional activity of document usage:

- External motivation (e.g. external and internal audits)
- New employee recruitment

2.3 Document system development

Document types and common understanding of their meaning (procedures, instructions, guidelines, standards, templates, etc.) are preconditions for starting document system development.

Number of documents in the quality system differs, depending on practices adopted for the company, size of company and scope of quality system. For instance, a software company having about 100 employees owns following number of documents during ISO 9001 certification: 23 procedures, 21 instructions, 16 guidelines, 45 templates and 9 internal standards.

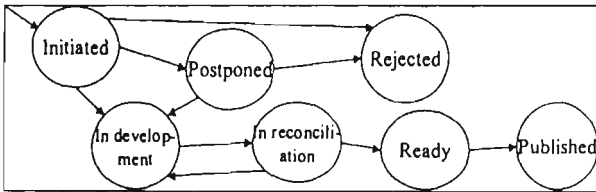


Figure 3. Document status transition diagram

As the total number of documents is high, a document control system can help quite significantly. Main functions for such system are the following:

- Document initiation and development start-up
- Document development (co-ordination and reconciliation)
- Document approval and document accessibility assurance
- Document version control, which means initiation of changes and control of implementation in the same way as the development of a new document.

These functions are subject to automated support by appropriate software tools. Significant concept of document development system is "Document status". Fig. 3 shows a possible status transition diagram of a quality document. Nevertheless experience shows that too strict implementation of the document life cycle may slow down the real process of document development.

3. Measurement Process Development

Measurement process is another important part of the quality management system at the company level as information from different software development projects are analysed. Measurement process can be considered at the project level also as measurement data are analysed to improve project monitoring. The experience of two software development companies is analysed here.

A lot of information is generated by software development process suitable for measuring [4], [5]. For instance, productivity of software developers, time spent for single requirement implementation, number of new defects per fix, etc. It must be kept in mind that measuring takes some project development resources as well as project management, configuration management etc. Therefore it is very essential not to choose very large variety of measurements but concentrate on the most relevant ones. Our goal is to use measurement data in order to increase software development process transparency and minimise software development risks.

As these goals are rather abstract, we have to find out what are the major software development process risks. We have questioned 20 software projects managers to range a list of risks in the order of decreasing frequency. The list of risks was taken from different sources [6], [10]. Managers were allowed to add their own risks to the proposed list as well. Those software project managers have experience of managing at least two software development projects and at least 3 years in the position of software development manager. More than a half of the respondents (56%) said that the software development projects had different problems related to software requirements: poor quality, unstable. This is the most frequent source of risks yielding cost and schedule overruns and other software development problems.

Our task was to provide methods and tools to monitor software project development process because the real situation awareness is the very first step to the mitigation of risks. Measuring software development process and using measurement data effectively is the key to software development process transparency. The mitigation process itself is creative and must be left to managers and developers responsibility while measurements provide information for project monitoring. Hence two issues are essential:

1. Information provided by measurement process has to be analysed and discussed regularly, let say monthly, in order to follow changes of development process characteristics continuously.

2. Some reference information from best practices is necessary to compare current process measurements.

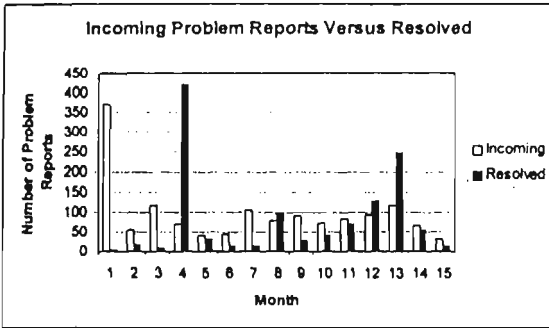


Figure 4. Sample chart illustrating defects' analysis on monthly basis

Monthly measurement reports for each software development project consist of:

1. Overall effort spent on project development: actual effort versus planned.

2. Time spent on different activities during software project development. This includes time spent on software design, implementation, testing etc. as well as on different supporting processes.

3. Classification of defects by types, by severity etc. This measurement is the most prevalent in projects.

4. Defect distribution throughout the software. Defects are grouped by source file, configuration unit, functional unit etc. It can help to find the most erroneous chunks of software as well as those, which aren't tested enough.

5. Problem report status changes over time. This measurement is very useful for early identification of problems in software development process. See Fig. 4 illustrating the project crisis starting at the first months and ending on the third month.

6. Defect resolve time (see Fig. 5). Defect response time data rises more questions than answers, for instance, is this response time profile what we actually want, or what are we going to do to change it.

Is it enough to gather all these measurements in order to improve project monitoring? The answer is NO, because measurement information must be properly analysed regularly, otherwise there is no sense to spend effort on capturing measurement data.

At the end of the project all the project measurement information is summarised and stored in the database of closed projects. The following information is stored:

1. Size of the functionality of the project expressed in the function points or lines of code. Function points are used for the new software development projects. Lines of code are used for software maintenance projects.

2. Effort distribution by different activities during software project development. This measurement is essential for ratio-based software development process estimation. The basic problem with ratio-based estimation is the false assumption that there are constant ratios between coding, testing, project management and other key activities [9]. This distribution differs from one project to another.

3. Staff assigned to project development.

4. Cumulative number of fixed and resolved defects over time. This characterises "health" of the project and whether software developed is used by end users or not.

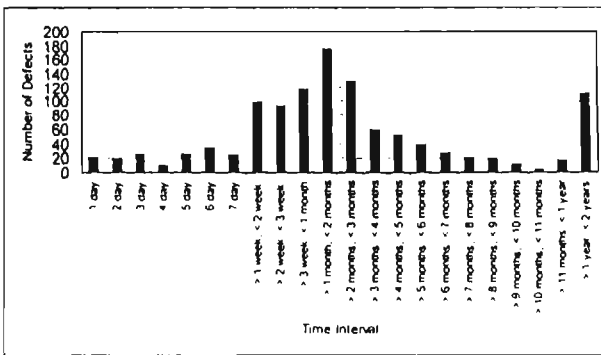


Figure 5 Sample chart illustrating defects resolve time

Software measurements proposed are enough to provide information for project monitoring, but in real life measurement team faces some problems while gathering real measurement data from development projects: different methodologies and different tools are used by development teams even in one company. For instance, the following measurement information is captured traditionally:

1. All development teams capture information about defects, usually including identifier, description, function and version where defect fixed, defect fix date, severity, current defect status, status fix date etc.
2. Some development teams capture information about time spent on different activities during project development.

There are various defect-tracking systems used to record information about defect reports. Such a situation is common for companies developing custom software, because:

1. Some software development teams have customer-defined defect tracking tools.
2. Some software development teams historically have developed their own tools for defect tracking using spreadsheets or simple database systems.

Different data sources are rather complex to analyse properly: data of different structure and from different sources must be summarised in case to create defect status reports. There are two alternatives to handle the situation:

1. Define unified format for defect tracking information and recommend using it in all software development projects.
2. Use the existing data captured in different formats.

The first solution seems to be rather complex and resource consuming. Switching to unified defect tracking format for all software development teams at different projects' development stages will be very time consuming. This solution doesn't fit to outsourced projects at all.

The second alternative left, and it was chosen. Using measurement data already available and let development teams continue to capture measurement information in the way they did for years. This way matches with organisational culture and is less staff resistant, but is more complex for measurement data analysis team.

4. Project Process Improvement Through Traceability

As software process improvement and measurement process discussed in previous chapters concentrate on the software development process as a whole at the company level, possibility of a more detailed analysis of project development process is very important as well. Within projects there always are some relations between the project items - between various documents, modules, tests and reported problems, but it is often problematic to answer to questions like "What are the dependencies? How some change in one requirement will impact the code or the tests?" The relations between items may not be indicated explicitly, but there are numerous questions produced about if and how any two project items are related.

Questions of project item relations mentioned above rise the project traceability concerns. There has been an extensive research made by O.C.Z.Gotel and A.C.W.Finkelstein [2] indicating requirements traceability issues that take place prior and after the requirements specification is created. Analysing the project information structure, one can conclude that requirements traceability is a subset of a more general traceability mechanism within the set of related project items.

Feature traceability: an ability to follow from one project item to another via relations with evolutionary (one is created as a result of another) or event character.

Formally, the existence of a traceability mechanism or process property within the software development life cycle is required by industry standards, like ISO/IEC 12207 [8]. Also the interpretation of the standard ISO 9001:2000 for software development TickIT Guide Issue 5.0 in a number of requirements includes the traceability issue [1]. Also, the standard J-STD-016 [7] requires traceability presence among produced deliverables.

4.1 Traceability model definition

There may be multiple ways of how several items may be related. First, there may be evolution-provoked relations like ones that can exist between requirements and tests. In most cases they may belong to class called strong relations, as they may be often tied with a strong dependence. Second, there may be event-provoked relations that do not propagate necessity for change from one item to another. These are called weak relations, and they may take place between test and test log. The project items could be the information of almost any sort, but the main criterion is its relation to project development or subject under development. Some most typical items are: requirement, design item, function, module, screen form, test, test case, test log, problem report, individual task, change request, e-mail of the customer, review report. Each item may have one or more several items that reference to it, and it may reference to one or more other items.

The identification of project item types and their possible relations defines a traceability model. A sample model is shown in Fig. 6, this model was identified in one real-life project. Arrows signify relation direction (e.g. forming the phrase "Test is based on business function") where solid line corresponds to a strong relation, but dotted - to a weak one.

4.2 Experience report - tool for traceability managing

Seeking the ways of how the traceability model could be implemented in the real-life projects, the requirements were defined for the traceability tool [3]. The very basic requirement for the traceability tool was the ability to support the universal approach in terms of the project type and the methodology used. This includes the possibility to define the item types, relation types and the traceability model. A tool named Tracelt was developed in one software company. It is web-based application, and effectively supports definition of traceability model, individual items and

supports multiple users. It is possible to work with item lists, work with relations, to see the relations for particular item in a form of a tree, to analyse the impact of changes.

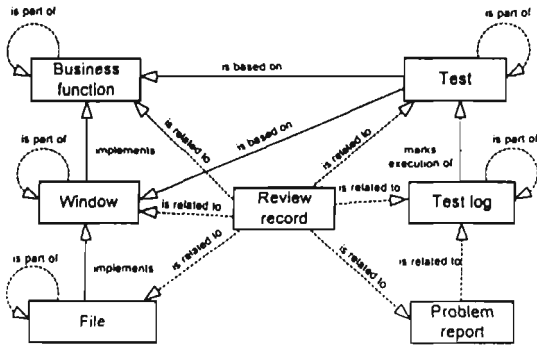


Figure 6. Sample traceability model

TraceIt was successfully used into 4 projects in pilot mode, and new software development projects are being considered for further research. The main gain for projects was that this tool did not introduce a completely new methodology. In this pilot usage, it also did not challenge developers to change the project culture, but it did improve some particular activities of the software development process and reduced the internal diversity of different media where exact item is placed. It was discovered to be difficult to place completely in the traceability database items like requirements or the code, but at the same time entirely all the testing information was maintained in TraceIt repository. Contents of items that did not contain the complete information consisted of a reference in a form of an ID, title and a basic classifying descriptor information only. The application reduced considerably the risk of ignoring important dependencies or deviating from the initial requirements in the further development process.

