Proceedings of MASPLAS'04
Mid-Atlantic Student Workshop on Programming Languages and Systems
Seton Hall University, April 3, 2004

Chapter 1 Efficient Software Development

Introduction

Most software projects fail. In fact, the Standish group reports that over 80% of projects are unsuccessful either because they are over budget, late, missing function, or a combination. Moreover, 30% of software projects are so poorly executed that they are canceled before completion. As per the market experience, software projects using modern technologies such as Java, J2EE, XML, and Web Services are no exception to this rule.

Traditional Software Development Cycle

1. Review The Specifications.
    a. Functional Specifications.
    b. Technical Specifications.
2. Informal Design.
    a. List the Major Task.
    b. List the Subtask, sub- subtasks, and so on.
3. Formal Design.
    a. Create a formal design from tasks lists.
    b. Desk Checking.
4. Code and Compile the program.
5. Software Testing and Debugging.
6. Risk Analysis.
7. Quality Assurance.
8. Use and Maintenance of a program.

Best Practices of Efficient Software Development

1. Development process.
2. Requirements.
3. Architecture.
4. Design.
5. WebSphere Application design.
6. Construction of the code.
7. Peer reviews.
8. Testing.
9. Performance testing.
10. Configuration management.
11. Quality Control.
12. Quality Assurance.
13. Quality and defects management.

14. Deployment.
15. System operations and support.
16. Risk Analysis.
17. Data migration.
18. Project management.
19. Measuring success.

Development process

It is important to choose the appropriate development lifecycle process to the project at hand because all other activities are derived from the process. For most modern software development projects, some kind of spiral-based methodology is used over a waterfall process. There are several choices, including the Rational Unified Process (RUP), Global Services Method, and eXtreme Programming (XP). Having a process is better than not having one at all, and in many cases it is less important on what process is used than how well it is executed. The commonly used methodologies listed above all contain guidance about how to execute the process and templates for artifacts

Requirements

Gathering and agreeing on requirements is fundamental to a successful project. This does not necessarily imply that all requirements need to be fixed before any architecture, design, and coding is done, but it is important for the development team to understand what needs to be built. Quality requirements are broken up into two kinds: functional and non-functional. A good way to document functional requirements is using Use Cases. Note that Use Cases are used for non-OO projects. Non-functional requirements describe the performance and system characteristics of the application. It is important to gather them because they have a major impact on the application architecture, design, and performance.

Architecture

Choosing the appropriate architecture for your application is key. Many times companies are asked to review a project in trouble and have found that the development team did not apply well-known industry architecture best practices. A good way to avoid this type of problem is to contact the consultants. The consultants can work side by side with your team and ensure that the projects get started on the right track. Tried and true practices are called patterns and they range from the classic Gang of Four patterns, Java patterns, to EJB design patterns. Sun's equivalent is the Core J2EE Patterns catalog. Many companies also publish best practices and reference application architectures. Many projects fail as discussed in the introduction. The study of these failures has given rise to the concept of antipatterns. They are valuable because they provide useful knowledge of what does not work, and why.

Design

Even with a good architecture it is still possible to have a bad design. Many applications are either over-designed or under-designed. The two basic principles here are "Keep it Simple" and information hiding. For many projects, it is important to perform Object-Oriented Analysis and Design using UML. Reuse is one of the great promises of OO, but

it is often unrealized because of the additional effort required to create reusable assets. Code reuse is but one form of reuse and there are other kinds of reuse that can provide better productivity gains.

WebSphere Application design

Organization should have extensive knowledge of the best practices and design patterns for the WebSphere product family. Each project is different and the consultants have the experience to help the organization. There is still a tremendous return on investment (ROI) even if the organization uses the consultants for a short time because they save the costs later in the project. Organizations experts have also published a great deal of this wisdom, including considerations for high-performance Web sites and guidelines for autonomic computing.

Construction of the code

Construction of the code is a fraction of the total project effort, but it is often the most visible. Other work equally important includes requirements, architecture, analysis, design, and test. In projects with no development process (so-called "code and fix"), these tasks are also happening, but under the guise of programming. A best practice for constructing code includes the daily build and smoke test. An organization goes one step further and suggests continuous integration that also integrates the concept of unit tests and self-testing code. Note that even though continuous integration and unit tests have gained popularity through XP, we can use these best practices on all types of projects

Peer reviews

It is important to review other people's work. Experience has shown that problems are eliminated earlier this way and reviews are as effective as or even more effective than testing. Any artifact from the development process is reviewed, including plans, requirements, architecture, design, code, and test cases. Peer reviews are helpful in trying to produce software quality at top speed.

Testing

Testing is not an afterthought or cutback when the schedule gets tight. It is an integral part of software development that needs to be planned. It is also important that testing is done proactively; meaning that test cases are planned before coding starts and test cases are developed while the application is being designed and coded. There are also a number of testing patterns that have been developed.
Testing is the set of activity that can be planned in advance and conducted systematically. For this reason a template for software testing- a set of steps into which we can place specific test case design technique and testing methods- should be defined for the software process. Software developers with a template for testing and all have the following generic characteristics.

