Software Maintenance Risk Management Process – A Case Study

Vinicius Miana[1], Calebe Bianchini[1], Selma Melnikoff[2] and Marcelo Martins[2]
[1]FCI, Mackenzie University, São Paulo, SP, Brasil
[2]POLI, Universidade de São Paulo, São Paulo, SP, Brasil

Abstract - *Software Maintenance risk management differs in many aspects from software development risk management. Although it is the longest and the riskier phase on the software life cycle, differently from software development where many processes and models were established, very few processes have been developed to deal with software maintenance. Because it deals with systems that are already in production, software maintenance presents much more sources of risks. In this paper, we go through the software maintenance process identifying the main sources of risks and defining a process that can help mitigate those risks. Finally, we present a case study where this process was applied and some of the results are shown.*

Keywords: software evolution, risk management

1 Introduction

Software maintenance encompasses all activities enacted after software deployment that aim to modify it [1]. Many studies have shown that the costs associated with software maintenance have grown as time goes by [2]. Since it is not as attractive as new software development, software maintenance was not studied and researched in the same depth as development and therefore very few models were developed to deal with it [3].

In this work a software maintenance risk management model is developed using the following work as reference: Bennet [1], Polo [2] and Webster [3]. This paper is divided in 6 sections. The first one introduces the subject. On the second section software maintenance and its state of the art is presented. The third section presents software risk management and its existing models. On the fourth section a proposal for software maintenance risk management is presented. The fifth section presents a case study and finally the sixth section presents the conclusions of this work.

2 Software Maintenance

Software maintenance is the modification of a software product after its deployment with the objective of correcting errors, enhancing performance and other attributes or adapting the product to changes in the environment IEEE [4]. Pressman [5] categorized software maintenance into four types:

- Adaptive: changes in software environment;
- Perfective: new user requirements;
- Corrective: error correction;
- Preventive: to prevent problems in the future.

Pressman [5] also believes that software maintenance should be viewed from 3 main perspectives:

- The activities involved in software maintenance and software engineering impact on the efficacy these activities;
- The costs related to software maintenance;
- Problems that usually happen during software maintenance.

Regarding software maintenance activities, Pressman [5] distinguishes structured software maintenance and unstructured software maintenance and highlights the following activities on structured software maintenance:

- Design evaluation;
- Maintenance approach planning;
- Design change;
- Re-coding;

Revision: which may imply on re-changing the design and re-coding depending on whether the desired results are met or not.

Regarding software maintenance costs, there is a concern with growing costs in various studies [2], however it was observed that intangible costs are often not taken into account. These are the main intangible costs in software maintenance [5]:

- Customer dissatisfaction with unmet requirements or on the delay on meeting them;
- Quality reduction as result of some changes introducing latent errors;
- Problems caused in a development effort when programmers are forced to stop what they were doing to work on software maintenance activities.

Regarding usually found problems in software maintenance, the following arises [5]:

- Impossibility to trace software evolution: changes not documented;

- Impossibility to trace the process in which the software was created;
- Difficulty to understand code written by someone else;
- Difficulty in finding the code's author for clearing doubts;
- Inexistent or bad quality documentation;
- Difficulty in changing the software: software design does not take into account possible future changes.

The considerations mentioned earlier are very important when we are defining maintainability, which is the easiness that software can be understood, corrected, adapted and/or enhanced, and it is influenced by the following factors [5]:

- Skilled personnel availability;
- Capacity to understand system structure;
- Easiness to manipulate the system;
- Use of standard programming languages;
- Test case availability;
- Code debugging mechanisms availability;
- Availability of adequate environment for conducting maintenance.

