Lt.Col.Dr. Phaderm Nangsue

กองวิชาวิศวกรรมไฟฟ้า ส่วนการศึกษา
โรงเรียนนายร้อยพระจุลจอมเกล้า

Scorpion :

A Prototype Solution for Solving Software Crisis

Abstract

Scorpion is a web-based application software that attempts to help solving the on-going problem of software crisis. It supports management of custom-made application life cycle from procurement to retirement. Procurement of custom-made software is much harder than procurement of its hardware counterpart because custom-made software procurement involves the full cycle of software engineering process. This process includes planning, system analysis, system design, system implementation, testing and maintenance. The required level of technical difficulties is often above the skill level available in the procuring organization. This problem has caused software projects to fail at a frightening rate, which is often referred to as "software crisis". According to the research paper from the Standish Group (1) there is only 16% successful software project. The rest is either a failure (31%) or required additional time/resource on the project (53%). This is where Scorpion comes into play. It combines the arts and science of software engineering into a single non-abstract tool that can guide the project to success by utilizing many tools such as risk analysis, brain-storming, feasibility study and knowledgebase of proven facts.

1. Introduction

In a software procurement process, the procuring organization hires a software house to build custom-made software. It may seem like a simple process: the organization pays the software house and the software house then submits the desired software to the organization.

However; in reality, it is quite difficult for this deal to complete satisfactorily. In a software project, the goal is to get quality software within the planned time frame and budget. As shown in **Figure 1,** a software project may be any dot in the figure. For project A, the quality is obtained but it is over-budget and exceeds the given time frame. Project B is even worse; it does not achieve any goal. Project C is the ultimate goal where the quality is obtained within the planned budget and time frame. Because it is not so easy to be at the desire position in the figure, there are so many failed software projects. Statistics (1) shows that the chance of getting this deal to complete satisfactorily is only 16%. 31% of the software projects is terminated before its completion. 53% is over budget or requiring additional time to complete. These figures are from the surrey of 23,000 projects in the U.S.A. during 1994-1998. It is quite frightening that software project fails at this rate. Moreover, the bigger the project is, the higher chance that it will fail. A one-year project has a success rate of 25%. For two-year project, the rate is dropped to 8%. For three-year project the success rate is zero! This is understandable because large projects require many people and the communication overhead for n people is n(n-1) = *O*(n2) which grows very quickly.

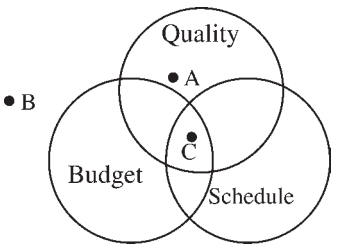

Figure 1 Project management goals.

1.1 Why software projects fail

According to (2), there are 6 major factors that cause software project to fail. These causes are 1) lack of planning 2) unclear scope of work 3) lack of communication 4) lack of skilled personnel 5) lack of collaboration and 6) under estimated project size. As can be seen, these causes are not that serious. It is not the lack of budget or the lack of technology to complete the project. All of these causes can be summarized to one issue: project management. Thus it is a good chance to solve this problem by creating a project management tool that addresses all of these causes. This kind of software procurement support system has never been created. The present so-called project management software is intended for software project manager in the developer side. The organization which pays for the project is usually not allowed to use the software and therefore collaboration is not achieved.

2. Philosophies in designing a solution for solving software crisis

Because we are building a new kind of system, we have three design philosophies as follow.

2.1) The system must be able to manage all stages of software life cycle. In each stage of software life cycle there are tools to support management of the software being built. For example, in the planning stage, there are tools for plan management, plan collaboration and plan checklist.

2.2) The system must have a ready-to-use knowledgebase. The first design philosophy is a program, which will need accompanied data. This data must be a high level data that has been proven that it is useful for project management. This proven data is our knowledge and many kinds of knowledge are accumulated into a knowledgebase. Examples of items in the knowledgebase are checklists, risks, non-functional requirements, templates and styles.

Checklists are lists of items that should be verified at a certain stage. For example, if a plan is created, it should be checked against the checklist for plan. If any item in the checklist is identified as "no-go", the issue must be solved before the plan is employed. A good set of checklist can prevent project failure by making sure that certain operations have not been unintentionally omitted.

