UNCERTAINTIES MANAGEMENT FRAMEWORK
Foundational Principles

Deniss Kumlander
Department of Informatics, Tallinn University of Technology, Raja St.15, 12617 Tallinn, Estonia

Keywords: Software engineering, Methodologies, Uncertainties.

Abstract: Uncertainties management is the crucial part of modern software engineering practices, which is mostly ignored by management and modern software development practices or dealt with reactively. In the result unhandled uncertainties do introduce a lot of threads and cause later delivery of projects or over-budgeting, which means the failure of the software engineering process in most cases. In this paper foundation principles of uncertainties management framework are defined.

1 INTRODUCTION

The main goal of software engineering is to implement tools for customers accordingly to their wish and vision that will let them achieve their goals faster, better and in a less expensive manner. The modern software engineering business faces a lot of new challenges because of permanently increasing competition on the market, high expectations of customers requiring constantly increasing quality, shortening developing time and increasing flexibility for proposing new features and changes. The flexibility becomes more and more important as our global business environment is starting to change with an incredible speed. The required flexibility and constant changes of the environment produce quite a lot of uncertainties for any software project ignoring of which is very risky or nearly impossible nowadays. Researches show that even the most modern approaches to software engineering still leave us in a position when up to 27% of all projects fail because customers are not satisfied with the delivered software (Bennatan and Emam, 2005) and only 20% of functionality in average is used “often” or “always” (Khan, 2004). This clearly demonstrates existence of gaps between developed software (features, budget) and customer expectations. Therefore it is important to explore reasons when the shortened development cycle with constants demos and constant collaboration inside the team used in modern approaches are still not able to bridge this gap. Mostly those are connected to different uncertainties arising in projects since not all of them are temporary. Therefore the uncertainties management becomes very important in order to ensure software engineering projects success (Kumlander, 2006a).

The aim of this paper is to propose foundational principles of the uncertainties management in order to shift current software engineering practices towards a new approach involving uncertainties management and providing so much desired flexibility in complex projects sufficiently increasing the quality of software engineering process.

2 UNCERTAINTIES MANAGEMENT IN CURRENT PRACTICE

Modern software development approaches mostly concentrate on either preventing uncertainties somewhere early in the work-cycle doing a complete design before passing to software implementation stage (Kumar 2002) or eliminating them on a constant base handling those as any other errors - for example by implementing short sprints in agile methods (Beedle and Schwaber, 2001) in order to review software by stakeholders as soon as possible and consequently accept or reject the result eliminating the uncertainty of how the product should look like, function and so forth. Unfortunately it is very much simplified approach as uncertainties could stay on a relatively permanent base and only temporary uncertainties can be eliminated by the methods described so far. Therefore, in this paper, under uncertainties management we mean not only eliminating those, but also handling uncertainties rapidly and ensuring the best software engineering process result despite of uncertainties arising on different stages of the project. We try to achieve an increase in the resulted software quality not despite all uncertainties we have in the projects, but considering uncertainties and aligning required activities to existing issues. Under activities we mean both standard software engineering activities and activities to handle uncertainties, risks and so forth. Assuming that the testing on bugs (run-time errors or incorrect calculations) is handled by standard methods and accepting that 100% bug free software is nearly impossible (at least for commercial software), in this paper under quality we will mean a size of gap between developed software and customers expectations. The smaller this gap the better is quality of the delivered software. Alternatively we can be targeting reducing efforts we need to spend in order to develop software with on same, constant quality level. Obviously in case of any problems occurring in the project we could increase involved resources and bridge the quality gap during additional hours by extra developers. Therefore we can conclude that quality and resources are mutually dependent. Unfortunately practice shows that it is not always true – resize one angle of the quality triangle (resources, quality, time) will guarantee we could keep the opposite side the same size and if it doesn't happen then we are likely to fail in the engineering process. Here we would like to emphases that the reason of failures is an impossibility to resize the triangle at the final stages of the software engineering process since it is too late to do something sensible and the need to resize mostly occurs due permanent uncertainties resolution into demands now.