Bennett [1] introduces a software maintenance life cycle model, which allows distinguishing software regarding its age and maintainability. This model is called a software-staged model and it divides software product life cycle into 5 different stages counted from its deployment:

- Initial Development: in this stage software is exactly as it was and it was deployed. Software will be in this staged until its first maintenance is executed;
- Evolution: in this stage the application is adapted to constant changes in user requirements and operational environment;
- Service: as time goes by, changes in software corrupts the initial architecture and members of the original development team leave until you get to a point where making changes becomes so hard, either due to lack of knowledge in the current team or due to the need of large architectural changes, that is no longer possible to evolve the software. In this stage only small tactical changes are undertaken;
- Phase-Out: in this stage no changes are made in the software. Users must work around software problems;
- Close-Down: in this stage, the software is disconnected and users are directed to a substitute.

3 Software Risk Management

Before starting with software risk management, we shall define software risk: Software Risk is a measure of the probability of loss and its impact related to a software project, process or product [6].

Risk Management is a general procedure for resolving risk and has two main components [7]:

- Risk Assessment defines risk by identifying hazards, evaluating their potential effects and the likelihood of their occurrence.
- Risk Control is the process of developing risk resolution plans, monitoring risk status, implementing risk resolution plan and correcting deviations from the plan [6].

The risk management process can be divided into 6 elements, three related to risk assessment: identification, analysis and prioritization and three related to the control: planning, monitoring and resolution of risks [6]. According to Boehm [7], risk management can be classified in the following way:

- Risk assessment
  - ◦ Risk identification
  - ◦ Risk analysis
  - ◦ Risk prioritization
- Risk Control
  - ◦ Risk Management Planning
  - ◦ Risk resolution
  - ◦ Risk monitoring

The risk identification process encompasses activities that lead to the identification of the hazards that may threat the software product, process or project. Software Risk Identification may use methods involving one or more of the listed below [6]:

- Checklists – use of lists as a reminder of possible risk areas;
- Interviews – use of group interview session where people may talk about their concerns, doubts, problems and uncertainties related to the software;
- Meetings – use of periodic meetings to discuss project risks;
- Revision – use of plan, procedure and work products review;
- Forms – use of standard risk management form to input routinely found risks;
- Survey – use of questionnaires as a faster way than interviewing people about their perceived risks;
- Working Group – use of brainstorming, meditation, modeling, simulation and other group activities.

The risk analysis process consists on quantify each hazard identified on the Risk Identification Process by calculating its occurrence probability and impact. By using a probability/impact matrix, risks can be classified as critical, high, moderate, low or negligible [3]. This process encompasses the following activities: grouping similar and related hazards, determining which may have an impact on risk, determining sources of risk, using risk analysis techniques and tools, estimating risk exposure, evaluating risk against criteria, ranking risk according to its severity [6].

The risk planning process consists of all activities and methods used to develop risk resolution alternatives [8]. This process encompasses the following activities: development of risk scenarios for high-severity risks, development of risk resolution alternatives, selection of risk resolution approach, development of risk resolution action plan, and establishment of variables to be monitored with threshold values for warning [6].

The risk monitoring process consists of activities of risk measurement and indicator tracking, which may indicate that a risk resolution plan must be executed. Tracking indicators may anticipate the loss occurrence, giving more alternatives to mitigation [6].

The risk resolution process consists of activities that aim to reduce risk to an acceptable level. The activities in this process include: response to a notification of triggering event, execution of a risk resolution plan, report of progress against the plan, and correction of deviations from the plan [6].

4 Software Maintenance Risk Management

Using software maintenance and software risk management concepts, we developed a software maintenance risk management model. As stated in many references ([1],[2].[3], [7], [8], [9], [10]), most of software maintenance environments present some factors that increase risks in software maintenance. These factors were used as premises when developing this model and are listed below:

- The deployment of risk management process will be in a environment with no previous risk management culture;
- The software to be maintained were developed by other people;
- Documentation either does not exist or is outdated or has bad quality;
- Languages and platforms used in many modules are old.

Given these premises, we used Hall [6] risk management model and added activities proposed by Charette [8] for cultural adjustment of personnel and good software maintenance practices proposed by Weber [11]. The use of the premises was important to create a process that can be applied in an organization that already performs software maintenance activities and needs to have its software maintenance risk management process improved. The work of Hall [6], Charette [8] and Weber [11] was integrated into one single model after a careful review of their proposal. Additional elements were added based on author’s experience and redundancies eliminated. Finally, the proposed model was checked against the IEEE Std 1219-1998 Standard [4] which defines software maintenance, to make sure that correct naming was used and that the process would conform to the standard.