Risks in the knowledgebase are a set of known risk factors that may affect project success. Examples of risks are late required hardware shipment and the need to shift project members to another project. When we manage a project, we do not need to identify all risks by ourselves- it is preconfigured for the system.

Non-functional requirements are a set of requirements that are not directly related to the function of the software. Function requirements are application-specific but non-functional requirements are relatively application-independent. Non-functional requirements include usability (user-friendliness), efficiency, reliability, security, maintainability and portability. A procuring organization may not be technical enough to express these requirements in a clear and complete manner. Therefore these initial non-functional requirements in the knowledgebase is really essential on specifying the scope of the software.

Document templates are structures of document to be generated. For example, if a user manual is to be written, the writer does not need to start writing from scratch. He can utilize a standard user manual template and fill in the desired information. This approach has the advantage of getting rid of the chance that an important issue is omitted. There are

about 30 document templates in the program.

Coding styles are formatting styles for source code in a particular language. When a procuring organization hires a software house to write software, the software house may divide the project into several pieces and assign each piece to a subcontractor. If the source code is to be submitted to the procuring organization, there would be many formatting styles, which is hard to maintain. Specifying coding style at the beginning of the project can prevent this problem.

2.3) The system must place emphasize on utilizing full potential of all team members. The procuring organization and the software house must be combined into a single team with a common goal of driving the project to success. Sometime, we have experts in the procuring organization and experts in the development team but there is no channel of communication that can bring expertise from individuals to the project. The system must serve as a channel of communication that is available anytime and anywhere and promote freedom of expression. Also, seniority in Thai culture is very strong. This can adversely affect freedom of expression. For example, if a senior says "I want to do it this way", no subordinates will dare to resist even though there is a better way to do it. The system must be able to handle this issue.

3. Architecture of Scorpion

From the three design philosophies of Scorpion, its architecture is laid out as shown in **Figure 2.**

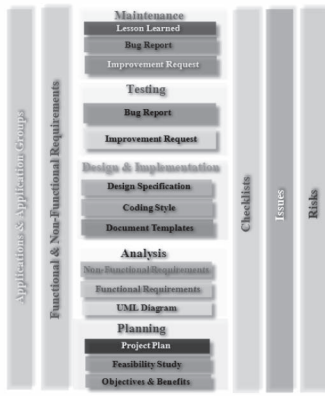

Figure 2 Scorpion Architecture

In the planning phase, Scorpion supports management of project objectives, benefits, feasibility study and project plans. Setting clear project objectives is important because it will direct the design and implementation of the software. Feasibility study allows team members to analyze the feasibility of the project in different aspects such as technical, timeframe, economics, legality, politics, marketing and actual utilization of the system. If there is an aspect with low feasibility, the project might be terminated at the present time to prevent future loss of time and resources on the project. Benefits to the projects allow us to analyze the worth of the project. Expensive projects with unclear benefits may be subjected to deduction or merging with other projects. Project plans specify the date of various project activities.

In the analysis phase of software development lift cycle, the program manages requirements. Requirements of the software are formulated as a list of requirement items. Each item must be discussed to determine whether it is really needed. Some requirement items might be deducted at this phase, while others might be merged with other requirement items. Requirements that truly meet the need of the organization will eventually emerge through the process of brainstorming that can be done anywhere and anytime.

In the design and implementation phase, the system supports management of project document, user interface design and database design. Project documents include design specification, UML diagrams, coding styles and document templates. Any type of documents (such as Microsoft word, Excel, PDF, ZIP, photo images or scanned documents) can be shared and discussed conveniently. The system supports user interface design by allowing developers to show their design of a user interface screen. Each screen is then discussed to search for the best possible design. For the database design, the team will try to specify what items are to be stored by the program. In relational database management model, data items are stored as tables with associated columns. Scorpion supports design of database by keeping track of what tables and fields are needed. Each field is associated with an access right. Only users with certain rights can access the data of the system. Access right included viewing, adding, modifying and deleting. All team members will have a chance to specify their needs before the program is built.

In the testing phase of the software life cycle, Scorpion supports submission of bug report and improvement request. Bug reports are submitted by a team member to indicate that there is a miss-behavior in the system. The member can specify details such as when and where the bug happens along with any related information that can help fixing the bug. Improvement request of the project is a request from a team member to the developer to specify improvement that can be done on certain issues. Scorpion classifies improvement request into 4 groups : "Requested", "Denied", "Developing" and "Done".

