Change Risk Assessment: Understanding Risks Involved in Changing Software Requirements

Byron J. Williams   Jeffrey Carver   Ray Vaughn

Department of Computer Science and Engineering
Mississippi State University
P.O. Box 9637
Mississippi State University, MS 39762
Voice: (662) 325-2756
Fax: (662) 325-8997
{bjw1, carver, vaughn}@cse.msstate.edu

Presenter: Byron J. Williams
SERP'06 - The 2006 International Conference on Software Engineering Research and Practice

Abstract
One certainty in software development is that all projects will have to deal with change. Being able to effectively handle proposed changes is crucial for allowing continued development of a software project to occur. In order to effectively manage the changes and their effects, developers must first assess the risks involved in making the change. To understand the risks, the project manager must determine how the change will affect not only the source code, but also the entire project. These risks may affect a project's schedule, budget, and quality factors. Risk assessment will help to determine if the desired change can be implemented into the system. This paper identifies risks associated with late changes to the software requirements. Late changes are those changes that occur after one cycle of the development process has been completed and a working version of the system exists. It is important to understand late changes, because late changes to requirements often result in the most cost to an ongoing development project both in time and money. In this paper we identify several key risks that must be addressed when dealing with late changes to requirements. Then we provide a discussion of techniques for handling those risks.

1. Introduction

Requirements engineering is the basis for software development. It is the foundation for the development of budgets, schedule, tests, and design [19]. Ideally, developers would like to create a concrete set of stable requirements; however, this scenario is seldom possible. Software, regardless of the precision of the development process or the depth of problem understanding by the developers, will change. Changes to the requirements can occur at any point in the development cycle, from the requirements phase all the way through several iterations into the maintenance phase. In fact, it is likely that more than half of a system's requirements will change before deployment [10]. Manny Lehman, in his extensive study of software systems, created the Laws of Software Evolution to describe common issues concerning software systems that change. He stated that software undergoes never-ending maintenance and development that is

driven by the difference between its current capability and what is required by the ever-changing environment [11]. There are many reasons why software must change to accommodate these differences. The environment change could require changes in protocols and standards necessary for communication with other systems. It could also require changes in hardware, the need for more efficient utilization of hardware resources, or changes in a user preference [2]. Because of these different sources of changes, a study of software change should not solely focus on the reasons why software changes, but also on ensuring that developers can handle risks associated with implementing those changes.

Managing customer requirements is one of the key problem areas in software system development and production [10]. To begin development, a set of requirements must be agreed upon by the developer and the customer. But, it is often impossible to make all the correct requirements and implementation decisions at the beginning and so, completeness is often not fully realized [18]. For software systems with a baseline set of requirements, a new requirement is often referred to as a change request. A change request should contain all the information necessary to modify the requirements to achieve the desired functionality [2]. Change management is one of the most important aspects for a successful software development project. Therefore, developers must have effective mechanisms to manage the change process [9]. Developers must also be aware of the risks associated with changes. These risks increase as development progresses. It has been hypothesized that, just as the cost of fixing defects increases later in the development process, a requirements change made later in the lifecycle will be more difficult and costly to implement [3].

The remainder of this paper is organized as follows. In Section 2, we identify several key risk areas associated with the propagation of change requests in a development project. In Section 3, we survey techniques aimed at handling risk. Finally, we provide and an overview of the risk outcomes as defined by Barry Boehm in Section 4.

2. Risk Assessment

Software development organizations suffer chronic problems from delayed and over budget projects. These problems are not found only during new development, but also during changes and enhancements to the requirements of existing projects. The difficulties software practitioners encounter when they face such changes can be minimized through risk assessment [16].

Boehm defines software development risk as the possibility of an unsatisfactory outcome. An "unsatisfactory outcome" is different for each stakeholder. For customers and developers, unsatisfactory outcomes are budget overruns and schedule slips; for users, they are products with the wrong functionality, or interface, performance, and reliability shortfalls; and for maintainers, they are poor quality software [4]. These three risk areas are addressed in this paper.

No matter the context, practitioners need well-defined approaches to handle risks. One initial aspect of risk assessment is to identify risk categories that can be applied to the specific types of projects developed by a given software development organization. Such categories found in the literature include product size and complexity risks, scheduling and timing risks, system functionality risks, requirements management risks, system quality assurance risks [5, 16]. Since there are various types of risks to assess, practitioners must decide which risks are most important for each project. This process is called risk prioritization. Risk prioritization is the process of identifying, analyzing, and ordering risks [4]. When developing software or

making changes to existing software requirements, it is important that the developers assess and understand the risks involved.

3. Related Work

Several techniques exist in the literature to help developers assess risks in change management. The first approach, *impact analysis*, is the "activity of identifying what to modify to accomplish a change, or identifying the potential consequences of change" [1]. Several aspects of impact analysis provide developers with insight into the amount of effort involved in making a change. Impact analysis involves identifying which parts of a program reference a variable or procedure that is related to the change. It also involves identifying objects and relationships among objects to determine which objects will be affected by the change [1]. Impact analysis helps the developer answer the question of what is required to do make the change while minimizing unexpected side affects. Although this minimization is not always feasible, impact analysis is designed to help developers gain control of the "ripple effects" associated with a change [15]. This paper identifies such "ripple effects" in terms of risk management.

