Maintenance software processes for web 2.0 based learning management systems

Tomáš Poklemba*, Igor Sivý* and Zdeněk Havlice*
* Technical University of Košice, Department of Computer and Informatics, Košice, Slovakia
Tomas.Poklemba@tuke.sk, Igor.Sivy@tuke.sk, Zdenek.Havlice@tuke.sk

***Abstract*— As we know all software products require maintenance and support. After observing time requirements for maintenance and all needed innovations and changes I realize that cost for those activities is somewhere around half of all bugged for software product. The more years is software used and alive the more time and costs are required. Linear dependency between realized changes and costs is changing into exponential dependency as the years go. Bad and unwanted structural changes at the very beginning of software design could overcharge software maintenance. It is nothing uncommon that maintenance of poorly or wrong designed software products (for example LMS) can achieve point when cost of new development is lover than actual cost of maintenance. In this contribution is presented a proposal of effective software maintenance adverted for web 2.0 based LMS system. Our proposal exploits what should be the proper way of software design, what are the common mistakes, what are our personal experiences and finally how to gain the low cost software maintenance.**

I. INTRODUCTION

Almost all software developers would agree that shelve of maintenance is rowing future time and financial cost or could lead into software absolute crash. There are many theories and model driven architectures as an example of proper and solid maintenance. The real world of software development especially in smaller companies usually works slightly different as their model entities. Most of the code changes in real scenario of software maintenance include just smaller complements or data changes. With missing documentation and models from the beginning of software development all other changes and transformation are usually moved as potentially dangerous. Unfortunately, nobody could develop stable and static algorithms from the begging models. The reason is that alive software is changing its data structure and algorithms based on user requirements.

The main purpose of this contribution is to advertise importance of software maintenances, description of proper maintenance solution and finally explanation from know-how driven from our real scenarios.

I. CATEGORIZATION OF MAINTENANCE

Some authors such as [Phillips 2000; Balzert 2001] explain or define maintenance by listing possible categories of maintenance and defining those. Examples for such maintenance categories are:

- corrective maintenance, e.g. the removal of errors in the code
- adaptive maintenance, e.g. the adjustment of software for a new operating system
- perfective maintenance, e.g. the modification of the code to improve performance.

Although the above mentioned authors agree on the number and names of the categories, they differ with regard to the definition and explanations of some of them.

To demonstrate the variety and the related naming problem I will briefly describe the different definitions and explanations. Where a general agreement about certain terms or their core meaning exists, I will continue using them.

1. **Corrective Maintenance**: Event driven, reactive and partly unscheduled modification of a software product to correct discovered faults to keep a system operational.
2. **Adaptive Maintenance:** Event driven modification of the software product due to changed or changing environment or requirements
3. **Perfective & Preventive Maintenance**: Quality driven modification of the software product to improve efficiency, performance and maintainability and preventing problems in the future.

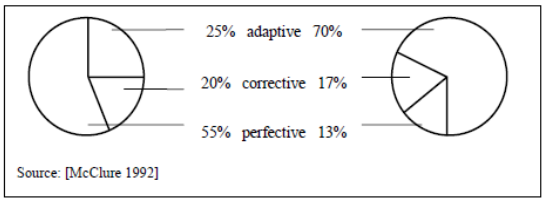

Figure 1.

II. ANALYSIS

As an proper way and stable solution fo V-Model

The V-model is a software development process model developed by the German Federal Ministry of Defence US. The V-model was developed with the focus on the following applications:

978-1-4577-0052-1/11/$26.00 ©2011 IEEE

- basis for contracts
- instruction for system development with detailed descriptions of the activities and documents
- communication basis with the feature of universal validity and project independence

This is the major difference to the Waterfall model. The V-model can be seen as a bent Waterfall model, enabling feedback to corresponding phases during the process. However, the V-model still belongs to the category of linear sequential models because one phase needs to be completed before the process moves on to the next phase. In addition to the needs of the organization and the users, which are transformed into requirements for the software product, a major aspect regarding the input is the detailed specification of the activities.

