Migration of Software Projects' Goals and Expectations

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

Abstract: -.The number of failed software engineering project remains very high despite numerous attempts to decrease that by employing different modern software engineering methodologies. There are different reasons for failing and only one of those is studied in this paper: a mismatch between delivered software package and customer's expectations used to verify this software. The paper shows that there exist several cases, where software that corresponds to the initial requirements appears invalid at the project completing phase. This happens because goals and expectations are not static sometimes and can mutate under different influencing factors. Moreover one of those factors can be the software project by itself as is sometimes released in several steps.

Key-Words: - Software design, software project goals, software project expectations, migration

1 Introduction

The ultimate goal of software engineering process is to provide customers with tools that will help them run their business in a better way. Nowadays increasing competition and globalisation of business demands much higher quality of the released software, much shorter development cycle and increased flexibility of defining requirements. The proper software implementation over the low quality one provides benefits for both projects sides – mostly because of saving resources, quicker applying the software that business needs to have, better imago for software providers and some others. Unfortunately many software projects are far from the described goals and the number of failing projects is still high. Researches of projects fails reasons show that up to 27% of all projects fail because customers are not satisfied with the delivered software [1] and a lot of other projects fail since those do not fit into budgets. Sometimes it happens since those projects are having difficulties with meeting customers' requirements and are rebuilding the software again and again. A situation with the actual percentage of functionality that is in use doesn't look better: only 20% of functionality (in average) is used "often" or "always" and 16% "sometimes". The remaining 64% is either never used or used just occasionally [3]. Of course, certain percentage of rarely/never used functionality belongs to activities that should be implemented, but is normally never used, but at the same time this high percentage demonstrates that a lot of features do not meet end-users of customers' expectations and therefore remains unused. The increasing complexity of nowadays software systems doesn't allow to hope that the situation will become better in the future by itself. The described above facts shows that a problem of mismatches between delivered software and customers' expectations do require serious researches nowadays. It is crucial to understand why such mismatch exists and identify reasons that lead to the described situation.

In this paper the mismatch is seen as a result of changes in the software engineering projects' goals and expected functionality, which happens after a project is started because of different reasons. Moreover the software project is proposed to be seen as one of those reasons as the iterational development process can be an evolutional mechanism forcing goals and expectations to migrate from one formulation to another.

The paper is organised as follows. The section 2 discusses software engineering methods including output produced by those methods as a result of the software development process. The following section examines how stable software goals and customers expectations are and researches reasons why those do change. The last section concludes the paper.

2 Software Engineering Methods and Output

It is possible to separate software engineering methods into two major types: a traditional software development and a business oriented "modern" technologies [2, 7].

The traditional method asks to do a project in one big step and therefore moves from stage to stage only after the previous one is completely done, i.e. the development phase starts only after the design is 100% complete. This method is basically used for implementing error critical systems, where requirements are mostly stable during the project. Examples of such projects could be a missile launch system's program or submarines' systems.

The second software engineering approach is business oriented and therefore normally releases software in steps (iterations) since:

- Requirements are usually unstable and there should exist a possibility to modify a project during its development time;
- Time frame is shorter for business oriented projects than for "critical" systems engineering and this leads to the rawer design. In results the developing software needs a "milestones" control over the implementation phase to see its correctness. Milestones' releases are either demonstrated to the customer to verify concepts or even goes to the production;
- Business (companies) would like to have a better control over the development process (read over and the project's budget and schedule) and therefore uses those intermediate releases to track the project's progress;
- It is possible to install and run partly ready software; software is normally less tested than for the "critical" systems to minimise its cost; business is interested sometimes in receiving some parts of the software (for example of an information system) as soon as possible etc.

All this demonstrates basic differences of the discussed methods. An important difference aspect to be especially studied in this article is results of those software development approaches. Normally the traditional software development results in one release of the developing software in the project's end. Here software (i.e. result of the developing process) is compared to the ordered one. Basically it is a comparison of customer expectations and the delivered "reality". If those differ significantly then the project is defined as "failed". The modern business oriented methods like agile, iterative etc are satisfying business needs by doing a set of releases, which are presented to the customer. This enables to focus the software project on set goals if any mismatch occurs during the project and should increase the overall success factor for projects that are employing this type of software engineering methodologies. At the same time the method of detecting if a project failed remains the same, but some variations could occur like for example: the project is stopped earlier, the project is prolonged etc.

3 Mutation of Software Projects' Goals and Expectations

Software engineering projects goals and customers 'expectations were considered so far in most cases as something unchangeable. The only exception was projects with incompletely formulated goals, but the changes type in those was always: "predictable" and the variation factor was always under control at the project starting phase. Generally saying the software project is normally planned and designed in two phases: the first phase considers it from an idealistic perspective, i.e. as something that can be delivered immediately at certain agreed date, containing features that will be required at this time; thereafter the projects' schedule is discussed using the tradeoffs triangle between time (schedule), number of features to be included and available resources to be used in the project. At the same time many projects are reported to fail because of mismatches between delivered software package and customer's expectations since expectations were changed during the project, i.e. although the delivered package does meet requirements those are not actual any longer and software cannot be used. This is an effect to be descried in this paper and can be defined as a change of projects goals and customers expectations during the project. It is even possible to use the "mutation" term since the changes to be described are goals and expectations structural changes caused by different, mostly environmental factors and are random by the nature (i.e. cannot be directly controlled by the project team or foreseen). Expectations mutation factors can be divided into the following groups, which are described more after the list:

- External factors: external factors are mostly environmental ones, but the main proposition to be done: the project can serve as an environment for itself also since will influence customers (that are outside the team) and lead to changes in their minds;
- Internal factors: mostly ideas that can arise within the project team on how to make software better including new (proposed by the team) features.

