Study of Impact Analysis of Software Requirement Change in SAP ERP

S. M. Ghosh[1], H. R. Sharma[1], V. Mohabay[2]

[1]*Chhatrapati Shivaji Institute of Technology, Durg (CG) – INDIA*
[2]*Department of Electronics and IT, Kalyan Mahavidalaya Bhilai (CG) - INDIA*
samghosh06@rediff.com, hrsharma44@gmail.com, mohabay@gmail.com

Abstract

As software systems become increasingly large and complex, the need increases to predict and control the effects of software changes. Our research objective is to provide an impact analysis methodology that uses historical change record for executable and non executable files in a software system to identify and prioritize potentially affected area of a system modification based on risk. Software Change Impact Analysis captures the latest information on the science and art of determining what software parts affect each other. It provides ideas for doing impact analysis better, presents a framework for the field, and focuses attention on important results. To identifies key impact analysis definitions and themes and illustrates the important themes to give a solid understanding for tackling impact analysis problems specifically in ERP. It includes study on software source code dependency analysis and software traceability analysis and shows how results from both areas can more effectively support impact analysis in software engineering repositories.

Keywords : *Enterprise Resource Planning, Change Management, Impact Analysis.*

1. Introduction

Software change management is the process of selecting which changes to encourage, which to allow, and which to prevent, according to project criteria such as schedule and cost. The process identifies the changes' origin, defines critical project decision points, and establishes project roles and responsibilities. It need to define a change management process and policy within company's business structure and team's development process. Change management is not an isolated process. It is clear to the project team on what, when, how, and why to carry it out.

Where Changes Originate

A variety of issues drive software changes but the random changes may give birth to bad quality software [1]. Understanding the origins of prospective changes is the first step in prioritizing them. The sources of change can be classified as planned development, unexpected problems, or enhancements.

Planned Software Development : Ideally, all software change would result from your required and planned development effort, driven by requirements and specifications, and documented in your design. However, adding new code is a change we must manage. Adding functions that were not requested (no matter how useful and clever) consumes project resources and increases the risk of errors downstream. Even requested features may range in priority from "mandatory" to "nice to have." Monitoring the cost to implement each request identifies features that adversely affect the project's cost-to-benefit ratio.

Unexpected Problems : will undoubtedly discover problems during any development effort and spend resources to resolve them. The effort expended and the effort's timing need to be proportional to the problem—small bugs should not consume your project budget. The impact of the software changes should be identified at an early stage[2].

The team must determine whether the code fails to implement the design properly or whether the design or requirements are flawed. In the latter case, we should be sure to correct design or requirements errors.

Enhancements : All software projects are a research and development effort to some extent, so it focuses on enhancement ideas. Here is where project management is most significant: the idea could be a brilliant shortcut to the project goal, or a wrong turn that threatens project success. As with requirements or design errors, we need to document these types of changes. Adhere to your development standards when implementing an enhancement to assure future maintainability.

2. Critical Decision Points in Change Progress [7]

We should address changes when they are only potential changes, before they've consumed project resources. Like any project task, changes follow a life cycle, or change process, that we must track. In fact, three critical decision points, as shown in Figure 1, drive any change process. These decision points form the framework of change management.