3 FOUNDATIONAL PRINCIPLES OF UNCERTAINTY MANAGEMENT FRAMEWORK

3.1 Transparency

It is crucial to ensure transparency dealing with uncertainties by providing visibility of any details on each uncertainty issue for all teams involved into the process. It can be debatable who needs or should have access to organisational, may be highly confidential, data, how access will be defined (all members of the team or just key persons of the team), but there is no doubts that the biggest mistake that can be ever made is to host uncertainty information exclusively inside local teams. Consider as examples the following cases:

- Requirements uncertainty is kept locally within consultants, business analysis teams records;
- Risks recognised in the development team;
- Low coverage by test cases detected by the test team.

Notice that modern approaches try to address certain individual uncertainties, but clearly all they lack a methodological approach dealing with all uncertainties types. Consider for example the agile development key elements (Cockburn, 2002). Any method included into the agile practice tries to increase visibility of the software implementation process for any involved (but not "committed") parties like stakeholders, designers etc. It is done by publishing graphs of the desires completion percentage compared to the actual one, visibility documents on the sprint features and so forth. The other uncertainty class the agile practice deals with as a key methodology element - uncertainties in requirements were the software implementation result is different in compare to consultants, customers or designer expectations. Notice that there are a lot of reasons why the result and expectations could differ (Kumlander 2006b) including a certain probability that expectations will change during the project life-cycle. Considering all this we still should confess that the agility framework does lack the systematic approach managing uncertainties. Sometimes the engineering process converted into a time-race replacing proper implementation addressing real uncertainties drivers by quick "visible" releases containing factual errors and so having to do a lot of extra work later, when uncertainties are resolved. Therefore the first step that should be done dealing with uncertainties is to increase transparency of those by implementing a central information repository:

- Containing enough information about uncertainties nature, time-frame each is likely to be resolved and possible resolution alternatives;
- Providing access for different teams to this information on constant basis;
- Capturing evolution of the uncertainty including history of changes, discussions and feedbacks from customers and experts.

"On constant basis" means here that it is not enough to publish the information. People should be both aware that it is published and know how to obtain data. Visibility of data is defined by existence of data, quality of data and accessibility of data. If data is inaccessible then it does not exist. The same can be said in case the information repository has complex or low usability user interface.

3.2 Handling

The transparency of uncertainties provides a basis for handling those. There are uncertainties of different kinds and one distinguishing property is uncertainties' stability from the time perspective: they can be either permanent (information required to resolve those cannot be obtained at the moment) or temporary (we are ready to resolve, but it hasn't been done yet). Obviously those two types require different approaches and strategies to handle them correctly. Therefore been aware of existing uncertainties means start to handle them on permanent basis synchronising different types of activities with this information. We need to assess each uncertainty considering all available information and plan our tasks in order to minimise overall efforts required to engineer software in given conditions. Although all this seems to be easy, it is rather easy to define than to follow on the permanently base and use as a basis in all planning exercises, and therefore is constantly missed by methodologies and managers.

Looking into the past it should be mentioned that some authors have treated uncertainties as risks and so advised using risk management and assessment methods to handle uncertainties existing in projects. It seems not to be quite correct from our point of view. There is a risk of certain negative consequences if uncertainties will not be handled correctly, but it is just one side of uncertainties and therefore risk management will not be able to handle uncertainties in all aspects. We still need to consider entire context related to it, interdependencies between different features and teams, different solutions alternatives and the time-perspective. The intensive handling means also additional information to be added into the uncertainty information package. This information will be crucial for those who should clear our uncertainties or develop something connected to such area.