The well-known V-form of the V-model can be seen in the sub model of system development. The other sub models are Quality Assurance, Configuration Management and Project Management. The nine phases of System Development, as illustrated in , are:

- SD1: System Requirements Analysis: Requirements are analyzed, a description for the system and its environment is established and a risk analysis is realised.
- SD2: System Design: The system is divided into segments.
- SD3: Software/Hardware Analysis: Technical requirements are described in more detail.
- SD4: Software Design (rough): The software architecture is designed.
- SD5: Software Design (detailed): The detailed design is specified.
- SD6: Implementation: Software components are coded.
- SD7:Software Integration: Software components are verified.
- SD8: System Integration: Components are integrated and validated; the components are integrated into segments and the segments are afterwards integrated into the system.
- SD9: Transition to Utilization: The finished system is installed at the point of operation.

In addition to early fixed requirements and a linear sequence of activities, test cases and scenarios are important inputs for later phases which are used for verification and validation of the implementation and the system requirements in the second half of the model.

The final output is an operational system with all functions and a complete and detailed documentation.
Roles The main roles in the V-model are project leader, quality assurance manager, configuration management representative and controller. Other roles are the usual, candidates like system analysts, programmers, etc. who are in involved in software development.

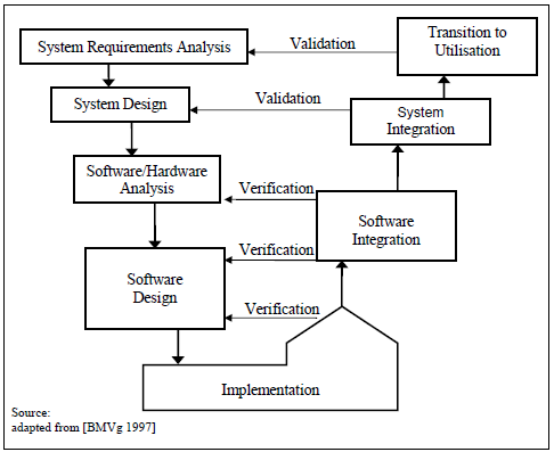

Figure 2.

Maintenance in the V-Model

The V-model with regard to maintenance:

- reduction in the number of maintenance cases as a result of improved product quality
- decrease in maintenance effort via easy comprehensibility and the existence of an adequate software documentation.

It is right to say that the V-model comprises all necessary activities to make use of the above listed advantages. However, it is a complex model and only suited for large
projects. Furthermore it is not always possible to transfer the ideas of a perfect world model to the real world. In contrast to the previously described Waterfall model, maintenance is not explicitly embedded in this model; it only comes implicitly after transition to utilisation. What is said about maintenance is that maintenance itself is reduced through proper development. Activities for maintenance after product delivery are not described. Thus the V-model is only suited to reduce the need for maintenance but not for maintenance projects.

Maintenance Management

Maintenance management is attached to the standard as an appendix. It includes, amongst others, management activities which need to be performed to organise the process. This belongs to the informative part of the standard. Determine maintenance effort Comment
In a recent paper [Sneed 2003] repeats his opinion that the cost of maintenance is better to estimate than the cost of new development. Although the likely return of reengineered software is smaller than of newly developed products, so is the risk. The return on investment can be much better calculated as there is less uncertainty, which is a huge problem of new developments. However, this is only valid for certain reengineering projects. With maintenance in the long run it is hardly possible to know what errors will occur or how the environment changes. Nevertheless, it seems that many authors believe that the estimation of costs with a horizon of one year is possible.

The aim is to determine the cost for maintenance for a time horizon of at least one year, i.e. the next budget round.

Required are figures about completed maintenance projects. This includes not only financial data but also data about elements that needed modification as well as the source and reasons for modification requests.