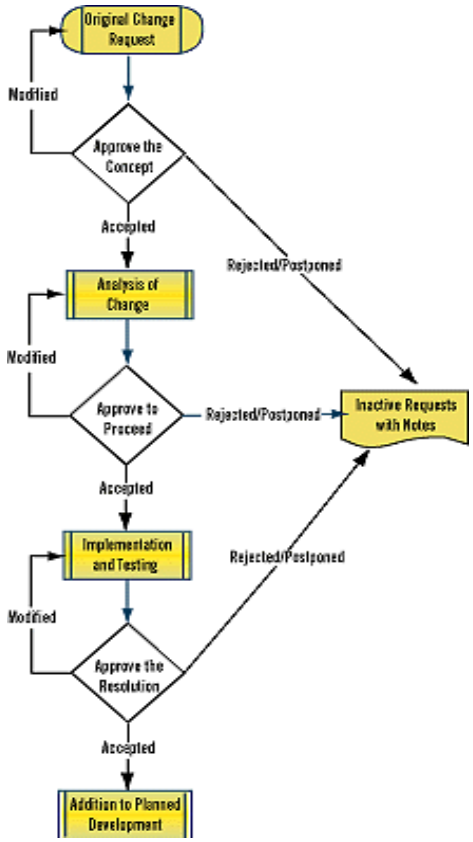

Figure 1 : Change Decision Points

Approve the Concept. Change requests come from testers or users identifying problems, and from customers adding or changing requirements. We want to approve all changes before investing significant resources. This is the first key decision point in any change management process. If we accept an idea, assign a priority to ensure appropriate resources and urgency are applied.

i. Approve to Proceed. Once we've accepted a change request, evaluate it against your project's current requirements, specifications, and designs, as well as how it will affect the project's schedule and budget. This analysis may convince we to revise your priorities. Sometimes, the team will discover that a complex problem has an elegant solution or that several bugs have a common resolution. The analysis will also clarify the cost-to-benefit ratio, making the idea more or less desirable. Once we clarify the facts, make sure the change is properly managed with a second formal review.

ii. Approve the Resolution. A change request is completed when the change is folded into the planned development effort. During requirements analysis and design phases, this may occur immediately after we approve the request. During coding, however, we often must conduct separate implementation and testing to verify the resolution for any unplanned changes, including both testing of the original issue and a logically planned regression test to determine if the change created new problems.

After testing, we must still review the change to ensure it won't negatively affect other parts of the application. For example, the developer may have changed a correct user prompt to match the incorrect software logic. If the testing indicates a risk of further problems, we might want to reject the change request even at this point.

iii. Rejected or Postponed Requests. At any of the decision points, we can decide whether to reject or postpone the change request. In this case, retain the change request and all associated documentation. This is important because if the idea comes up again, we need to know why we decided against it before. And, if circumstances change, we may want to move ahead with the change with as little rework as possible.

iv. Emergency Processing. If a problem has shut down testing—or worse, a production system—we may not have time for a full analysis and formal decision. The change management process should include an emergency path distinct from the flow , with shortened analysis and streamlined approval. Focus this process on an immediate resolution, whether a code "hack" or a work-around, that eliminates the shutdown. We can update the change request to document the quick fix and change it to a lower priority. By leaving the change request open, we won't omit the full analysis and resolution, but we can properly schedule and manage these activities. Alternately, we can close the emergency change request when the fix is in place, and create a new change request to drive a complete resolution.

3. Study of Change Management in ERP at Cement Business Group

The ERP Implementation at Cement Division SAP ERP covers the following modules: Finance Controlling (FI Co), Sales & Distribution (SD), Materials Management (MM), Human Resource & Payroll (HR PY), Plant Maintenance (PM) , Project Systems (PS) , Production Planning in Processing Unit (PP – PI) and Quality Management (QM).

Study of Impact Analysis [3]

As the document generated based on the study of impact analysis is done as follows:

Current impact analysis research follows two different basic methods in determining the effect of a change: either a static or dynamic analysis of the code base. Dynamic impact analysis techniques rely upon information gathered from a system during runtime, often gathered through execution of the system or test suites with an instrumented code base. Orso et al. compare two such dynamic techniques, `CoverageImpact` and `PathImpact` , to determine the major differences in cost and effectiveness. These two techniques examine call graphs and execution records from an instrumented execution of the system using a comprehensive test suite. Since Software change is generally considered to be expensive, difficult and time consuming hence it is important to consideration the inter-relationships between the change and the rest of the system. There are complex inter-dependent relationships between the different components of the software.

There are various reasons for changes in a software system[4] :

1. The performance of the software must be maintained.
2. The errors in specification, design and implementation must be corrected.
3. The final *product* must operate in diverse environments.
4. The final *product* must also evolve.

The problem of change in software can be viewed as[5] :

1. Identifying the *victims* of change
2. Identifying the types of changes that may operate on each object of the software;
3. Evaluating the effects of a change;
4. Acquiring knowledge on how to perform a change.