No detailed information workflow management was supported within the current version of TraceIt, and the workflow feature is evaluated to be useful especially in large project teams.

4.3 Requirement change tracking

Although the tool did not provide specific means for requirement engineering, it proved to be useful to control the transition phases from informal requirements to formal ones, and from the formal requirements to system design components. As it was mentioned earlier, there are almost no

projects that do not change their requirements over the development time. Due to TraceIT capability to indicate all item dependencies it proved to be extremely useful to evaluate the change impact and to reassign some relations from old requirements to new ones.

4.4 Problem resolution tracking

Process of problem reporting and handling took place in those projects where the tool was used extensively for testing documentation and defect tracking purposes. First, the complete set of test descriptions was placed into traceability database, marking the related requirements, functions, windows, etc. Second, all the results of test execution were documented with relation information kept. Third, all the problem reports were maintained in the database, allowing developers to find out the corresponding test or requirement. Also the information on problem resolution was exchanged through this tool.

5. Conclusions

As a result of the experience of SPI initiatives in two companies, the following conclusions are declared:

1. The confirmation was found that at the company level the management commitment is essential for any SPI initiative.

2. Quality management system may show no immediate improvement that is reflected by some measurements or financial indicators, but in the long run project transparency increases. This confirms the well-known SPI practices.

3. All practices described in literature and required by quality standards must work as a whole. Lack of good integration of, say, measurement program, with the process change management will make personnel not to understand the system as integral unit. If the components (even very good ones) will work as separate units - it will not provide sufficient effectiveness and efficiency.

4. Human factors have to be taken into account during the improvement - it is easy to invent a system, but it is hard to make it work. It is true both for SPI initiatives at the company level as well as for measurement process at the company and the project level.

5. Implementation of any kind of improvements needs to be evaluated - is the improvement really observable. The only way to answer to these questions is a measurement program.

6. Software measurements proposed in the chapter 3 cover the main issues to provide information for project status monitoring.

7. Traceability in projects cannot be ignored as it gives good results in improving project monitoring at the project level.

8. The traceability model helps to understand and improve the project information structure. Tool for traceability support can provide not only reference information, but additionally serve as project information base for certain activities. Introduction of a Tracelt tool does not break the current methodologies used in project development, it adapts to the existing ones.

Acknowledgement

This paper is partly supported by the Latvian Science Council programme no. 02.0002.

References

- [1] British Standards Institution. *The TickIT Guide Issue 5.0* (2001)
- [2] Orlena C.Z. Gotel Anthony C.W. Finkelstein. *An Analysis of the Requirements Traceability Problem*. 1994. <http://citeseer.nj.nec.com/78573.html>
- [3] Gills, Martins. *The Concept of a Universal Traceability Tool for Software Development Projects*. Scientific Proceedings of Riga Technical University. Series - Comp.Sc., Applied Computer Systems. 2nd issue (2001).
- [4] Grady, R.B.. *Practical Software metrics for project Management and Process Improvement*. New York: Prentice-Hall. (1992).
- [5] Hetzel, B. *Making Software Measurement Work*. New York: JW&S, Inc. (1993).
- [6] Hyatt L. E., Rosenberg L. H., *Software Metrics Program for Risk Assessment*, (2001). http://satc.gsfc.nasa.gov/support/IAC_OCT96
- [7] IEEE, J-STD-016-1995. *Standard for IT Software Life Cycle Processes*. (1995).
- [8] ISO. Std ISO/IEC 12207. *Standard for IT- SW Life Cycle Processes*, (1995).
- [9] Jones, T.C. *Estimating Software Costs*. London: McGraw-Hill. (1998).
- [10] Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, and Charles V. Weber, *Capability Maturity Model for Software, Version 1.1*, Software Engineering Institute, CMU/SEI-93-TR-24, February (1993).
- [11] Zahran, S. *Software Process Improvement - Practical Guidelines for Business Success*, Addison-Wesley, (1997).

Meta-model Based and Component Based Approach for Information Systems Design

Baiba Apine, Ilgvars Apinis, Ojars Krasts, Uldis Sukovskis

Riga Institute of Information Technology
Kuldigas 45, LV-1083 Riga, Latvia
uldis.sukovskis@dati.lv

Abstract

The paper introduces an object-oriented and component based approach for software design. This approach allows to design software in convenient and easy to understand manner using metamodel of real world system. The first step - building of the entity-relationship diagram, is discussed using project management support information system as an example. Universal repository as a storage of model objects is proposed to build using meta-meta-model based three-layer architecture. And the final step - implementation of the designed model using COM objects is described with illustrations from the same example of project manager's information system.

Keywords: software development, object-oriented, meta-model.

1. Meta-model Based Tool for Registering of Objects

One particular implementation of meta-model based universal repository will be used to explain an approach applicable to design of different types of information systems. PARK is a tool for registering various kinds of objects and their attributes into universal repository. We created the tool for registering information about software development projects. This includes information about project itself, staff involved, activities and documents. Documents could be generated using information in the repository and preliminary designed document templates. Tool accepts MS Word, MS Excel and MS Project documents. The PARK tool is considered as sample of usage of the repository only.

The main features of this tool are:

- a relatively small number of concepts used: only eight object classes and 14 types of relations between objects, with the terms defined intuitive and easily interpreted even by a novice;
- new types of objects and relationships could be added easily;
- information from the repository could be retrieved in various ways;
- project documentation could be generated using information from the repository.

The entity-relationship diagram shown in Figure 1 is the formal meta-model for PARK. All objects have their own attributes depending on object type. Besides individual attributes there are some applicable to all objects: for example, name, type and who and when created or modified this object in the repository.

Project is an object containing information about software development project. Usually there is one Project object in the repository only. Individual attributes for the Project object are: identifier, development environment, start date, end date etc.

Performer is an object containing information about a man involved in the project development process. This could be project manager, programmer, customer etc. Name, position, phone, fax and e-mail address are attributes of the Performer object.

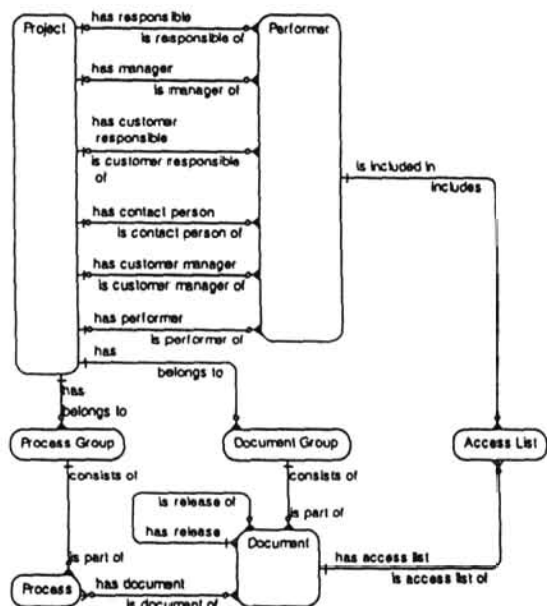


Figure 1. Object types and relationships

Process group is used for grouping of some processes. Process group samples are primary software life cycle processes, supporting processes etc.

Process is an object describing any activity within software development process. Start date, end date, description are attributes of the Process object. Samples of the Process object are coding, testing, auditing etc.

Document group is a head object for some group of documents. Document is an object describing a piece of project documentation. User manual, software requirements specification are samples of the Document object.

Access list is an object describing whether the particular Performer object has read, read/write or write access to the particular Document object.

PARK presents a model in the window which shows the objects and their relationships in a tree-like structure (see Figure 2). The way in which objects are displayed on the object tree is user defined. The branches of the tree are objects interconnected by relationships.

2. Repository of Models

The data describing a model are interrelated in a complex manner. Several versions of model need to be saved. Each model consists of various objects and relationships between them described by meta-model. Also a model itself may be considered as a complex object which refers to other models.

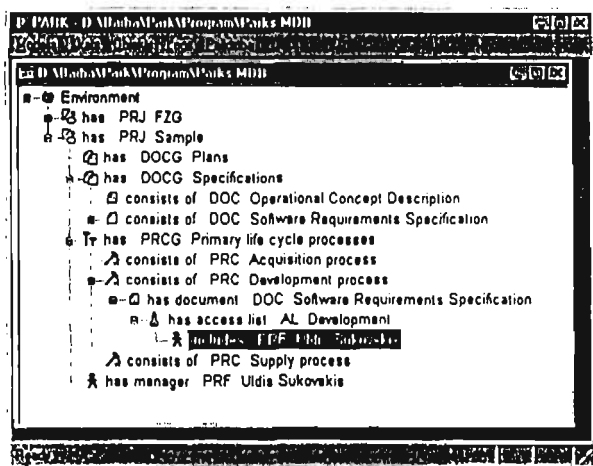


Figure 2. PARK window presenting information about software development project

Taking this into account we designed "universal" Repository. Structure of the Repository could be described with meta-meta-model (see Figure 3).

Meta_Object stores information what types of objects contain the meta-model. Long and short names of the object type are essential attributes for this object. For instance, objects of type named Performer and short name PRF (see Figure 2) could be stored in the Repository for the PARK meta-model.

Meta_Link stores information about relationships in the meta-model. Name of the relationship, two role names and cardinalities are the attributes of this object. For instance, PARK meta-model contains relationship consists_of / is_part_of with cardinalities 1..1 and 0..N (see Figure 2).

Meta_Object_Link stores information what objects are connected with what relationships.

Object stores specific information about objects of all types defined by meta-model (see Figure 2). This entity stores information about project as well as about performers. Values of attributes specific for every object type are coded as string. Role stores information about relationships between Objects.

There are three layers, related to design and usage of the Repository:

- 1) Physical layer: the lowest level of abstraction, at which it is described HOW the data are actually stored,
- 2) Conceptual layer: the next level of abstraction at which it is described WHAT data are actually stored in the data base and the relationships that exist among data. At this level it is possible to manipulate with generalized objects in a standard fashion, of course using methods which are supplied by a Physical layer.
- 3) Logical view layer: this is the highest level of abstraction at which only a part of the entire data base is described. This level is for particular tool. Many logical views may be provided for the same Conceptual level. The Logical view layer can't use functions provided by the Physical layer directly.