3.3 Light Agility

The agility still plays an important role in eliminating certain types of uncertainties, mainly temporary, where the only missed information is lack of visualisation and implementation of the proposed solution. Having shortened the cycle we could stimulate quick resolution of temporary uncertainties. There are no points to describe intensively the agility manifesto in this paper since all agile methodologies are well-known and extremely popular nowadays. There are a lot of articles describing agility methods, their pros, contras and implementation practices (Cockburn 2005, Braithwaite and Joyce, 2005, Stapleton, 1997).

The light agility term in this foundational principles refers to the fact that not all organisations can use agility practices, not all of them right and those do not resolve the uncertainty problem in every aspect.

- It is not possible to apply those in distributed organisations due communication gaps;
- There are a lot of ordinal people that cannot be included into self-organising teams;
- Agility practices do ignore long-term uncertainties;
- Constant, but mainly short-time collaboration is not possible in complex projects requiring extremely complex features and so producing time consuming review cycles as described in the next subchapter.

3.4 Communication Ambassadors

Constant collaboration (Rauterberg and Strohm, 1992; Forsgren, 2006) in many cases is either impossible or is not enough to resolve complex uncertainties in complex projects. Uncertainties could be either produced by communication gaps or would require bridging those in order to resolve uncertainties correctly. Communication gaps are defined as either a problem occurring in communication between people (group of people) where the information is either lost or sufficiently corrupted (Kumlander 2006b) or a problem of improperly used resources (including the available time) since the communication process is very slow. The first part can occur due:

- Weak ability of persons to communicate, for example to express a message;
- The lack of context of the message in the transferred communication flow, including previous discussions, differences in backgrounds, information on the environment the message was formulated and so forth.

The second will be produced if involved parties are under pressure due other projects, sufficient time-differences between communicating locations or they are not really motivated to participate in the immediate and continuous communication.

Moreover the communication doesn't only mean the transferred message. It is also a process that should be managed. The problem is that existing managers cannot be responsible for all kinds of communications and self-organising teams cannot be used for long term communication to external sources. Therefore there is a need to appoint certain people to the "ambassador" position managing the communication process by:

- Chasing other people to provide communication, do reviews, talk to external people and so forth;
- Supervising communication flow and ensuring that context is also transferred;
- Directing the flow to "right" persons;
- Ensuring none-zero communication and monitoring the communication from the time-perspective;
- Synchronising efforts of different teams working with different issues in different projects and been in different time-zones.

A typical case of projects, were uncertainties arise is a complex-features' project. Here the required level of revisions to completely understand delivered features during users acceptance testing demands several days of work. Therefore the longer collaboration cycle than modern software engineering practices would account with should be used and communication supervising ambassadors will be crucial to have ensuring permanent and rapid communication. It is also important to use such ambassadors to provide visibility of any processes happening in isolated teams. Consider for example a distributed organisation when the distance team start to become later in their activities. Sometimes it is critical to react on such processes immediately, but the lack of transparency will not allow doing that. Ambassadors should be used to intensify exchange of information and increase transparency of the entire organisation. Notice that isolation is not always physical. It is possible that a group of people included into the project belongs to another hierarchical structure, so been in the same location, but isolated due management reasons.

3.5 Well-defined Work Procedures

Uncertainties are introducing into the project enough problems to generate even more having work procedures' gaps. Well-defined procedures will introduce a structure into chaos produced by unspecified features. Besides they are required to cope with different issues resolving uncertainties. It should be specified who is responsible for what and what are procedures deciding on alternatives.

Notice that having clear rules doesn't always mean prohibiting certain activities like late expansion of the scope. Those just define how it should be done, who will be paying the extra cost and how trade-offs are made. The development team is stressed enough in such projects to not be responsible neither be blamed for not resolving uncertainties correctly in time and functional perspective. It is also important to define how a feature affected by any uncertainty is skipped from the release and what should happen with it next.

4 POTENTIAL BENEFITS

In this chapter we would like to review potential benefits arising when uncertainties are properly handled by the proposed framework.