An ideal environment for the software engineering process is an environment, which ensures that there is no difference in the outside world between the project start and the project end moments. Conditions, the customer's wish to get software and other environmental elements remain the same including software project's goals and specifications. This static environment eliminates uncertainties and simplifies moving toward declared project's goals. Unfortunately the environment is not static in many cases and therefore the question of environment stability should be considered by the project management in order to avoid project's failing. The unstable environment can affect the project and some those examples were studied in several previous works dedicated to uncertainties in software functional specifications and requirements [4, 5, 6]. These works have shown how unstable requirements and changes in the environment could affect the project and how it can

be avoided or addressed to meet customers' expectations. In this paper the uncertainty question is stated for the project's finalising stage instead of the starting one, i.e. the project releases are proposed to be seen as a part of the environment that could also affect customers' expectations and follow to certain changes.

Until now the environment was always defined as an external element from the software project point of view. At the same time the migration of goals and expectation was seen as a shift because of uncertain requirements and mistakes that are fixed during the project [4]. The number and the nature of those fixes were considered as primary factors of the expectations migration process and therefore were defining the size and the nature of the migration process. In this paper the customers' expectations and goals changes are proposed to be seen not as result of fixing mistakes, but as a result of the developing process also. Originally intermediate releases were designed to re-focus software development process on declared goals in case there are wrongly implemented features. In reality all releases that are shown to the customer could make him to understand how the ordered software can be developed in a better way; s/he could get more information from different sources about modern technologies and decide to implement those; intermediate releases could highlight certain areas where the software will not provide much information and where it is even more successful that was believed and therefore requires more in-depth research and development and so forth. This is an evolutional process of reformulating requirements, increasing and decreasing expectations from the ordered software. At the same time looking at the developed component the customer could better understand what he is going to have and all general phrases that probably were included into the project's specifications could crystal out into an exact knowledge, which is not always the same as developers have. Intermediate releases influence the software expectations migration a lot (read expectations that will exist at the project end stage and will be used to verify the delivered software) and their underestimating could be a cause of many projects troubles. Notice that changes of expectation because of reasons that are not connected to the project directly could appear in the single release projects also. Therefore it is important to contact customers on a permanent base in order to be informed on changes as soon as possible. The software package is developed during certain time and this time can affect the project via influencing the end stage environment at all. Especially it is true for the single-release project in dynamically changing sectors.

Another type of mutational factors locates inside the project team. The more software engineers work on a project the better they usually understand goals of that, reasons to have it and ways it should work. Basing on previous skills they can start to see more, better ways to do something, organise processes, provide information and so forth. Those ideas could affect customers' expectations, projects goals, the number and nature of releases if are presented to customers. It depends on the type of the customers – project team relation. One case could be a project team that is an IT department of a big organisation. In this case they do communicate to the customer-department a lot, tend to provide as much as possible and therefore are free in distributing their ides. An opposite case could be a contractor-company that just tries to do the project as soon as possible minimising internal costs and therefore will do as it was specified without any attempts to extend the project or make it better.

4 Conclusion

Unfortunately nowadays software development methodologies do not secure software engineers from failing and the number of failed projects is still very high. The percentage of functionality that is in use for successful projects is also quit low and this fact indicates that a lot of "successful" projects are quite far from a complete success. Projects that are able to produce an output in the end of ends are normally verified by comparing the delivered software to the goals and expectations and sometimes those are different matters. In this paper the problem of mismatching between software projects outputs and delivered packages is researched. It is a wide believe that software is verified using goals and requirements that are defined before the project has started or during the first stages. Those goals and expectations are not static in many cases as the environment in which the software is developed is not static also. All this influence criterions that are used to accept the project and therefore an effect of goals and expectation migration can be identified. Main factors that are causing the migration process can be divided into external and internal from the project point of view. The first one included mostly environmental factors. The second one includes ideas and suggestions that the project team could do. The most important proposition here is that external factors can be activated by the software project, i.e. iterational releases could act as an evolutional mechanism leading to expectations changes. It is crucial to consider this fact during the implementation and planning phases and address it by either removing, if this evolution is seen as a problem, or by employing, if it is seen as an opportunity.

References:

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

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

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

[4] D. Kumlander, Software design by uncertain requirements, *Proceedings of the IASTED International Conference on Software Engineering*, 2006, pp. 224-229

[5] D. Kumlander, On software design and development supporting requirements formulation, *Proceedings on the 10th WSEAS International Conference on Computers*, 2006, p. 818-825

[6] D. Kumlander, Supporting software engineering, *WSEAS Transactions on Business and Economics*, Vol. 3, No. 4, 2006, pp. 296-303

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