A second approach to managing change is through *change classification*. Classifying changes to requirements is one way to help practitioners understand the nature of the changes. By accurately classifying change requests, a developer can utilize generally accepted heuristics to handle a particular type of change and predict the change's impact on the system [8]. In a study by Zowghi and Offen, the participants observed several benefits from using a classification approach for managing change. The benefits included:

1. Classifying change requests could be used as a means of controlling and managing changes.

2. Classification can help in assessing the impact of requirements changes in a reliable way.

3. Classification can promote a common understanding within the software development team of what the changes actually mean.

4. Classification can be used to identify risk associated with each change request or the group of changes.

5. Classification can help determine the change acceptability (i.e. reject or approve changes), hence supporting crucial decision making throughout software development lifecycle.

6. The requirements change categories and the subsequent constructs could be used to develop a multi dimensional matrix of all change requests. The particularly useful dimensions for more effective decision making are schedule, effort and reasons for changes. The matrix could be populated with all the change requests after categorization has been performed. For example, when project managers need to prioritize the change requests for implementation purposes, they can consult the matrix to identify the effort, schedule and the reasons for each change request for their assessment [20].

A third method, *causal analysis*, has been applied to change management to "identify implementation problems in a requirements change process, and to identify the causes of these problems" [7]. The authors of the causal analysis study wanted to identify the points in the change process where implementation problems occurred and identified the causes of the problems by tracing them back through the process. By performing causal analysis, an organization will be able to better understand why changes occur in the software that they are developing. Understanding the underlying causes of change will allow the organization to better manage changing requirements in future projects.

4. Change Implementation Risks

There are several risks identified in the literature that practitioners must be aware of when managing changing requirements. These risks include reduced quality, a requirements change snowball, and requirements change feasibility. Each of these risks is discussed in more detail in Sections 4.1-4.3, including some approaches to deal with the risks.

4.1. Reduced Quality

One major risk involved with implementing a change to a software system late in the product's lifecycle is the risk of reduced quality. Each time a change is introduced to existing software systems late in the lifecycle, there is a risk that the quality of the software system will degrade, meaning the system will be harder to maintain. If the design of the system is not fully understood by the developers, then the resulting system could suffer from degraded quality that does not match the ideal original design. Eick et al. studied code decay, a property that causes the software to be more difficult to change than it should be, that resulted from violations to the original design principles [6]. As a program is changed, each change to the structure of the system causes the system to become more complex. Quality measures must then be put in place to handle and reduce this complexity. Lehman's second law of software evolution stated it this way, "as a program is evolved, its complexity increases unless work is done to maintain or reduce it" [12]. Parnas called the phenomenon *software aging*. That is, changes to the software cause the structure of the system to degrade or age. The effect of software aging is that the software will become buggy, its performance will be reduced, and the development organization will lose customers to new products that perform better [13].

In order to mitigate the risk of reduced quality, prior to making a change, software developers must understand that change's impact on the quality of the software. The developers must know the design constraints and commit to maintaining the original design constructs, such as the coupling of the modules, the level abstraction, and the principle of separation of concerns [13]. The better a developer understands the design of the system, the more likely he will be to implement a change without degrading the quality of the system.

4.2. Requirements Change "Snowball"

When faced with a single change request that requires a developer to add new functionality to a system or change an existing function, there is always the risk of even the smallest requirement change "snowball" into a large project. This snowball effect can have several causes. When changing a system requirement, that requirement will likely have dependencies on other requirements, and other requirements will in turn depend on that requirement. When this is the case, a requirement change is not isolated to a single requirement.

It can turn into a large project while dealing with the changes that need to be made to satisfy the requirement dependencies. There is also the case that changing a single requirement may introduce a conflict with an existing requirement. In this case, both requirements must be modified so that the new functionality can coexist with the other requirements. If any principle software functions are affected by a single requirement change, then those functions must be changed to ensure the change is successful. Another issue is that of performance. When a single change is made that reduces a quality factor such as CPU performance, then this change may require the hardware components or the system allocation of processes to change [17].

Schneidewind points out that there are numerous risks associated with changing software requirements. Many of these bring about the snowball effect when making a software change. He recommends performing dependency checks among risk factors to determine how a change will affect the various aspects of a system. Once this check is completed, a systematic planned approach can be developed to handle dependencies and properly estimate the cost and schedule requirements of the change [17].