Software Maintenance Risk Management should fit into the Software Maintenance Process [4], with the following changes:

- Team preparation: through training and mentoring, the team will be better prepared to find and communicate risk found in their activities;
- Communication: risk communication strategies should be established and periodic meetings should take place to evaluate those risks;
- Problem Classification: the process of receiving new functionality and bug fixing requests should be redesigned to take into consideration that the risk involved in these activities when prioritizing them;
- Documentation: a task-force should be defined to produce, enhance and update all maintained software documentation. The documentation activities should be prioritized according to the degree of changes made in each software and/or module. Also, when modifying any part of the code, the maintenance team should enhance and correct/update its existing documentation;
- Tests: an automated test policy should be established, this tests would expedite the maintenance process and ensure that any corrections and changes made on the software did not cause an error somewhere else.

The changes mentioned earlier were implemented first, by adding extra activities in the following IEEE Std 1219-1998 [4] Process:

4.1 Problem/modification identification, classification, and prioritization

- For every problem/modification identified, the risks associated to them should be evaluated. It should be considered the risk of doing the change against the risk of not doing it. When not performing a modification or fixing a problem, we have a risk of loosing customer base by not attending some desired/expected requirements. On the other hand, some modifications may bring distortions to the system architecture making it more difficult to perform future maintenance and reducing system life. Another risk that must be taken into consideration is an excessive maintenance cost that must be compared with the cost of replacing the solution. When prioritizing modifications, the risk involved in each one of them must be used as reference when defining what must be done immediately, what will be postponed and what will not be done;
- A mitigation plan must be established and the team should be prepared to act if a loss occurs;
- To identify risks, we suggest using the taxonomy proposed by Webster [3].

4.2 Analysis, Design, Implementation

- When performing analysis, design and implementation activities, the team must pay attention to the risks already listed in previous activities and also to new ones not previously identified. All identified risks should be entered in the risk matrix and monitored by the team that is performing the maintenance;
- During these activities, updating and enhancing documentation should be done as a measure to reduce future maintenance risk.
- System Test and Acceptance Test:
- Tests should be automated for faster and more reliable execution.

Then there were few activities that would not fit into the IEEE Std 1219-1998 [4] proposed phases and they should be executed as specified below:

4.3 Team Preparation

- This phase prepares the team to changes in their daily activities, introducing them to risk management paradigms;
- In this phase, training in risk and in the proposed process should be given to the whole team.

4.4 Documentation

Documentation can be the most important ally or enemy when maintaining a legacy system. Due to that, the proposed process has documentation activities in the analysis, design and implementation, but also a documentation taskforce. This taskforce should perform search, organization, consolidation, complementation and correction activities on the existing documentation. These activities even though not directly related to risk management they are verify important for providing resources that will allow a more precise assessment of maintenance risks.

5 Case Study

In order to test the proposed model, a case study was developed. It was chosen to apply it in an legacy university crm system that has a web interface and as it was complex enough to have maintenance issues and simple enough to have results easily monitored.

The application was developed using the JAVA language and was very recent. As it had tough deadlines and integration requirements with other systems, some developed in Natural/Adabas which are old and have many documentation issues, the project was deployed very fast and faced constantly changing requirements moving it fast to the Evolution stage. Complying with the activities proposed in the process was very time consuming and we face challenges both from user expectations and developer hastiness. With weekly deployments, sometimes more problems emerged when a simple change was performed. It was hard convincing developers to apply the process and we decided to count new bugs per week and use it as metric to show progress. Since the system was recently developed, we did not face any challenges with use of old technologies or corruption of architecture coherence. With bug tracking system in place and version control using cvs, we could easily recover the statistics before the process was implemented and be able to show a reduction in new bugs per week with few weeks of implementation.