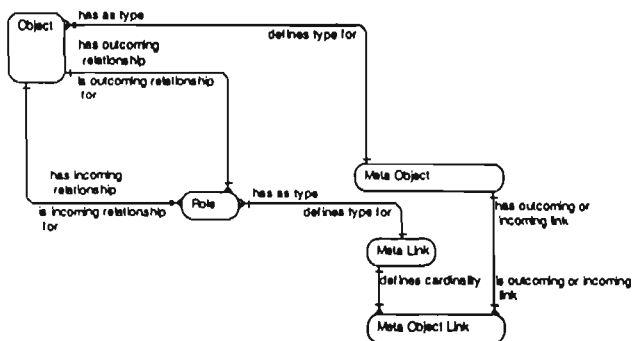


Figure 3. Meta-meta-model of the Repository

Such the layered architecture allows relatively easy changing of the underlying physical level database (for the case if it is necessary to change a platforms, or in the case of performance problems). If a new tool is created, then corresponding Logical view can be added. We use PARK meta-model as the base of such Logical view.

Taking into account the layered architecture of the Repository, we choose object - oriented approach in tool architecture. Object - oriented approach fits together with MS COM application architecture. Each object is implemented as COM object. There are attributes and functions common for all object types. Each object inherits those and has its own attributes and functions (see Figure 4). Objects expose their functions which are legal for the particular object type. These functions could be used by any software tool. We created one – Browser. Browser operates with objects on the Logical view layer. The main function of the Browser is to show the contents of the Repository according to the meta – model. Browser itself doesn't support any objects' function. It calls those implemented in corresponding object (see Figure 4).

This approach allows easy to change meta-model. For instance, add new object types. New Meta_Object and appropriate links to other object types must be registered in the Repository and corresponding COM object has to be implemented.

3. Functionality of Objects

Every object defined in the meta-model has functions applicable for this object. There is functionality corresponding to the conceptual layer and functionality corresponding to the logical view layer. Every object is implemented as COM object having its own set of functions. There are some functions equal for all objects defined in the meta-model. Those functions are defined in the conceptual layer. They don't depend on the type of object. Examples of functions defined in the conceptual layer are:

- functions Edit and View for entering or changing of the attributes of the object or viewing object attributes without ability to change them;
- functions Add object and Delete object for adding and deleting objects in the repository; function Add object allows add objects having some relationship with the particular object only;

- functions Add relationship and Delete relationship for adding and deleting relationship starting from the particular object;
- function Rename for global renaming of the object.

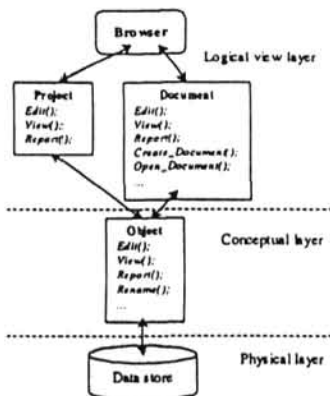


Figure 4. Object layers

Functions declared above have some implementation on the Conceptual layer. As all objects must be derived from an abstract Object defined on the Conceptual layer (see Figure 4), this set of functions is considered as default for objects defined in the meta-model and stored in the repository.

If the new object type is added to the meta-model, it by default has this set of functions with implementations on the Conceptual layer. If other implementation is necessary for some function, it is redefined on the Logical view layer.

Some functions are defined and implemented in the logical view layer. These are functions specific for document objects. For instance, function Create document for creating MS Word, MS Excel or MS Project document for the document object. Documents are generated on basis of the predefined document templates. Values from the repository are filled in the document.

3.1 Generation of Documents

Sample of implementation of the objects' functionality on the Logical view layer is generation of documents.

MS Word, MS Excel or MS Project documents could be generated using information from the repository. This is the essential function for the document object. Documents are generated on basis of predefined document templates. Function uses predefined document properties and user defined document properties. Function Generate documents fills values of the predefined and user defined properties from the information stored in the repository (see Figure 5).



Figure 5. Document sample with automatically filled properties

There are two steps to define document template:

- define properties to the document template;
- add fields of previously defined properties in the document text in appropriate places (see Figure 6). Standard properties of the document could be also used.

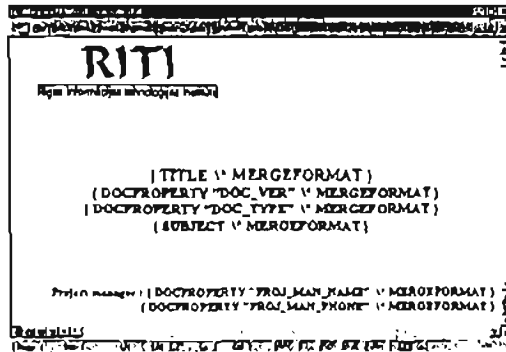


Figure 6. Document template

Software Measurement Practice to Address Customer Satisfaction

Baiba Apine, Uldis Smilts & Uldis Sukovskis
Riga Institute of Information Technology, Riga, Latvia

Keywords: Software measurement, Software development process improvement, Software quality

ABSTRACT: Monitoring of customer satisfaction is necessary to evaluate and validate whether the software under development meets requirements or not. Experience in implementation of measurement program in the software development company is analysed in this paper. The Goal Question Metric method is used as guidance for the measurement program, set of goals is determined and measurements to capture are defined. Case study describes implementation of measurement program in the real software development project.

1 INTRODUCTION

Customer satisfaction and loyalty are key drivers of profit. The saying "if you aren't measuring it, you aren't managing it" certainly applies to software development companies. True "customer satisfaction" is an organisation's ability to attract and retain customers and enhance the customer relationship over time. It is not simple and the answer cannot be collapsed into a single "customer satisfaction index". Addressing customer satisfaction is a key issue in the ISO 9004:2000 (ISO 2000).

Monitoring of customer satisfaction is necessary to evaluate and validate whether the product under development meets requirements or not. On the one hand, it is not possible to create single customer satisfaction index and monitor its change during whole software development process. On the other hand, we can not manage what we can not measure. The set of measurements is necessary to monitor whether the software product meets requirements provided by customer and whether it is on time and within budget.

2 GOAL QUESTION METRIC METHOD

The principle behind the Goal Question Metric (GQM) (Solingen et al. 1999) method is that measurement should be goal-oriented. GQM defines a certain goal, refines this goal into questions, and defines metrics that should provide the information to answer these questions. By answering the questions, the measured data defines the goals operationally, and can be analysed to identify whether or not the goals are attained. Thus, GQM defines metrics from a top-down perspective and analyses and interprets the measurement data bottom-up.

The GQM method contains three phases:

1. The Definition phase, during which goals, questions and measurements are defined.
2. The Data Collection phase, during which actual data collection takes place, resulting in collected data.
3. The Interpretation phase, during which collected data is processed to provide answers to the defined questions.

There are some sources indicated problems to apply GQM method (Hetzl 1993):

- The experience in many companies (especially the bigger ones), is that no one knows or there are unable to agree what the right set of goals should be. Some measurements are necessary to set the goals.
- Some goals and questions are fairly easy to set but extremely difficult to measure effectively. For instance, how to measure increasing or decreasing customer satisfaction?

Nevertheless having clear goals in mind it is easier to sell the measurement program to senior company management.

3 MEASUREMENT PROGRAM

Facts discussed in this paper are taken from measurement program's implementation experience in large software development company in Latvia. This company is ten years old and has approximately 500 employees working on 40 different software development projects at the same time. Main business of the company is development of the systems projects for state institutions as well as for customers from finance, telecommunication and transport.

Quality management of the company is oriented towards development, implementation and maintenance of information systems and software at the highest level. This means developing only high quality products, which meet client requirements, without deviating from contractually agreed technical requirements. The decision was made to start to implement software measurement program in the company as a part of the entire quality management system. The Goal Question Metric method (GQM) was chosen as guidance for the measurement program.

4 SETTING GOALS

A set of the most common measurement programs' goals (SPC 1999), (Grady 1992), (Solingen et al. 1999) were chosen and provided for discussion among software development team leaders and senior management. As a result of the discussion some of them, like improving productivity, improving software performance, minimise schedule etc., were scratched out from the list as not relevant. Priorities were assigned to the remaining measurement program's goals. Top five measurement program goals in the order of decreasing priority:

1. Improve software estimation: avoid development cost and schedule overruns, minimise software development risks, accurate projects' proposals
2. Improve project tracking
3. Improve software quality: meet product requirements, reduce delivered defects, reduce time spent on rework etc.
4. Minimise development cost
5. Increase customer satisfaction

From the list above we can see that increasing customer satisfaction has the lowest priority. Project managers in their everyday life deal with problems like creeping requirements resulting in slipping schedule and cost overrun, etc. Nevertheless respondents avoided of scratching off this goal of the whole list of the goals, probably, because of politeness. In fact, increasing customer satisfaction is derived from all goals having higher priority. For instance, improving accuracy of the software cost and schedule estimation increases customer satisfaction.

While thinking about the list of goals listed above, two questions rise:

1. Where do we are now on the way of achieving each of the goals? This question could be analysed further: Do we have software cost overruns and how often? How do we track project progress at the project management level and at the company's management level? What is the quality of the developed products? How satisfied are our customers?
2. What exactly do we want to achieve?

Seeking the answers to the questions above, a set of measurements is defined to provide information for analysis. Important is that measurements themselves do not answer any question, they provide information for analysis only. Software development specialists must carry out analysis.

There is a lot of information provided by software development process for measuring. For instance, productivity of developers, time spent by single requirement implementation, number of new defects per fix, etc. It must be kept in mind that measuring takes some project development resources as well as project management, quality management etc. So very important is to select the minimum set of measurements to answer the selected questions.

To answer the questions above the following measurements should be useful:

- Actual number of person hours/days/months to complete each activity during project development. This must include software requirements specification, implementation, testing and other software development activities as well as activities like internal and external project audits, project management, quality management etc.
- Date the particular activity started and ended.
- Defect type, severity, date defect discovered, item or function defect discovered, defect status, date the defect closed etc.

5 DEFINING SET OF MEASUREMENTS

There are some measurement traditions established traditionally. Every software development team captures some measurement data, but in many cases without clear goals in mind. The following measurement information is captured traditionally:

- All development teams capture information about defects, usually including identifier, description, function and version where defect fixed, defect fix date, severity, current defect status, status fix date etc.
- Some development teams capture information about time spent on different activities during project development.

There are various defect-tracking systems used to record information about defect reports. Such a situation is common for companies developing system software for particular customers, because:

- Some software development teams have customer-defined defect tracking tools. This is common for outsourced projects.
- Some software development teams historically have developed their own tools for defect tracking using spreadsheets or simple database systems (MS Access, for instance).

A lot of different data sources are rather complex for measurement program: data of different structure and from different sources must be summarised in case to create defect status reports. There are two alternatives to handle the situation:

1. Define unified format for defect tracking information and recommend using it in all software development projects.
2. Use the existing data captured in different formats.

The first solution seems to be rather complex: switching to unified defect tracking format for 40 software development teams at different projects' development stages will be very time consuming. This solution doesn't fit to outsourced projects at all.