4.3. Requirement Change Feasibility

A project manager must be able to determine the feasibility of a change request based on the cost and budget constraints of the project. Many software development projects have some form of change control board that makes the final decision on the status and priority of a change request. This board determines whether a change should be implemented into a system and prioritizes those changes. The change control board then presents an engineering change order that describes the change, the constraints, and the criteria for review and audit [14]. When the determination is made that the change is required, the project manager must assess the costs associated with the change. This assessment includes creating a schedule and determining the manpower requirements for the change in order to provide an estimate for change cost. This analysis should be performed in a similar manner to the procedures performed at the onset of the project to determine costs and scheduling. If a particular cost estimation model is used to determine the initial cost of the software development model, such as COCOMO or Function Point Analysis, then the same model should be used to calculate the costs and budget requirements of change implementation [10, 19]. These costs must then be presented to the customer for acceptance.

The risks associated with any new software development project also exist for the change implementation. The project manager now has to control any factors that would cause the change implementation to be over budget or to exceed the schedule. The risks described in section 3.2 must be taken into account prior to setting a cost and schedule for the work.

5. Summary

Risk assessment in changing requirements of existing systems is an important aspect of producing the desired results of a change. Software developers must understand the risks involved and develop ways to mitigate those risks while maintaining budget and schedule constraints. This can be a difficult task for anyone. What practitioners need to understand is that being aware of the risks involved is only the beginning. A consistent risk management and assessment process must be applied to ensure that risks are handled for every modification requested for a software system. In order to obtain this consistency, project managers must

address the risks involved at every phase of development and incorporate risk assessment techniques into the culture of the organization.

Acknowledgements

REFERENCES

[1] R.S. Arnold and S.A. Bohner, "Impact Analysis-Towards a Framework for Comparison," *Proceedings of the Conference on Software Maintenance* Montreal, Que. , 1993, pp. 292-301.

[2] L. Arthur, *Software Evolution: The Software Maintenance Challenge*, John Wiley & Sons, Toronto, Canada, 1988.

[3] B. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[4] B.W. Boehm, "Software Risk Management: Principles and Practices," *IEEE Software*, vol. 8, no. 1, 1991, pp. 32-41.

[5] Y. Chuk, "A Quantitative Methodology for Software Risk Control," *IEEE International Conference on Systems Management and Cybernetics*, San Antonio, TX 1994, pp. 2015-2020.

[6] S.G. Eick, T.L. Graves, A.F. Karr, J.S. Marron, and A. Mockus, "Does Code Decay? Assessing the Evidence from Change Management Data," *IEEE Transactions on Software Engineering*, vol. 27, no. 1, 2001, pp. 1-12.

[7] K. El Emam, D. Holtje, and N.H. Madhavji, "Causal Analysis of the Requirements Change Process for a Large System," *Proceedings of the International Conference on Software Maintenance*, Bari, 1997, pp. 214-221.

[8] S.D.P. Harker, K.D. Eason, and J.E. Dobson, "The Change and Evolution of Requirements as a Challenge to the Practice of Software Engineering," *Proceedings of the IEEE International Symposium on Requirements Engineering*, San Diego, CA, 1993, pp. 266-272.

[9] C. Jones, "Software Change Management," *Computer*, vol. 29, no. 2, 1996, pp. 80-82.

[10] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*, John Wiley & Sons, Chichester, West Sussex, England, 1998.

[11] M.M. Lehman and L. Belady, *Software Evolution - Processes of Software Change*, Academic Press, London, 1985.

[12] M.M. Lehman, J.F. Ramil, P.D. Wernick, D.E. Perry, and W.M. Turski, "Metrics and Laws of Software Evolution-the Nineties View," *Proceedings of the Fourth International Software Metrics Symposium*, Albuquerque, NM 1997, pp. 20-32.

[13] D.L. Parnas, "Software Aging," *Proceedings of the Sixteenth International Conference on Software Engineering*, Sorrento 1994, pp. 279-287.

[14] R. Pressman, *Software Engineering: A Practitioner's Approach*, 6th ed, McGraw Hill, New York, NY, 2005.

[15] J.P. Queille, J.F. Voidrot, N. Wilde, and M. Munro, "The Impact Analysis Task in Software Maintenance: A Model and a Case Study," *Proceedings of the International Conference on Software Maintenance*, Victoria, BC 1994, pp. 234-242.

[16] J. Ropponen and K. Lyytinen, "Components of Software Development Risk: How to Address Them? A Project Manager Survey," *IEEE Transactions on Software Engineering*, vol. 26, no. 2, 2000, pp. 98-112.

[17] N.F. Schneidewind, "Investigation of the Risk to Software Reliability and Maintainability of Requirements Changes," *Proceedings of the IEEE International Conference on Software Maintenance*, Florence, 2001, pp. 127-136.

[18] J. Siddiqi, "Requirement Engineering: The Emerging Wisdom," *IEEE Software*, vol. 13, no. 2, 1996, pp. 15.

[19] G. Stark, A. Skillicorn, and R. Ameele, "An Examination of the Effects of Requirements Changes on Software Releases " *Crosstalk: The Journal of Defense Software Engineering*, vol. 11, no. 12, 1998, pp. 11-16.

[20] D. Zowghi and R. Offen, "A Logical Framework for Modeling and Reasoning About the Evolution of Requirements," *Proceedings of the Third IEEE International Symposium on Requirements Engineering*, Annapolis, MD, 1997, pp. 247-257.