Regarding the implementation of the proposed activities, we made the following findings:

5.1 Problem/modification identification, classification, and prioritization

Associating risk for every problem/modification identified was easy in most cases. At the end of the week, all problems and changes requests were discussed and maintenance team evaluated the risk. Webster's taxonomy was used and helped raising the right issues and making the meeting more productive [3]. This extra task didn't make the meeting much longer than usual and helped bringing consciousness of the impact of changes to the development team, making them more careful. In general terms, we could also say that better decision were made in the Problem/modification identification, classification, and prioritization activities.

Before, starting working on a new release, the code was tagged in the source control software, allowing going back as a mitigation step. Also, to prevent problems when larger architecture changes were made, the whole cvs tree was backed up. During this study, sometimes it was necessary to move back to the previous compiled version; however we didn't face situations where we had to roll back source code.

5.2 Analysis, Design, Implementation

Finding risks during analysis, design and implementation activities was not very successful in the beginning, as the team was not used to do that and differently than in a meeting there is no driver of the discussion, developers are working on the own. We perceived that they were afraid to present the risks as they felt as showing weakness on their work. It took strong persuasion to improve that and we still feel it is not working as well as it should.

Documentation activities also were hard to implement, developers didn't like doing that and were always in a hurry, not having time to document. Regular documentation activities during Analysis, Design and Implementation were only made after few weeks of micro-managing this topic.

Choosing correct tools and environments could help the team not only writing down analysis, design and implementation documents, but also could generate automatically some architecture design and source-code [12]. Furthermore, these tools could help finding and reusing components already developed and available in a common repository [13].

5.3 System Test and Acceptance Test

Slowly, Junit tests were developed and helped a lot during System tests.

5.4 Team Preparation

A PowerPoint presenting this process and some literature regarding risk was presented to the maintenance team to prepare them to the changes in the process. After that, these changes were discussed individually with each member of the maintenance team to make sure they really understood how the team would perform from that point.

5.5 Documentation

Due to lack of resources, we were not able to implement the documentation taskforce.

After analysing the data, we found that the quantitative results were inconclusive. Many variables may have had an effect on bug reduction, including that as time passed, requirements became better known and more stable. Nevertheless, the experience of implementing this process gave to the developer team a greater level of conciousness regarding maintainance risks. That led to more carefully designed, documented and coded software. The weekly release meetings after risk was brought to the table made developers more careful before making bold movements of, for instance, changing a database structure.

From that experience, we perceive the following challenges, when implementing the proposed process:

Change resistance:

- In many cases, the additional activities proposed by the process may be seen as bureaucratic and pointless, making necessary a strong convincing work to make people adopt this new way of working;
- Aiming to make this argument stronger, metrics that allow monitoring the progress and seeing the benefits to clients, team and company when adopting the process should be implemented;
- Management must be convinced before anyone else to adopt risk management as a priority. An implementation of risk management process should not be started without total support from management.

Lack of skills in the team:

- The adoption of risk management demands higher skills than what is usually found in maintenance teams. In most cases, this problem can be reduced by the proposed training, however many times it involves more basic education in software engineering matters;
- The adoption of a process implies in discipline and skills. When there are no previous processes in place, this may mean a big leap in skills needed;
- As it is not as attractive as new software development, maintenance most often has less experienced professionals.

Difficulties to access information:

- Documentation activities present a enormous challenge, since, in many cases, there is no documentation or it is outdated and the team that developed the system is no long available;
- To gather this information it is often necessary to read source-code which is often obscured by many patches brought by the changes in the software;
- Users themselves could be a great source of information, since they supposedly know well the business rules that were automated by the system.

To face to those challenges, many measures must be adopted, the study of those measures is the objective of future work in this area. One approach to deal with this problem with lower overhead might be documenting directly in the source-code using annotations [14].

6 Conclusions

During the development of this work we came across significant differences between software maintenance and software development. These differences make risk management also very distinct when dealing with software maintenance versus software development. Even though maintenance is responsible for 90% of software costs in its life cycle, very few studies were developed on software maintenance risk management. Maintenance process and practices were studied as way of analyzing its risk factors, which helped adding risk management practices in the process. Similarly current software development risk management work and software maintenance work were studied. All this information was compiled and helped generating the proposed software maintenance risk management process.