For the maintenance phase, Scorpion supports management of the lessons learned in addition to bug report and improvement request that exist the testing phase. Team members can keep tracks of what they learned in each project and accumulate this knowledge to get better in their field of interest.

Scorpion also supports management of items that are applicable to any phase of software development life cycle. These items include application and application

group, checklists, risks, issues and meetings. Applications that are in the same budget allocation may be considered as being in the same group. Usually all applications in the same group have the same start-of-project and end-of- project dates. Checklists are managed so that items in the list can be added or removed as needed. An issue can be raised by a team member at any time if there is a concern about the project. Each issue is discussed until a clear solution is evolved. If a face-to-face meeting is conducted, the user can record the meeting for future reference.

4. Features

Scorpion has a large set of features. There are a total of 228 users interface screens written in various languages such as SQL, VB.Net, HTML, JavaScript, CSS and XML. The followings are major features of the program:

1) Support for setting clear objectives of the project. Each objective can also be assessed to determine the level of achievement.
2) Support for analyzing the worth of the project. All benefits of the projects can be openly discussed and determined whether the benefits are worth the cost of the project.
3) Support for feasibility study. All feasibilities are statistically summarized and presented to determine whether the project is feasible in all aspects (technical, timeframe, marketing, legal or actual utilization).
4) Support for risk analysis. Each risk has an associated probability of occurrence and a level of consequence when the risk happened. Risks that have high probability of occurrence and severe consequences require extra attention to prevent it from happing or mitigate its severity.
5) Support for making use of existing similar systems. In some cases, near by organizations may be running a system that is similar to the one that is being built. Large volume of knowledge can be readily transferred from the existing systems to the new system.
6) Support for planning process. Each plan can be discussed and checked against related checklist to make sure that the obtained plan is the best possible one.
7) Support for database design. Each item to be stored by the application being built can be managed and discussed. Access right of each item can be specified in this phase of the application life cycle.

8) Support for keeping tracks of all the functional and non-functional requirements of the application. Each requirement can be analyzed and tracked. Some requirements may not be approved to be included in the system specification; while other requirements may be in a development or verified stage.

9) Support for management of next-phase features. Scorpion promotes all team members to think ahead. If the second or third phase of the project is to be built, what should be their features. The current system can be designed to allow smooth transition to the next phase.

10) Support for checklist management. Checklists are classified into group with clear indication of when the checklist is applicable.

11) Support for bug reports and improvement requests. All bug and requests are tracked. Developers can discuss on issues concerning the bug or request with the submitter.

12) Support for management of meeting records. All face-to-face meeting can be recorded for reference purpose.

13) Support for discussion of project-related issues. A team member can raise an issue that requires particular attention or clarification. The issue is then discussed until it is clear.

14) Support for management of lesson learned. A team member can keep track of what he has learned from a project. These are valuable knowledge that might be useful in the future.

15) Support for application and application group management. Application can be classified by its group, development stage, or warning level. A warning level is a system-wide indicator that specifies the overall condition of the project. Examples of warning levels include "OK", "Caution" and "Danger"

5. Summary

Scorpion is an attempt to bring theories in software engineering textbooks along with the art and science of project management into one complete software package that can guide the project to success. Experienced or novice project managers can start using the program for their project immediately at their preferred location or time of day. All known causes

of project failure have been analyzed and tackled by various tools, knowledgebase and collaboration. Moreover, it supports management of the procured applications' life cycle until its retirement phase. It is our goal to solve software crisis of this era and bring the software project failure rate down to a more reasonable figure.

References

(1) Standish Group Report, The Chaos Report, http://www.standishgroup.com/sample_research/chaos_1994_1.php

(2) Al Neimat, Taimour, Why IT Projects Fail, http://www.projectperfect.com.au/info_it_projects_fail.php 2005

(3) Roger S. Pressman, Software Engineering: A Practitioner's Approach, McGraw-Hill Science/Engineering/Math; 6 edition, 2004

(4) Hoffer, Jeffrey, Modern Systems Analysis and Design (4^{th} Edition), Prentice Hall; 2004.