The second alternative was chosen: to use measurement data already available and let development teams continue to capture measurement information in the way they did for years. Those teams which haven't customer restrictions on the defect tracking tool and haven't their own tools use configuration management tool PVCS Tracker for defect tracking. Choosing PVCS Tracker for defect tracking is recommended only, project manager is free to choose different tool.

6 DEFINING FEEDBACK

Having a good feedback mechanism in place ensures that the information obtained from the program is effectively communicated and used. Monthly measurement program reports for each project are generated and send to the project leads. These reports consist of:

- Overall effort spent on project development: actual effort versus planned.
- Time spent on different activities during software project development. This includes time spent on software design, implementation, testing etc. as well as on different supporting processes (see Figure 1).

This measurement is very useful for ratio-based software development process estimation in the company. The basic problem with ratio-based estimation is the false assumption that there are constant ratios between coding, testing, project management and other key activities (Jones 1998). Measurement about

time spent on different activities within software development process gives information how does ration varies depending on software type and development process risks in the particular company.

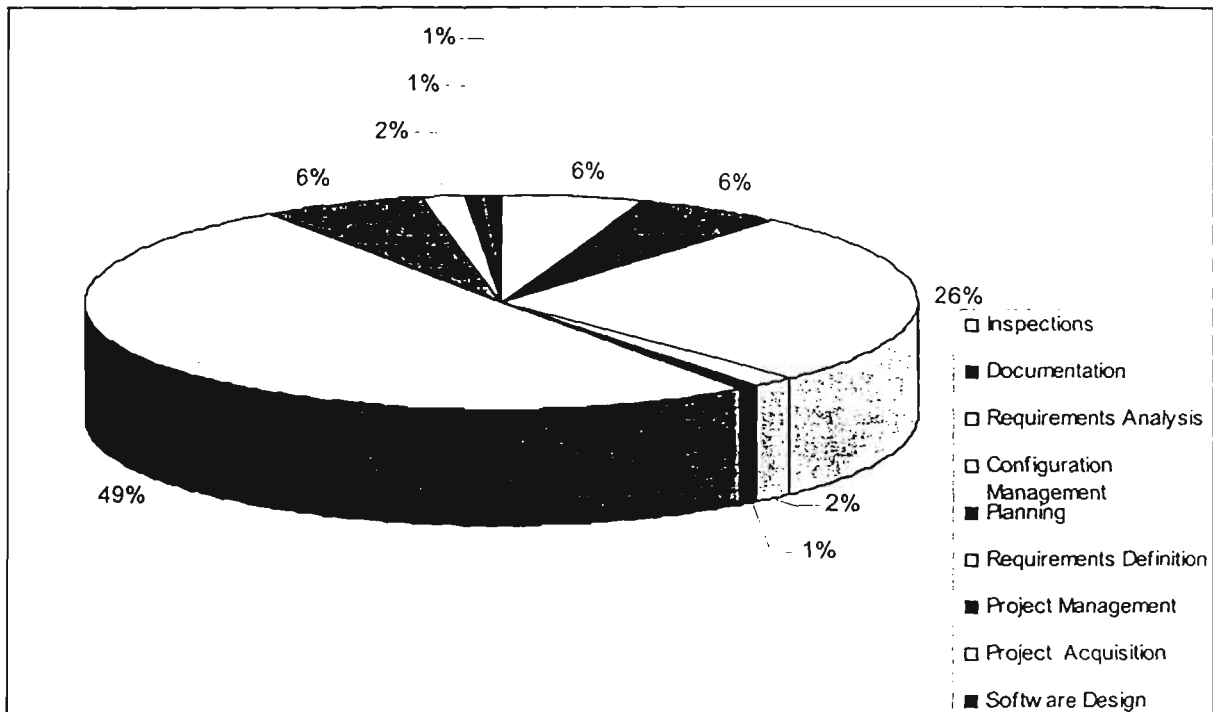


Figure 1. Sample monthly report about time spent on different activities within software development process.

- Tables containing classification of defects by types, by severity etc.
This measurement is the most prevalent in projects; because all projects use some kind of defect classification. Nevertheless this measurement shouldn't be used to compare different project development processes, because different projects use different classification of defects.
- Defect distributions throughout the software.
Defects are grouped by source file, configuration unit, functional unit etc. It can help to find the most erroneous chunks of software as well as those, which aren't tested enough.
- Problem report status changes over time.
This measurement is very useful for early identification of problems in software development process (see Figure 4 and the case study for project A).
- Defect response time (see Figure 2).
This metric is the most important as customer satisfaction measurement. At the same time these data rises more questions than answer, for instance, is this response time profile what we want, or what are we going to do to change it.
This set of measurements is summarised in the monthly report for every project team. The monthly reports consisting of 9 tables and 9 charts are sent to the leads of the projects. Analysis of the received measurement data reports at the project level is time consuming, therefore it is no sense to summarise measurement information weekly or biweekly: it shouldn't be analysed properly.

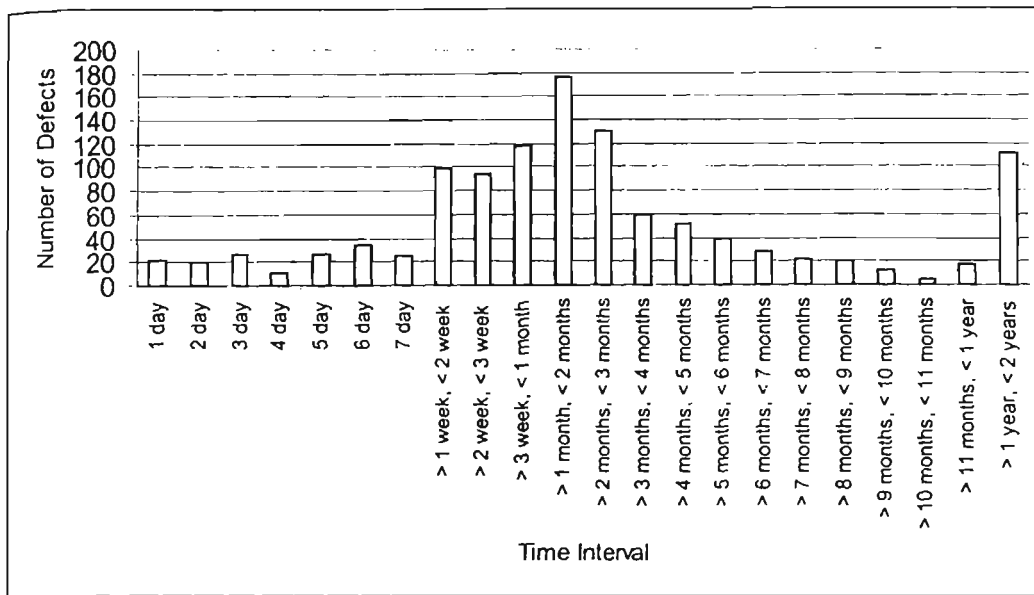


Figure 2. Sample chart illustrating defects response time.

7 CASE STUDY: MEASURING PROJECT ALPHA

This sample project describes the implementation of a measurement process on a real project. This case study shows how measuring software development process helped to get over the project crisis. Special consideration is given to use software measurement data that is readily available within the established software and project management processes.

Project ALPHA is State register information system; therefore software quality has extremely high importance. It is a project of typical scope and size for software development market of Latvia. This new information system replaces the old one developed in MS-DOS environment. The project is under development for several years. The total amount of effort involved in project development is about 200 person-years. Developers are involved in the implementation and maintenance of the developed IS, as well as in the user training. They are assisting in the administration and maintenance of the corporate network, which covers the whole territory of Latvia with offices located in across the country. Figure 3 shows the architecture for the information system under development.

Decreasing customer satisfaction has become a serious project development risk factor. Customer discovered a lot of defects after every product delivery, as usually. Customer laid the blame on the software development process on the developers side resulting in long response time complaints. The measuring was introduced with clear goal in mind: to increase customer satisfaction through increasing product quality and development process tracking.

Fortunately there was information available about defects detected during preliminary testing. Data about defect reports received from customer was also available. Data was captured for a year, the collection procedure is well established, but data weren't summarised and used for real decision support. In order to answer the following question: what are the real measurement values defining poor customer satisfaction, all historical data were analysed.

Figure 4 contains a sample bar chart illustrating number of incoming reports versus the resolved ones. This sample clearly illustrates crisis in the project ALPHA during the 1st and the 2nd months. In reality the situation was even worse: defect removal efficiency was 23%. This was far from good product quality: the approximate U.S. norm for defect removal efficiency is about 85% (Jones 1996).

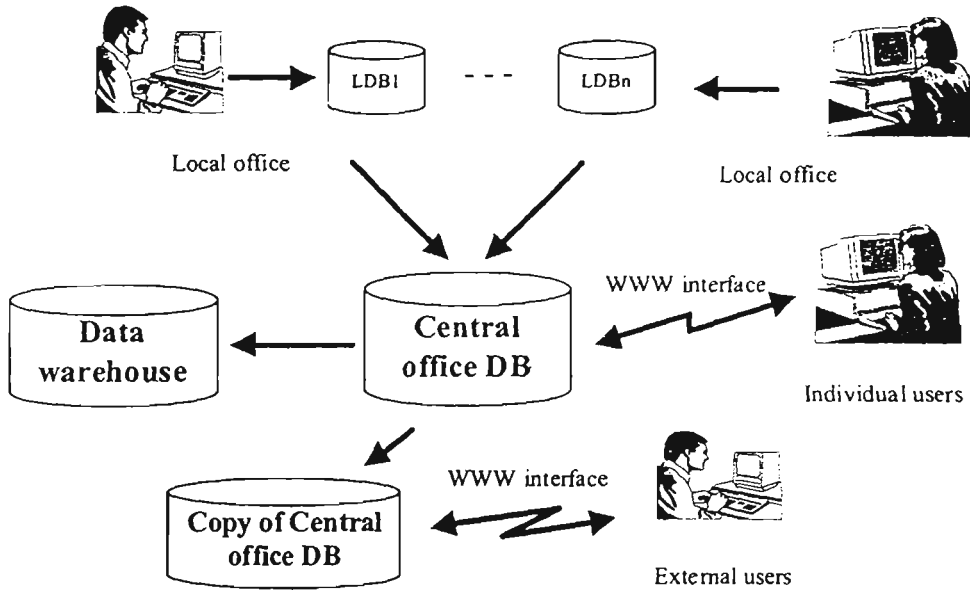


Figure 3. System architecture for project ALPHA.