Using figures from the identification and analysis stages, especially from the 'cost analysis/estimation', and data about actual costs, evaluate the effectiveness of preliminary cost estimation. Extrapolation of current maintenance efforts assists the planning of future maintenance needs which can be used for the budget. mAdditionally it should be determined what the costs would be if no maintenance is done, i.e. identify the benefit or value of maintenance.

The results are current and future efforts/needs for maintenance and a budget draft.

Roles involved are a project reviewer to evaluate the project data and an accountant to analyze the financial data. Supporting tools are a database analysis tool to extract and analyze data about the system and a calculation application for financial planning. Embedding in Process Models Linear models and XP do not cover this aspect. RUP contains the evaluation of projects with the aim to plan for the future but not financial planning.

Statistical Data

Current figures about the importance of maintenance are not available. mSurveys from Lientz and Swanson conducted in the 1970s and early 1980s are still quoted in more recent books as for example [Balzert 2001] and [Sommerville 2001]. Current figures are not available. [Lehner 1999] writes that it is often assumed how much of the software budget is used for maintenance. And it still needs to be proven that the percentage of maintenance of the budget increases as suggested by other authors. However, the statements of some interviewed software engineers made demonstrate or indicate that at most companies, there is no separation between development and maintenance in their budgets. Most companies can only assume how much is spent on maintenance if they are interested in this at all. This leads to the next aspect.

Awareness of Maintenance

Maintenance is not sufficiently observed or regarded as important in industry. ,It is surprising that many companies do not have a dedicated position (or cost centre) for ,maintenance within otherwise detailed budgets. Furthermore, only one some companies has a clear definition of what maintenance is. Considering these facts, it is not surprising that many companies do not have a dedicated process for maintenance. At the first branch of one ,company they presented an abstract from the company's handbook showing a maintenance process description, while at another branch they said that maintenance was not included in that handbook. Statements like "we do not do any maintenance" round off the picture. In the literature, maintenance is said to be so important that it is not suitable as a training ground. Some companies go to the other extreme by explicitly using maintenance tasks as a training ground for new staff to become acquainted with the ,software. Depending on the tasks this may be useful, but it is the most obvious ,difference between theory and practice. Another issue is whether maintenance is considered as an unpopular area to work in. Here academics and practitioners agree, partly saying that long maintenance projects are a kind of punishment, but not much is being done to improve this image. Thus, it is necessary to promote the importance of maintenance in industry and within organizations. Companies as well as their employees need to be aware of the significance of maintenance and have to understand and believe that working in the
field of maintenance can be as challenging as working in development.

Learning management Systems and their maintenance

Learning management systems are software systems used for on-line and independent education. From the lookup of software maintenance they are not markedly different as standard software processes. LMS are distributed in following areas :

- General LMS– are mostly used to cover education in many individual and autonomous can cover education in many fields
- Problem-oriented – are supporting tools and functionality for education of a particular subject. Generally study of other subject can be problematic.
- Combination of both general and problem-oriented [4].

Problems in this distribution and maintenance of LMS are eliminated with creation of model driven architecture, which transforms standard learning management systems from the pint of maintenance into stable software solution.

Tool support for Maintenance

There is a gap between the tool support suggested in literature and the actual tool usage in practice.
According to software engineering literature, general software tool support is very important and can be the crucial factor for success in maintenance projects. Many tasks such as refactoring can be partly automated using adequate tools. Good tool support is also mentioned as one reason why reengineering is getting better and more efficient. One definition of reengineering even included automation through tools as one aspect. In contrast to that view are the results from the interviews. Only one company, IBM, named tool support as an important factor influencing the quality and efficiency of its process. This company provides services for many customers. The use of tools is efficient because IBM has the critical mass to justify the investments. The others said that most of them tried tools at some stage but decided not to use them. The reasons are that too much effort is required to adapt (to) the tools, the investments are simply to high, and that no tools for their software exist. Only for simple tasks are tools such as
error report databases and line debuggers used.