The first, but not the only one, is an overall decrease of time and efforts required to develop the software as risks connected to uncertainties are properly managed and handled now.

The second benefit can be demonstrated assuming the opposite to what is proposed. If uncertainties still exist and are not handled correctly then the team members' motivation decreases producing dramatic drop of the team performance. Such negative impact can be explained by the following reasons. First of all, if uncertainties stay hidden then gaps between expectations and software arise extremely suddenly and bridging those requires sufficient efforts applied immediately. This increases overall pressure on team members and consequently produces considerable stress. Stress and depression, coming from an impossibility to foresee or avoid such problems, affecting team members. In the second case the schedule can be reluctant and there is no pressure from stakeholders, which is quite hard to imagine in the modern competitive business world, but anyway it happens sometimes. Despite of this gaps should still be bridge and this should be done constantly. Bridging normally means re-implementing functionality throwing away hours or days spent so far. People generally hate such situations as mostly see that they have spent time on tasks that were not properly addressed by initiators. This produces certain conflicts within the organisation or de-motivate people do their tasks properly as they tend to believe that it should be reworked later anyway.

The third benefit we like to demonstrate is an increased mobility of the team and consequently of the organisation. The proper uncertainty handling will let the team to be ready to support uncertainty resolving just in time it can and should be resolved. In other words exactly when all involved parties are able to do the work, their have acquired all required information and the team is in a position to start functionality specification, development and testing activities. It is extremely important to avoid attempts to resolve uncertainties both earlier wasting time and depressing and later leaving certain questions open and so requiring extra communication rounds, letting business drivers to switch to other topics loosing their concentration on the current one.

5 EXAMPLES

This chapter is designed to present project cases from our practise were uncertainties arose and were not properly handled. In the result projects were declared as failed. We leave it to readers to see how earlier proposed method had to be applied and what benefits will be derived as following the framework in those cases is a straightforward process.

A target of the first project to be revised was to develop an accounting system addressing quite a complex model accordingly to an international accounting standard recently announced to be mandatory in some world regions. The description of the accounting model contains a lot of interconnections and has quite restricted possibilities to divide those into sub-steps. The company had some consultants, but the topic was too novel for them and therefore external consulting resources were required in order to progress the project. Unfortunately external resources were highly demanded on the market at that moment and therefore hardly available. The following key-words can be used to summarise the project nature: a novel approach without clear implementation practise at the moment; sophisticated, complex requirements; long software implementation cycle with massive interdependencies.

The implementation result was the following. The project took a considerable amount of time to deliver to market – approximately two years instead of envisioned six months. During first iterations consultants failed several times formalising the required functionality even using external resources. Under failing we mean a set of attempts to release drafts to development that were proved to contain mismatches of logic that were discovered when developers tried to implement those. During the first year quite a little progress was made and ten or so cycles were made until the functionality have been compromised in certain key areas to have it finally delivered to development in a more or less acceptable way. A constant lack of information were observed due complexity of the features on all levels of the work-process. Therefore it was decided to push the project by switching to one of the agile practices in order to have a possibility to look at results and discuss arising difficulties. Unfortunately the lack of consultants' time and constant postponements of external resources availability have factually degenerated the review process into a simple "try, see, fail, not approve and try again" approach. Key drivers for this were:

- It is uncertain how we could implement it, so let developers do something until designers think;
- The released part is so complex that cannot be understood at once during the demo, so everybody needed sufficient time to test and think through before additions can be proposed. It was also hard to fit such sufficient time intervals into everybody schedule and developers were waiting for this unable to continue as no other functionality is proposed – all designers involved into analysing the current step.

Summarising all previously said, the project was finally completed, but it took sufficiently more time than it had to take. In the result the product lost a lot of opportunities and became outdated immediately after it was released. A lot of functionality items were not addressed during the lost time producing a lot of existing customers' criticism. The team motivation level decreased sufficiently and some key persons left the company looking for a better, well-organised work place in order to spent their efforts and life on something useful (as they said in private conversations before their left the company).