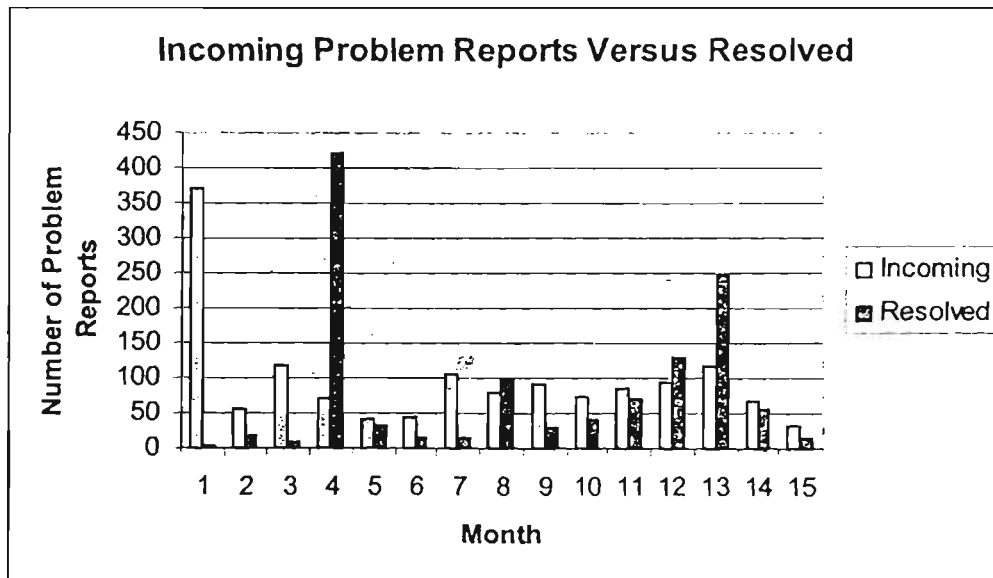


Figure 4. Sample monthly report for project ALPHA.

The reasons of the crisis were:

- A lot of new functionality was provided to customer with the newest version of the software. Software wasn't tested enough.
- Users had used system developed in MS-DOS environment. They weren't trained enough to use Windows applications: data entry on client side in local offices was slow, legacy data wasn't converted to new structure.

After measurement data analysis, the decision was made to concentrate on testing and resolving of the already received and incoming problem reports and customer training. The following actions were carried out to take over the development crisis and to eliminate future crisis:

- Problem reports were ordered by severity and priorities were assigned to each problem report. Defects were corrected in the order of decreasing priority. New software versions were delivered to customer quite often.
- System users were trained to use software as well as basic principles of working in Windows environment. Biweekly inspections increased customer and developer understanding in real system functionality.
- Data capturing was continued using the same templates; because changing the metrics definition during project development is time consuming and may result in receiving inconsistent and unreliable data while new collection procedure establishes. Since then monthly data analysis was performed using the captured data.

The first two activities together gave customer ability to follow project development process. Quantitative measures progress slowly, number of incoming problem reports exceeded the handled ones for next 3 months (see Figure 4), but customer has feeling that project development goes on.

All the previously discovered problems were resolved, defect removal efficiency for software versions delivered increased to 80% and users were trained to use the system.

Nevertheless discussions about software process improvement continued. Further analysis of the defect report data showed, that problems with development process are not on the developer side only. Figure 5 shows that delivery of new upgrade is held on the customer side for 2 months and more before accepted. This is because of complex acceptance procedure or lack of resources on the customer side.

Initially all measurement information was treated as confidential. As project development has become more stable, project leads started to use measurement data in negotiations with customer. Discussing any information concerning project development process from inside is rather risky: information must be interpreted carefully. Nevertheless if information is captured and analysed the software development process is under control.

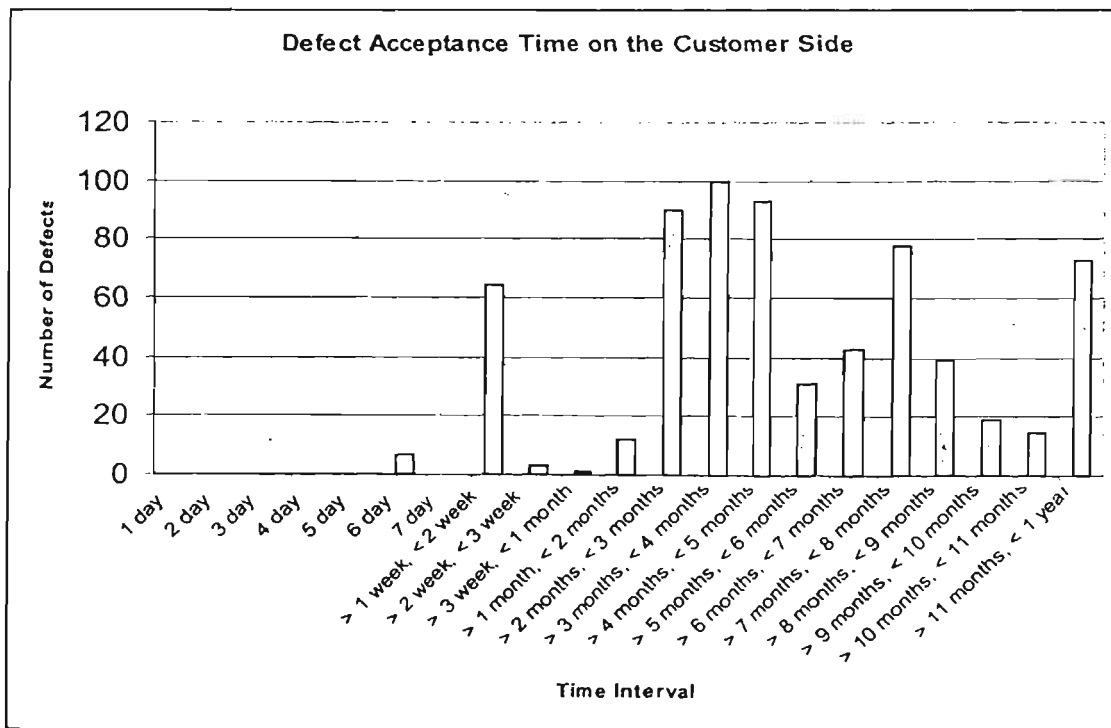


Figure 5. Sample chart for the project ALPHA illustrating defects' acceptance time.

8 CONCLUSIONS

GQM method is hard to apply without any customisation in companies having no any measurement and quality management traditions.

For companies having software development experience it is better to use data already captured. This allows achieving results with less additional investments.

It is relatively easy to collect data and a lot of companies have done it for years. The most difficult part of the measurement program is to use data effectively.

The experience in real projects shows that measuring software development process significantly decrease project development risk factors and improves customer satisfaction.

References

- Grady, R.B. 1992. *Practical Software metrics for project Management and Process Improvement*. New York: Prentice-Hall.
- Hetzel, B. 1993. *Making Software Measurement Work*. New York: John Wiley & Sons, Inc.
- International Organization for Standardization 1999. *Quality management systems – Requirements*.
- Jones, T.C. 1996. *Conflict and Litigation between Software Clients and Developers*. Software Productivity Research, Inc.
- Jones, T.C. 1998. *Estimating Software Costs*. London: McGraw-Hill.
- Software Productivity Centre, 1999. <http://www.spc.ca>
- Solingen, R. & Berghout, E. 1999. *The Goal/Question/Metric Method*. London: McGrawHill.

Baiba Apine, Riga Institute of Information Technology, Kuldigas 45, LV-1083 Riga, Latvia, phone: + 371 7067703, fax: + 371 7619573, e-mail: baiba.apine@dati.lv

Uldis Smilts, Riga Institute of Information Technology, Kuldigas 45, LV-1083 Riga, Latvia, phone: + 371 7067782, fax: + 371 7619573, e-mail: uldis.smilts@dati.lv

Uldis Sukovskis, Riga Institute of Information Technology, Kuldigas 45, LV-1083 Riga, Latvia, phone: + 371 7067703, fax: + 371 7619573, e-mail: uldis.sukovskis@dati.lv

Apine B., Smilts U., Sukovskis U. Software Measurement Practice to Address Customer Satisfaction

Monitoring of customer satisfaction is necessary to evaluate and validate whether the software under development meets requirements or not. Experience in implementation of measurement program in the software development company is analysed in this paper. The Goal Question Metric method is used as guidance for the measurement program, set of goals is determined and measurements to capture are defined. Case study describes implementation of measurement program in the real software development project.

Apine B., Smilts U., Sukovskis U. Mērijumu programmas ieviešana pasūtītāju prasību apmierināšanai

Rakstā analizēta mērijumu programmas ieviešanas pieredze programmatūras izstrādes uzņēmumā. Mērijumu programmas ieviešana realizēta saskaņā ar Goal Question Metric metodi: aprakstīti izvirzītie mērķi un vācamo mērijumu kopa. Darbā aprakstīts reāls programmatūras izstrādes projekts, kurā regulāra mērijumu uzkrāšana un interpretācija palīdzēja pārvarēt krīzi projekta izstrādes gaitā.

Апине Б., Смилтс У., Суковскис У. Мониторинг удовлетворения потребностей клиентов

Мониторинг удовлетворения потребностей клиентов необходим для определения соответствия разрабатываемого программного обеспечения требованиям. В статье анализируется опыт внедрения программы измерений на фирме разрабатывающей программное обеспечение. Методика Goal Question Metric применяется в качестве руководства для программы измерений, определено множество целей программы и накапливаемых измерений. Приводится исследование на конкретном примере внедрения программы измерений в реальном проекте разработки программного обеспечения.

PRACTITIONER'S APPROACH TO SOFTWARE COST ESTIMATION

Baiba Apine

Riga Institute of Information Technology

Skanstes iela 13, LV-1013 Riga, Latvia

E-mail: baiba.apine@dati.lv

KEYWORDS

Software project estimation, COCOMO, function point, software development

ABSTRACT

One of the most difficult and important software development activities is effective software estimation. Nevertheless it is one of the most important. Number of formal software project estimation methods have been developed. Several methods based on function points are discussed and problems dealing with practical usage of these methods are highlighted. Our experience in Basic COCOMO, COCOMO II and ObjectPoint methods is presented.

INTRODUCTION

Effective software estimation is one of the most difficult software development activities. Nevertheless it is one of the most important. Underestimating a project will lead to under staffing it, under scoping the quality assurance effort, and setting too short a schedule. That can lead to staff burnout, low quality, loss of credibility as deadlines are missed, and ultimately to an inefficient development effort that takes longer than nominal [IFPUG].

Overestimating a project can be almost bad, the project will take as long as.

Usually estimates are made using past experience only. This solution is good in cases when new project is of the same size and requires the same amount of effort. Number of formal software projects estimation methods have been developed. All of them have their own strength and weaknesses.

We are going to use these methods to estimate

the workamount of the software projects we have to develop and do it as early as possible. Software total cost means the cost of the whole software development process: specification, coding, testing, documenting, configuration management etc. This process could be estimated from various aspects: productivity - output of the software engineering process, quality - how software developed satisfies the needs of customer and functionality - how the software is built [PRESM]. Let us concentrate on functional aspect of the software development process: how to estimate the cost of the software functions. The first who proposed this method was Albrecht [ALBRE]. Now there are various modifications of this method.