During the development of this process and on its trial in the case study, many challenges were found and they were highlighted in this paper. It was verified that the most impacting risk factors in software maintenance are the lack of skills in maintenance personnel and the lack of documentation. The proposed process aims to mitigate, at least partially, these risks. Specific risk mitigating measures that should be taken still rely on management experiences and can not be defined in a generic risk management process, as the one we proposed.

As we verified in our case study and on the literature, the implementation of risk management process as this has a great impact on diminishing problems related to schedule, costs and meeting customer needs in software maintenance activities. However, for better results, it is required to start taking maintenance in consideration from the first conceptual sketch of a new system and during the software entire life-cycle.

As future work, we hope to enhance the software risk management process, identify most common risks, metrics to help identifying them as soon as possible and include in the maintenance process activities that help mitigating those risks. On another research line, a very interesting research subject would be the definition of characteristics and metrics that could be used to evaluate software regarding its maintainability and its maintenance risk.

7 References

[1] BENNETT , K; RAJLICH, V. Software maintenance and evolution: a roadmap. Proceedings of the Conference on The Future of Software Engineering table of contents. Limerick, Ireland. p. 73 – 87. ISBN:1-58113-253-0. Publisher ACM Press New York, NY, USA, 2000.

[2] POLO, M. Advances in Software Maintenance Management: Technologies and Solutions. Idea Group Publishing. Loughborough, UK. 2002.

[3] WEBSTER, K. et al. A Risk Taxonomy Proposal for Software Maintenance. 21st IEEE International Conference on Software Maintenance (ICSM'05). p. 453-461, Budapest, 2005.

[4] IEEE. IEEE Std 1219-1998: Standard for Software Maintenance. Los Alamitos, CA USA. IEEE Computer Society Press, 1998.

[5] PRESSMAN, Roger. Software Engineering – A Practicioner’s Approach. London, England. Mc-Graw Hill Book Company, 6th Edition. 2004.

[6] HALL, E. M. Managing risk : methods for software systems development. Addison-Wesley , 5th Edition, ISBN 0201255928. Boston, 2002.

[7] BOEHM, B. IEEE Tutorial on Software Risk Management. New York, NY USA. IEEE Computer Society Press, 1989

[8] CHARETTE, R.N; ADAMS, K.M; WHITE, M.B. Managing risk in software maintenance. IEEE – Software. V. 14 N. 3, p. 43-50. May/June, 1997.

[9] BUCLEY, J. et al. Towards a taxonomy of software change. Journal of Software Maintenance: Research and Practice. V. 1 – 389. John Wiley & Sons, Ltd., 2003.

[10] RUIZ, F. et al. Utilización de Investigación-Acción en la Definición de un Entorno para la Gestión del Proceso de Mantenimiento del Software. In: 1er. Workshop en: Métodos de Investigación y Fundamentos Filosóficos en Ingeniería del Software y Sistemas de Información. (MIFISIS'2002). Madrid, 2002.

[11] WEBER, R. et al. Fit for Change: Steps towards Effective Software Maintenance. 21st IEEE International Conference on Software Maintenance (ICSM'05). p. 26-33, Budapest, 2005.

[12] BEZERRA, V. et al. Designing object oriented systems using stereotypes and patterns. Proceeding of IADIS WWW/Internet 2006. 1ed.Murcia: IADIS, 2006, v. 1, p. 162-166. 2006.

[13] LUCRÉDIO, D. et al. ORION – A Component-Based Software Engineering Environment. The Journal of Object Technology (JOT), v. 3, n.4, p. 51, 2004. URL : http://dx.doi.org/10.5381/jot.2004.3.4.a4

[14] BEZERRA, V. et al. Requirements oriented programming in a web- service architecture. IADIS www/Internet Proceedings. Lisboa: IADIS, 2010. v. 1. p. 287-292.