The second project was started by an insurance company as an internal project using recently hired young developers in order to comply with requirements defined by the state motor vehicles insurance policies regulation. A set of alternative solutions were proposed requiring different hardware implementations. Having to sell insurance policies via different channels the system required a possibility to set a point of sale in different company branches, at partners and resellers and all of them had very different software and hardware installed. Therefore one uncertainty raised in the project was a long running debate involving the management team about financing one or another alternative. Another problem was a lack of certainty of young developers' abilities and consequently uncertainty on plans and schedules defining features that should be promised in the scope of the project. This has produced uncertainties for other departments that were supposed to use this new system.

Finally there were a lot of problems discussing the required features list with different departments. First of all there were a lot of communication gaps as it constantly seemed that developers and insurance professionals are using completely different languages. The biggest problem was the lack of experience among young developers. Fortunately this problem was taken under control by rapid releases, but unfortunately there was a lack of believe among insurance people that the project will be completed and therefore they rarely attended demos or were not concentrated during those. Secondly the IT team found themselves been in a position between different departments getting conflicting requirements. The only resolution here was to post them to higher management and that produced sufficient uncertainty due actual lack of central decisions force and vision. In the result the team had to re-implement a sufficient portion of functionality.

Summarising all earlier said, uncertainties of that project were produced by undecided size of investments stakeholders were ready to put into the project, limited abilities of personnel producing a mess in interconnections between teams, scheduling chaos, an uncertainty of what and when will be delivered and opposite opinions on how the system should function coming from different departments.

6 CONCLUSIONS

Uncertainties management is the crucial part of modern software engineering practices, which is mostly ignored by management and modern software development practices or dealt with reactively. In the result unhandled uncertainties do introduce a lot of threads and cause later delivery of projects or over-budgeting, which means the failure of the software engineering process in most cases.

In this paper foundation principles of uncertainties management framework are defined. First of all it is transparency of uncertainty issues including their context, past and current discussions and infrastructure elements ensuring visibility of them to all involved parts. Secondly, it is adopting elements of agility practices in order to resolve the easiest class of uncertainties – temporary. Thirdly, it is handling and monitoring uncertainties proactively including planning resolution process, considering dependencies. Finally communication supervising ambassadors ensuring that all communication gaps are bridged and well-defined work procedures.

REFERENCES

Bennatan, E. N., Emam, K.E., 2005. Software project success and failure, *Cutter Consortium, http://www.cutter.com/press/050824.html*

Khan, A. A., 2004. Tale of two methodologies for web development: heavyweight vs agile, *Postgraduate Minor Research Project*, 619-690.

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

Somerville, I., Jane, R., 2005. An empirical study of industrial requirements engineering process assessment and improvement, *ACM Transactions on Software Engineering and Methodology*, 14(1), pp. 85-117.

Kumlander, D., 2006b. Bridging gaps between requirements, expectations and delivered software in information systems development, *WSEAS Transactions on Computers*, 5(12), 2933-2939.

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

Forsgren, O., 2006. Churchmanian co-design – basic ideas and application examples, *Advances in Information systems development: bridging the gap between academia and industry*, Springer, 35-46.

Kumar, R., 2002. Managing risks in IT projects: an options perspective, *Information and Management*, 40, 63–74.

Cockburn, A., 2002. *Agile Software Development*; Addison Wesley, Reading, MA.

Beedle, M., Schwaber, K., 2001. *Agile Software Development with SCRUM*, Prentice Hall, Englewood Cliffs, NJ.

Braithwaite, K., Joyce, T., 2005. XP Expanded: Distributed Extreme Programming, *6th International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Springer, 2005, 180-188.

Stapleton, J., 1997. *DSDM Dynamics System Development Method*, Addison Wesley, Reading, MA.