METHOD

From the user's point of view system functions would be grouped in five groups [PRESM]: user inputs, user outputs, user inquiries (on-line input that results in the generation of some immediate software response), logical files (logical grouping of data, and there is no difference whether it is a database or temporary data for internal use only), external interfaces (machine readable interface used for data transmission to another application). Each item of the each group should be estimated as simple, average or complex. Multiple each counted item with appropriate weighting factor given in Table 1, and count total. The total you get is called unadjusted function points (FP). Determination of complexity is considered to be subjective, but there are resources giving more formal criteria for determination [BOEHM].

Measurement parameter	Weighting factor		
	Simple	Average	Complex
Number of user inputs	3	4	6
Number of user outputs	4	5	7
Number of user inquires	3	4	6
Number of files	7	10	15
Number of external interfaces	5	7	10

Table 1. Function oriented estimation weighting coefficients

Most of software cost estimation methods based on COCOMO (COCOMO II) use unadjusted function points as input data.

Function points are independent from the software development environment and this makes the method especially convenient in the large companies with a diversity of software platforms being used in projects. To estimate the number of lines of code, conversion coefficients are used. These coefficients differs for each software development environment and gives average number of lines of code per function point [PRESM], [BOEHM]. Figures of estimated lines of code are very important. In companies having large software development experience these figures may give very precise estimation of the person months and calendar months necessary for software development.

A lot of different factors should be kept in mind, for instance, previous experience in solving similar problems, scheduling, communications, etc. A cost driver refers to a particular characteristic of the software development that has the effect of increasing or decreasing the amount of development effort, e.g. required product reliability, execution time constraints, project team experience.

Method could be adjusted for software maintenance, reengineering etc. In most cases we use one of the COCOMO estimation models - the Early Design model. This model is used in the very early stages of a software project when little may be known about the size of the product to be developed, the nature of the target platform, or detailed specifics of the process to be used [BOEHM].

Another useful model is the Application

composition model (ObjectPoint method). This model addresses applications composed from interoperable components [BOEHM].

Our experience is that the most difficult and the most dangerous activity of the estimation process is calculation of function points.

ESTIMATION ACCURACY

In the earliest stages of software development life cycle, when the request for proposal is received from a customer, very little may be known about software development environment, staff involved, etc. Some functionality may be not specified or imprecise. Chart given in Figure 1 [BOEHM] indicates the accuracy of software estimation. In cases when request for proposal is the only available document with functionality of the system described very approximately, project could be overestimated or underestimated up to four times, that could lead to significant project management problems. It is very important to involve experienced system analysts in software estimation process to achieve more precise results.

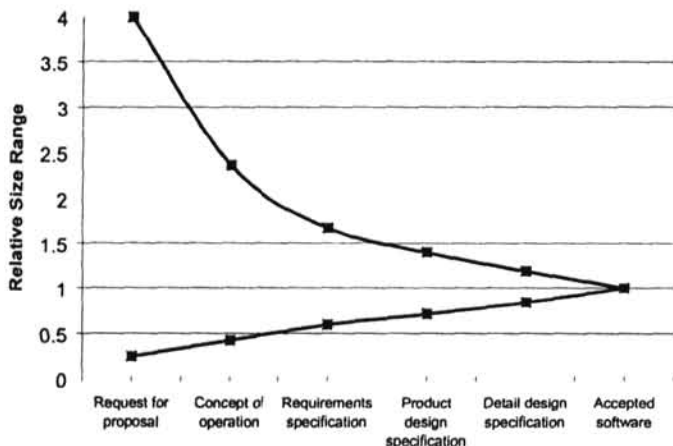


Figure 1. Estimation accuracy of software projects.

COMMON PROBLEMS AND SOLUTIONS

We use estimation methods in the earliest stages of software development process to get preliminary results when request for proposal is received. Hence there are not enough useful information which could be used for counting of function points or estimating the software development platform and staff being involved in potential project. Some information may be omitted in these documents. Additional information could be gathered during interviews with potential customers, but in most cases request for proposal is the only document available. Results of estimation must be included in proposal as accurate figures, but very often there is a lack of information to get them.

Not all customers provides clearly formulated requests. Some of them are brilliant experts in their own professional area, but have not enough experience in software systems. In this case interviews highlight basic software system demands. Results of interviews are the input material for both preliminary software estimation

and the software system's proposal.

In cases when at least software requirements specification is available, estimation results are very close to real development effort, estimation error is approximately 15% - 25% (see Figure 1). From the developers viewpoint, it is better to overestimate project (50% or more) than underestimate it.

Situation, when customer is another software company that outsources a project, differs from described above. Basically they already have done some estimation with their own methods and provide some initial figures. In most cases these figures are believable, but tend to underestimate the software development effort.

It is hard in the large variety of methods to choose the best one. For several years we are using COCOMO and COCOMO II methods for software estimation. The main reason we choose COCOMO was that it supports different quality levels of input data different quality. More precise input gives more precise output and it would be kept in mind.

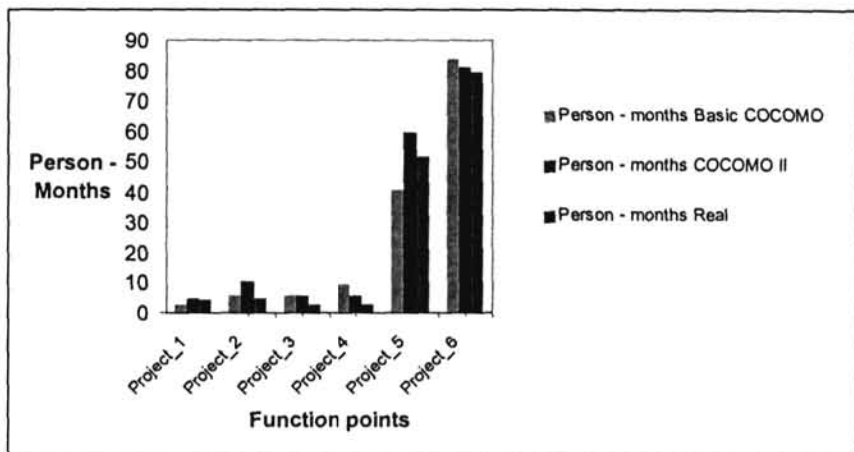


Figure 2. Comparison of estimation results

Basic COCOMO was the first method we started with. This method was quite good. Nevertheless some important aspects of software development process were ignored. Basic COCOMO method uses 14 very important cost drivers [PRESM] besides the classification of the software project's type - organic, semidetached or embedded. They cover software system itself, but do not touch the software development process. Cost drivers covering software development process are added in COCOMO II. For instance, whether the software development environment supports software development life cycle, whether developers' teamwork is supported, staff capability and interactions, etc. While COCOMO II method's Early Design Model gives more precise estimation results, we still use both COCOMO and COCOMO II to achieve more believable results (see Figure 2).

Chart given in Figure 2 shows software projects developed by using of Microsoft software development environments (MS Access, MS Visual Basic and MS Visual C++). All these projects were estimated using Basic COCOMO and COCOMO II Early design model with software requirements specification used as input

data for the estimation. The real workamount for the project development is compared with the result of estimation. Basically both methods overestimate projects. There are two projects underestimated by Basic COCOMO method (see Figure 2). The reason is that the software design specification was not precise and approximately 60% of code had to be thrown away. Percentage of breakage must be counted to adjust the effective size of the product.

Chart given in Figure 2 shows differences between estimation results given by Basic COCOMO and COCOMO II. The reason is appliance of cost drivers not supported by Basic COCOMO, but affecting estimation results of COCOMO II significantly. These cost drivers deal with personnel capability and experience. Values of the cost drivers are adjusted experimentally, so in different companies they may differ. Even in one particular company it is necessary to adjust values for each software development group.

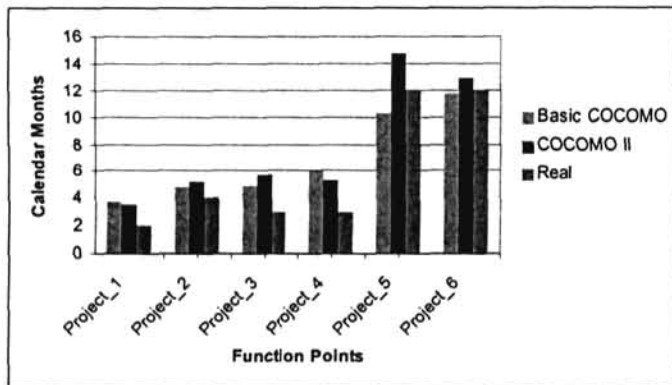


Figure 3. Time schedule estimation with Basic COCOMO and COCOMO II

Both methods Basic COCOMO and COCOMO II provide development schedule estimates. Chart in Figure 3 shows development schedule estimation results for the software projects, which development effort in person-months is given in Figure 2. Adjusting calendar months for software development process is very important activity. In most cases Basic COCOMO and COCOMO II overestimates calendar months necessary for software development. Nevertheless there are two projects underestimated in schedule by Basic COCOMO (see Figure 3 Project_5 and Project_6). Underestimation is relatively small: average 14% (see Figure 1).

Besides Basic COCOMO and COCOMO II we have tried to use ObjectPoint method for relatively small and simple applications, which could be built using standard data and user interface modules and without complex algorithms. Currently ObjectPoint method was rejected, because we have only few projects with software being developed by "ready to use" interoperable components only. This method tend to underestimate projects with additional programming activities (see Figure 4). Projects estimated with ObjectPoints method were developed using data access objects and standard visual objects connected to them (for instance, MS Access table and database grid control

connected to this particular database table) and some additional programming was necessary for data selection.

Software development environments are changing and coefficients used for transformation of the function points to lines of code must be adjusted for new environments. After examining 10 different projects developed using Microsoft software development tools in our company, we found the following coefficients for converting function points to lines of code (Table 2).

Sometimes software system is initially divided into subsystems or could be divided during the development process. The software system could be estimated as the whole one or by its subsystems. Figure 5 shows the difference between estimation results for whole system and for decomposed one both estimated by using COCOMO II. There are three reasons for higher values for whole project than for decomposed one:

- product complexity increases,
- additional management is necessary for larger project,
- staff communications are more time consuming.