4. Impact Analysis that Studied in the Implemented ERP[6]

can be defined as the activity of identifying which object to modify to accomplish a change, that is, estimating the potential consequences of carrying out a change.

Impact analysis is focused on the following parameter and its reason. The study of impact analysis done generates the following assumption regarding ERP-SAP implementation that if relative benefit is high and other effected requirement is high than the priority to perform change should be given more importance.

1. To estimate the *cost* of a change. If a proposed change has the possibility of affecting large disjoint sections of the software, then the change might need to be re-examined and if possible abandoned in order to design another safer change.
2. To understand the *meaning* and the *relationship* between the item of change and the structure of the software. The impact analysis can use the notion of *conductivity* because there are some dependency relations that are better *conductors* of changes. These types of relation allow the identification of the parts of the software that *must* be modified and that parts *may* be modified for a given change-type.
3. To record the *history* of change related information and to evaluate the *quality* of a particular change. Examining the feedback information from users of the changed product helps in maintaining the quality of the software.
4. To determine the *parts* of the software that may need to be regression tested after a change is carried out and to point out the *vulnerability* of some critical sections of the software in

order to determine whether the functionality of those parts of the software is susceptible to changes made to the software.

5. Conclusion

The study of impact analysis done gave the conclusion that to identify the priority and based on it to identify the need changes in the software. Thus software changes need parameters to identify the priority and then based on it to identify the changes. This study gives the dimension to think about the software change management based on priority setting to perform the effective (time and cost) changes.

References

[1] Software Metrics, SEI Curriculum Module SEI-CM-12-1.1, Carnegie Mellon University, Software engineering Institute, December 1988.

[2] Software Maintenance: An Approach to Impact Analysis of Objects Change by Samuel Ajila, Software Practice and Experience, vol. 25(10), 1155–1181 (october 1995)

[3] Business Impact Analysis (BIA) document by Gartner Consulting , 16 December 2002

[4] Benington, H. 1956. Production of Large computer Programs. Proceedings of the ONR Symposium on Advance program methods for Digital Computers Pp 15-27.

[5] B. Boehm, "A spiral model of software development and enhancement", Computer. (Vol. 21.) pp. 61 – 72. ISSN 0018-9162, 1988.

[6] W. Royce, Managing the development of large software systems: concepts and techniques. WESCON Western Electronic Show and Convention. Los Angeles: WESCON. Pp. A/1-1 – A/1-9, 1970.

[7] S. M. Ghosh, H. R. Sharma, V. Mohabay , April 2011 , Analysis and Modeling of Change Management Process Model ,IJSEA (Vol. 5, No. 2) Pp. 123 - 133

Authors

S.M.Ghosh

Director, Prism School of Business and Entrepreneurship

S.M.Ghosh received his Master Degree in Computer Application and authored 3 books i.e. Software Engineering , Introduction to Computing and Management Information System. Published research papers in Software Change management related topics in National and International Journals. Designed Enterprise Resource Planning Software for Job based fabrication and Casting Steel Industries. Teaching QMS , Software Engineering and MIS to the PG Students from last 12 years.

Dr. H.R. Sharma

Director, Department of Computer Science, Chhatrapati Shivaji Institute of Technology

H.R. Sharma received his M.Tech degree from Delhi University in Computer Science and has completed his Ph.D from IIT, Delhi. He has also published more than 380 research papers at National and International level. He has supervised 9 research projects sponsored by UGC & AICTE. He has more than 45 years of teaching experience at Graduate and Post Graduate level. He is presently working as the Director of Chhatrapati Shivaji Institute of Technology, Durg.

Dr. V.K.Mohabey

Head of Department, Depatment of Electronics and Information Technology, Kalyan Post Graduate College

V.K.Mohabey has published more than 356 research papers in which 68 are published at National level and 12 at International level. He has supervised 7 research projects sponsored by UGC, DST, MAPCOST, etc. He is presently working as a Professor and Head of Department of Electronics and Information Technology in Kalyan PG College, Bhilai. He has over 35 years of teaching experience at Graduate and Post Graduate level.