III. Solution and Results

As it is described in previous article of tools for software maintenance using refactoring and tracking code changes is very useful. Our experiences will greatly confirm that. We are using source control software to monitor code changes and time management from the very begging of development. By creating statistics and analyzing all necessary inputs we discovered really interesting achievement. Approximately 30 % of time spend on development is used to maintenance designed software processes. One of the most critical maintenance plans are applied to windows services. The reasin is that they are independent and not user directly controlled. Services are mostly doing their work through no load time at nights. Things like sending emails sorting data or advertise uncommon states are their main features. Developer has narrow abilities to test their functionality through enormous code changes. Models and their dependency could solve this problem. Our model driven architecture connected with source control software saved our applications many times. One of the explanations is to save Meta data connections between classes, services or important code structures. Those meta data will later warn developer that realized changes are affecting more code structures than just changes. Our solution to handle all maintenance properly and keep all developed software processes stable is shown on next figure :

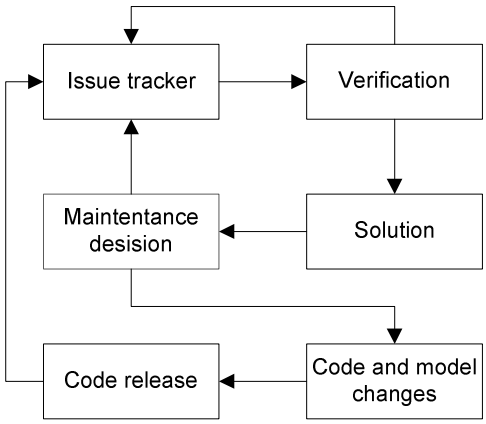

Figure 3.

Conclusion

This contribution is an explanation and introduction into most critical parts of software maintenance. We base all articles on our experience of maintaining LMS system with over ten thousand users. Software deployment and code changes as well as maintenance could not be made through critical times. Models and well created manuals are needed to provide stable maintenance. As we describe keep software well performed and stable requires proper way of maintenance. With over three thousand code changes and mostly related to modification of processes we observe that contribution of designers developers and testing site is necessary part of success.

Acknowledgment

This work is the result of the project implementation: Center of Information and Communication Technologies for Knowledge Systems (ITMS project code: 26220120020) supported by the Research& Development Operational Program funded by the ERDF.

References

[1] Lehman M. M.: “Lifecycles and the Laws of Software Evolution”, in Proceedings of the IEEE, Special Issue on Software Engineering, 19:1060-1076, 1980.

[2] Jarzabek S., “Effective Software Maintenance and Evolution: A Reuse-Based Approach”, in Auerbach Publicatio, Taylor & Francis Group, ISBN: 0-8493-3592-2, 2007.

[3] Grubb P.A., Takang A.A., “Software Maintenance: Concepts and practice”, SE, ISBN 978-981-238-425-6, World Scientific 2003.

[4] IEEE Std. 610.12, “Standard Glossary of Software Engineering Terminology”, IEEE Computer Society Press, Los Alamitos, CA, 1990.

[5] Pigoski T. M., “Practical Software Maintenance – Best Practices for Managing Your Software Investment”, John Wiley & Sons, New York, NY, 1997.

[6] Erlikh, L., “Leveraging Legacy System Dollars for E-Business”, in IEEE IT Pro, May–June 2000, pp. 17–23.

[7] Port, O., “The Software Trap — Automate or Else”, in Business Week, 3051(9), 1998, pp.142–154.

[8] Allen C., "Software maintenance – an overview”, in British Computer Society, Programming & Software Articles, en-GB 3rd, February 2006.

[9] Canfora G., Cimitile A., “Software Maintenance”, in Handbook of Software Engineering and Knowledge Engineering, volume 1,World Scientific, ISBN: 981-02-4973-X, December 2001.

[10] Fyson, M. J., Boldyreff, C., “Using Application Understanding to Support Impact Analysis”, in Journal of Software Maintenance – Research and Practice, 10(2):93-110, 1998.