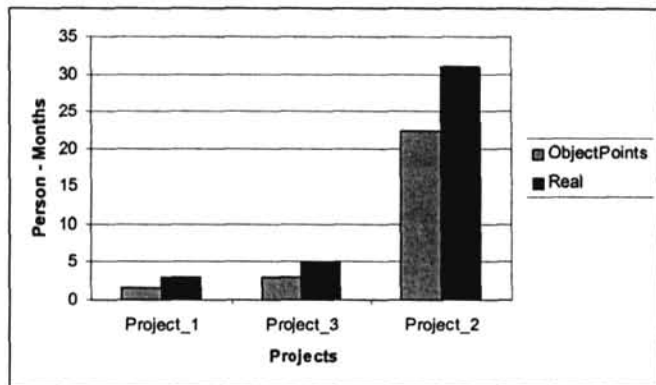


Figure 4. Software project estimation with ObjectPoint method

Software development environment	Source lines of code per unadjusted function point SLOC / UFP
Visual Basic 4.0, 5.0	25
Visual C++	27

Table 2. Coefficients used for converting function points to lines of source code

The conclusion from tendency given in Figure 5 should be: decompose complex system into subsystems and the price of the whole project will be lower. Besides it, chart given in Figure 6 shows difference in estimated calendar months between the whole system and decomposed one. This difference grows faster than difference in cost (see Figure 5). Divided systems need more development time. If time limit allows, some decomposition could be made to lower total cost of the whole project.

CONCLUSIONS

The main problem in the estimation process is lack of input information. Especially if formal estimation methods are used for software projects "could be". Results of estimation must be included in proposal as accurate figures, but very often there is a lack of information for

getting them.

The most complex and responsible part in the software estimation process is to get precise count of function points. The main reason of estimation faults is incorrect counting of function points. Therefore experienced system analysts could be involved in estimation process.

Although estimation process is based on formal models, it is the kind of art. It is very hard to make customer accept estimation results, which may be overestimated up to 4 times (see Figure 1). From the opposite side, upper level managers and administration tend to think that evaluation results are ideally precise.

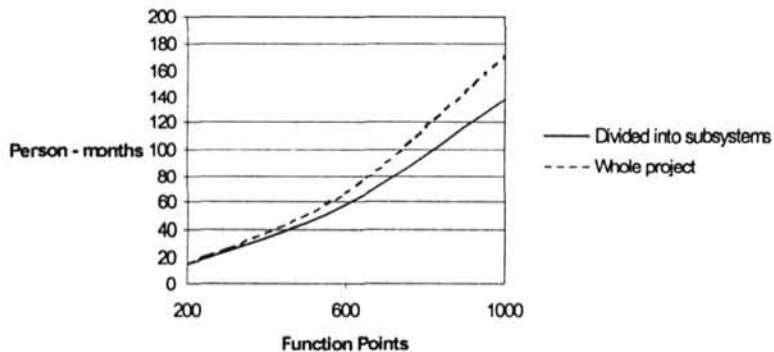


Figure 5. Estimation results for whole and decomposed system

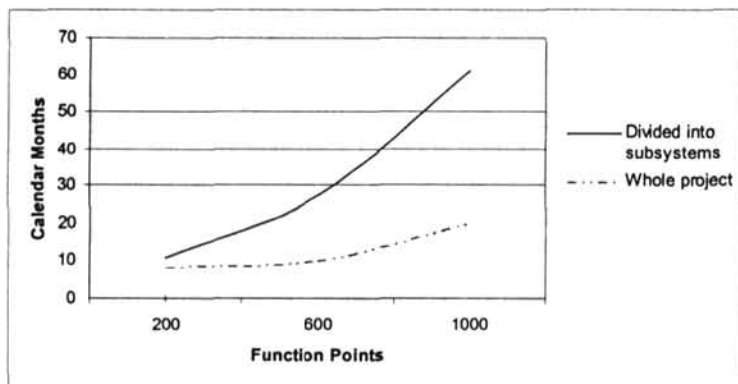


Figure 6. Estimation of calendar months for decomposed and whole systems

REFERENCES

[IFPUG] <http://www.ifpug.org/ifpug>
 [ALBRE] A.J. Albrecht. Measuring Application Development Productivity. Proc.

IBM Applic. Dev. Symposium, Monterey, California, 1979, pp. 83-92.

[PRESM] R. Pressman. Software Engineering: A Practitioner's Approach. McGraw-Hill, 1992, pp 41-91.

[BOEHM] C.Abts, B.Boehm, B.Clark,
S.Devnani-Chulani. COCOMO II Model
Definition Manual, University of Southern
California, 68 p.

Modeling methodology and tool for business systems: Registrar: *Baiba Apine, Arnis Kleins, Ojars Krasts, Uldis Sukovskis, Artis Teilans, Vita Zviedre*

**Riga Institute of Information Technology
Skanstes 13, LV-1013 Riga, LATVIA
E-mail: teilans@swh.lv**

The paper introduces a methodology and a tool for modeling and simulation of different kind of large scale business systems with a number of hierarchically structured objects and complex information, material or another flows between them. The tool for the discussed methodology is named REGISTRATOR which is being developed in Riga Institute of Information Technology. REGISTRATOR supports the modeling of organizational and/or technical systems. Behavior of any active object can be described using a simulation script language which is classified as the discrete event simulation language. The simulation process can be animated to get visual impression about system behavior, appearance time of every passive object and their routes within the system. During simulation experiments events are traced and statistics are collected. The simulation procedure is discussed in details. The main features of this tool

are: a relatively small number of concepts used in the building of the static model: only five object classes and 16 types of relations between objects, with the terms defined intuitive and easily interpreted even by a novice; REGISTRATOR allows for the implementation of different building strategies for the model, including: classic (top-down) strategies, reverse (bottom-up) strategies, mixed strategies; the possibility of checking the completeness and consistency of the model and of generating a list of inconsistencies and flaws

A VISUALIZATION AND ANALYSIS OF EXPERIMENTALLY GATHERED RESULTS IN AN USER-FRIENDLY MODE

Baiba APINE

Riga Institute of Information Technology
Skanstes St. 13, Riga, Latvia, LV1013

Abstract. Software development consists of several phases, where each phase has its own results. Language, which allows to describe collected results, their transformation and displaying, is discussed in this paper. A software tool is offered as an interpreter for this language. The language and software tool form a complex for data analysis. The complex is open and could be adapted for usage in different software system development stages. Object-oriented methodology for system specification design is used to show structure of the language and software tool.

Key words: computer aided system engineering, CASE, data analysis, information presentation, data description languages, software systems.

Specialists involved in software system development process use different software tools to study various aspects of the system. Tools accumulate information and create results, which must be organised and passed to the developer in the user-friendly mode (reports, charts, tables, etc.). Problem has three solutions:

- The first – to create results analysis tool for every aspect of software system. Results analysis tool provides minimum of calculations and reports, but supports data export to more powerful data analysis tool. This approach is very labour – consuming and depends on specific data.
- The second – to use already existing multi-purpose results analysis tools, spreadsheets, statistical packages, for instance. These tools are powerful, they have their own languages for data description. User must profit by the experience of using them. Frequently user doesn't need even fifty per cents of all features provided by universal data analysis tool.
- The third – to create a simple data description language, which de-

scribes collected information structure, transformation and visualisation, and software tool as interpreter for this language. Software tool is independent of the process, which stores results.

The third solution is provided in this paper. Solution is a midway of creating the particular results analysis tool for every software development tool and usage of universal data analysis tools. Process, which stores data, hasn't take care about collected data visualisation, but user won't need special skills for working with collected data (Fig. 1).

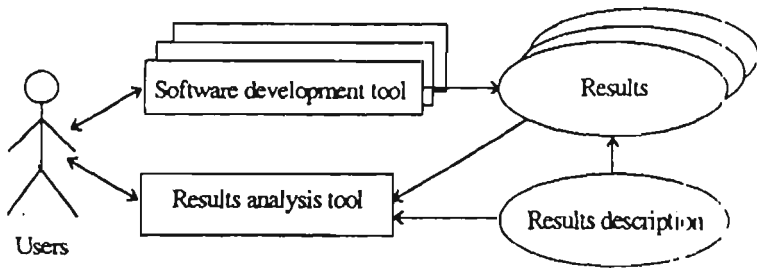


Fig. 1. Language for data description and tool as interpreter for this language.

Results analysis complex may be used by different processes. It is convenient to consider results analysis complex as closed system, which interacts with environment via separate objects, and choose the object-oriented methodology (Rumbaugh, 1991) for its specification design. Object model shows data description language structure as well as structure of the software tool (Fig. 2).

Object *Data description* and its subclasses show the data description language structure. Subclass *Data transformation* is the part of the language, which describes how get results from previously collected data by using of predefined functions (Fig. 3). For instance, a sequence of events is accumulated during some time interval. Each event in the sequence may be a seizing of a computer or it's releasing. Following attributes describe each event: time moment when event is registered, computer identification and event type (seizing or releasing). One of possible functions is calculate the total and average seances of computer usage during fixed time period.

Object *Data structure* shows, how data stores in data storage (Fig. 4).

One instance of the object Data transformation:

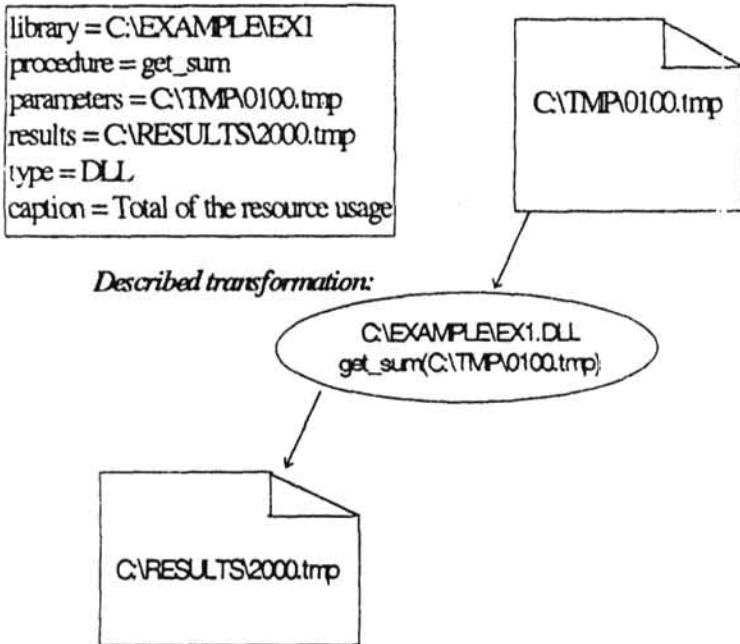


Fig. 3. Sample of data transformation description.

Instance of the object Data structure:

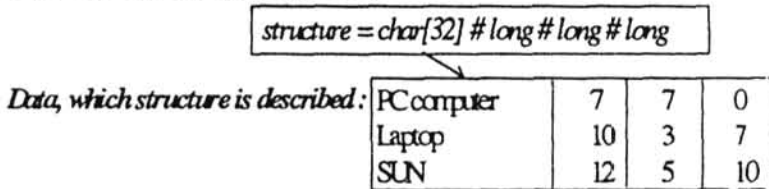


Fig. 4. Sample of data structure description.

Instance of the object Table description:

```
caption = Total About Resource Usage
headings = Resource # Total time used # Average seance length # Maximum
           seance length
fields = 0 # 1 # 2 # 3
format = # DURATION # DURATION # DURATION
sortDefault = 0
```

Data:

PC computer	7	7	0
Laptop	10	3	7
SUN	12	5	10

*Table generated by tool
according to this
description:*

Resource	Total time used	Average seance length	Maximum seance length
PC computer	7s	7s	0s
Laptop	10s	3s	7s
SUN	12s	5s	10s

Fig. 5. Sample of table description.

Object *Data visualisation* shows how collected and transformed data have been shown to user. Its subclasses *Table description* and *Chart description* show data representation in table or in chart. Every instance of these objects describes one table (Fig. 5) or chart (Fig. 6).

Software tool is an interpreter of the language. It takes collected data, transforms them, if necessary, and displays according to description. Object model highlights, which objects of the software tool are essential for interacting with an environment. For instance, user will interact with the software tool via object *Visual object*. This part of the software tool is interactive process, therefore it would be better to implement it in an event-driven environment. Part, which transforms (object *Transformer*) data may be considered as continuous process and implemented by using of a procedural environment.

Object Chart description instance:

X = 1
 Y = 2 # 3
 Xlabel = Resource
 Ylabel = Total time used # Average seance length # Maximum seance length
 format Y = Seconds # Seconds # Second
 style = 4

Data:

PC computer	7	7	0
Laptop	10	3	7
SUN	12	5	10

*Chart generated
 according to this
 description by
 software tool:*

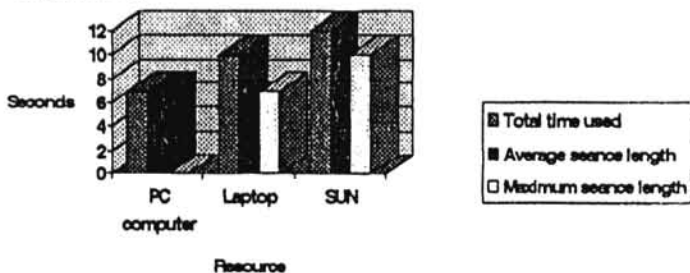


Fig. 6. Sample of chart description.

Software tool is implemented and used by some modelling tools. Software tool is the component of the set of GRADE (1993) system development tools. Its name is **Trace Browser**. **Trace Browser** is used by business modelling component for displaying of collected results (GRAPES-BM, 1995). Results may be described as follows:

- **Task name** – task name, at which events queue is considered. Task is a function, for which all necessary resources are fixed. For instance, task is to register an order and necessary resource for its carrying out is a computer.

- Event name – event name at the task identifies events queue. Event is an initiator of the beginning of the task execution. For instance, entering of an order invokes order's registration task.
- Maximum (average, minimum) length of queue-maximum (average, minimum) of events, located at the task during fixed time period.

Business modelling tool stores data. If business model designer wants to work with collected data, business modelling tool generates data description according to the data description language, calls Trace Browser and passes collected data and generated data description.

For instance, data are collected in data storage, which structure is described as follows: structure = char[33]#char[33]#double#double#double (Fig. 7).

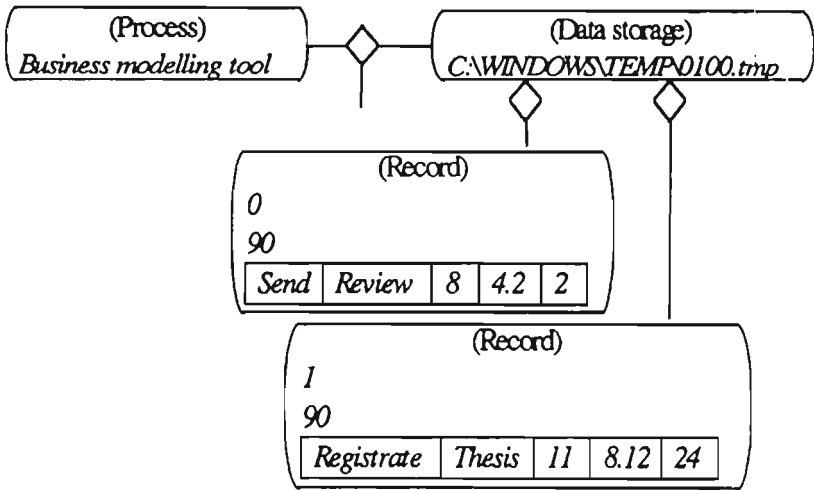


Fig. 7. Part of instance diagram, which shows data storage contents.

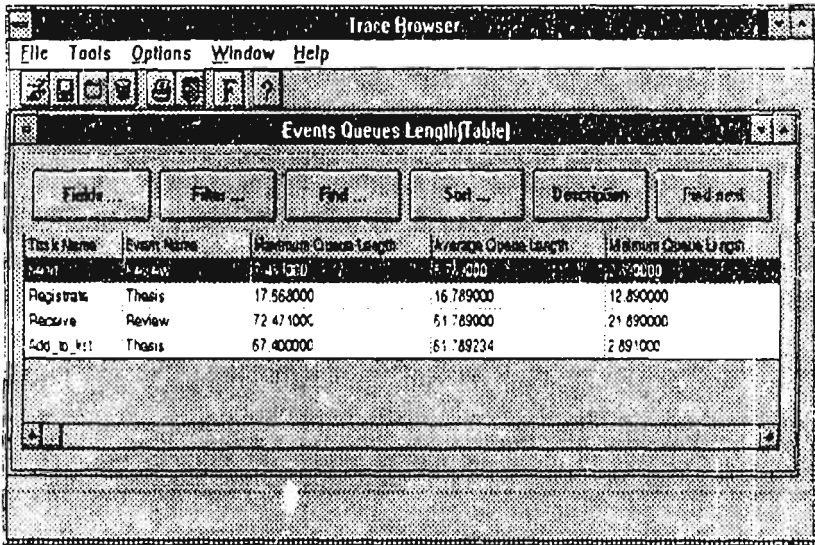
Business modelling tool assigns semantics to collected data, when generates data visualisation description. The following fragment is necessary to show collected data in a table (Fig. 8):

caption = Events Queues Length

headings = Task Name # Event Name # Maximum Queue Length # Average Queue Length # Minimum Queue Length

fields = 0#1#2#3#4

format = ##FLOAT#FLOAT#FLOAT



Task Name	Event Name	Maximum Queue Length	Average Queue Length	Minimum Queue Length
wait	wait	17.460000	16.710000	16.210000
Register	Thesis	17.560000	16.789000	12.890000
Receive	Review	72.471000	51.789000	21.890000
Add to list	Thesis	67.400000	61.789234	2.891000

Fig. 8. Table generated by Trace Browser according to table description.

Table caption is the value of the attribute *caption*. Values of the attributes *headings*, *fields* and *format* are lists separated by "#". Their values are interpreted simultaneously: value from data storage zeroes field is collected in the first column of the table in the same format as it stores in the data storage. The first column caption is *Task name*. And so on with every item of the lists.

The following fragment is necessary to display collected data in bar chart (Fig. 9):

caption = Events Queues Length

Xlabel = Task Name # Event Name

Ylabel = Maximum Queue Length # Average Queue Length
Minimum Queue

Length

X = 0#1

Y = 2#3#4

The heading of the chart is the value of the attribute *caption*. Labels on the horizontal axis are values from zeroes and first fields from data storage (attribute *X* value). Three values are shown for every record in the data storage. Values from second, third and fourth fields (attribute *Y* value).

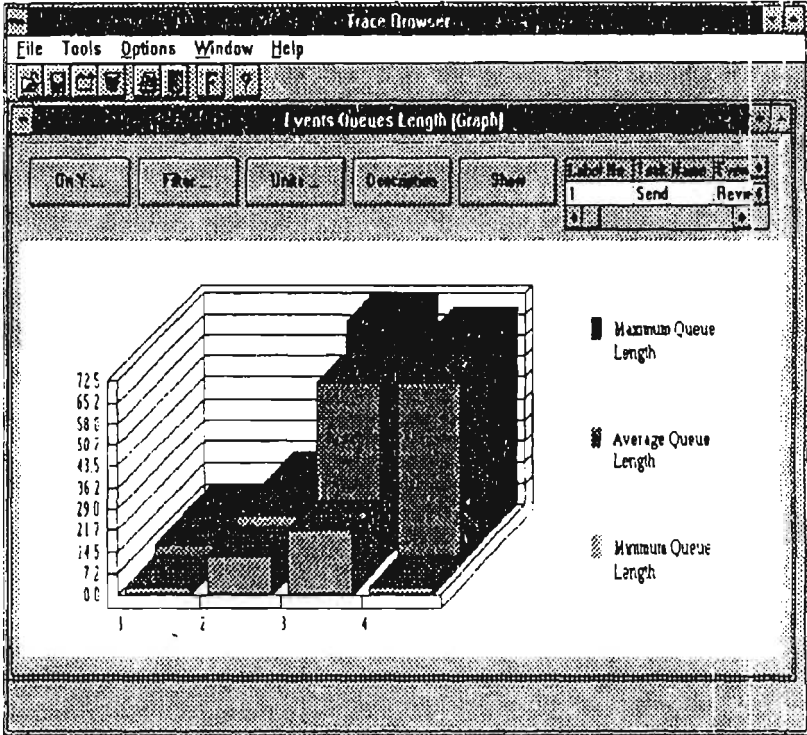


Fig. 9. Chart generated by Trace Browser according to table description.

Described approach has some advantages:

- Results analysis tool (Trace Browser) is independent from process, which stores data, and can be used by various processes.
- Results analysis tool may be updated with new functions for data transformation.
- It is better, if the whole set of software development tools use one and the same results analysis tool.

REFERENCES

- GRADE VI.0 (Windows): *Modeling and Development Environment for GRAPES-86 and GRAPES/AGL. User Guide.* (1993). Siemens Nixdorf Informationssysteme AG. 282pp.

- GRAPES-BM: Version 2.1: New Features of Language and Tool Support, Simulation Part.* (1995). August 3. 23pp.
- Rumbaugh, J. (1991). *Object-Oriented modelling and design.* Prentice-Hall, Inc. 542pp.

Received November 1995

B. Apine is a programmer at the Riga Institute of Information Technology (Latvia), Master of computer sciences. His current research interests include computer aided system engineering (CASE) and software systems, data analysis and information presentation methods.

APIE EKSPERIMENTO DUOMENŲ PATEIKIMĄ IR ANALIZĘ VARTOTOJUI PATOGIU BŪDU

Baiba APINE

Programų sistemos kuriamos etapais. Kiekviename etape gaunami rezultatai. Šiame darbe aptariama kalba, skirta gautiems rezultatams bei naudojamiems jų transformavimo ir vizualizavimo būdams aprašyti. Aprašytas tos kalbos interpretatorius. Kalba ir interpretatorius sudaro duomenų analizės priemonių kompleksą. Kompleksas gali būti pritaikytas bet kurio programų sistemos kūrimo etapo duomenims apdoroti. Kalbai ir interpretatoriui aprašyti panaudota objektinė metodika.