- Testing begins at the component level and works “outward” towards the integration of the entire computer- based system.
- Different testing techniques are appropriate at different points in time.

- Testing is conducted by the developer of the software and (for large projects) an independent test group.
- Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.

Software testing is one element of broader topic that is often referred to as verification and validation (V & V).

Verification refers to the set of activities that ensure that software correctly implements a specific function.

Validation refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements.

Performance testing

Testing is usually the last resort to catch application defects. It is labor intensive and usually only catches coding defects. Architecture and design defects may be missed. One method to catch some architectural defects is to simulate load testing on the application before it is deployed and to deal with performance issues before they become problems.

Configuration management

Configuration management involves knowing the state of all artifacts that make up your system or project, managing the state of those artifacts, and releasing distinct versions of a system. There is more to configuration management than just source control systems, such as Rational Clearcase. There are also best practices and patterns for configuration management.

Quality Control

Quality control involves the series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements placed upon it. Quality control includes a feedback loop to the process that created the work product. The combination of measurement and feedback allows us to tune the process when the work products created fail to meet their specifications. This approach views quality control as part of the manufacturing process.

Quality Assurance

It's not enough to talk and talk saying that software quality is important, we have to

- Explicitly define what is meant when we say "software quality".
- Create a set of activities that will help ensure that every software engineering work product exhibits high quality.
- Perform quality assurance activities on every software project.
- Use metrics to develop strategies for improving you software process and, as a consequence, the quality of the end product.

Quality and defects management

It is important to establish quality priorities and release criteria for the project so that a plan is constructed to help the team achieve quality software. As the project is coded and

tested, the defect arrival and fix rate can help measure the maturity of the code. It is important that a defect tracking system is used that is linked to the source control management system. For example, projects using Rational ClearCase may also use Rational ClearQuest. By using defect tracking, it is possible to gauge when a project is ready to release.

Deployment

Deployment is the final stage of releasing an application for users. If you get this far in your project - congratulations! However, there are still things that can go wrong. You need to plan for deployment and you can use a deployment checklist on the Construx Web site.

System operations and support

Without the operations department, you cannot deploy and support a new application. The support area is a vital factor to respond and resolve user problems. To ease the flow of problems, the support problem database is hooked into the application defect tracking system.

Risk Analysis

Risk analysis and management are the series of steps that help a software team to understand and manage uncertainty.

Software Risks

This always involves two characteristics.

- **Uncertainty**- the risk may or may not happen; that is, there are no 100% probable risks.
- **Loss**-if the risk becomes a reality, unwanted consequence or losses will occur. There are different categories of risks.

a. Project Risks

This threatens the project plan. That is, if project risks become real, it is likely that project schedule will slip and that costs will increase. Project risks identify potential budgetary, schedule resource, customer, etc.

b. Technical Risks

This threatens the quality and timeliness of the software to be produced. If a technical risk becomes a reality, implementation may become difficult or impossible. Technical risks identify potential design, implementation, interface verification and maintenance problems.

c. Business Risks

This threatens the viability of software to be built. Business risks often jeopardize the project or the product.

Data migration

Most applications are not brand new, but are enhancements or rewrites of existing applications. Data migration from the existing data sources is usually a major project by itself. This is not a project for your junior programmers. It is as important as the new application. Usually the new application has better business rules and expects higher quality data. Improving the quality of data is a complex subject outside the scope of a beginner.

Project management

Project management is a key to a successful project. Many of the other best practice areas described in this article are related to project management and a good project manager is already aware of the existence of these best practices. Given the number of other checklists and tip sheets for project management, it is surprising how many project managers are not aware of them and do not apply lessons learned from previous projects, such as: "if you fail to plan, you plan to fail." One way to manage a difficult project is through timeboxing.

Measuring success

We can measure our development process against an industry standard known as the Capability Maturity Model (CMM) from the Software Engineering Institute. Most projects are at level 1. If you implement the best practices described above, then you could be well on the way to achieving a higher maturity level and a successful project.

Case Study

Aim: -

30% REDUCTION IN LIFE CYCLE COSTS OF CUSTOMISED PRODUCT DEVELOPMENT

Case: -

ASHCON Automation is a successful manufacturer of precision turnkey robotic systems, selling primarily to the ship building industry. The software associated with individual applications is customized, and therefore a lot of manpower is assigned to software development. ASHCON wanted to reduce the life-cycle cost of these software products, by focusing more on development, reusability, documentation and maintenance. As a result of the work performed in this project, ASHCON have been able to reduce life-cycle cost by 30%.

Business Area And Market Profile

An efficient software development and maintenance process can tremendously increase the competitiveness of a final product. This realization amongst senior management is clearer in companies that are sole software developers, as their core product is 'software'. However amongst companies like ASHCON, who develop innovative software to support their final product, i.e. precision engineered systems employing robotic technology, this realization is not so clear. ASHCON have on the other hand, recognized the need to improve their software process. ASHCON's primary market, 'shipbuilding', has been through many changes in the last few years. These changes have been unpredictable. Suppliers have to remain competitive to stay ahead in this niche market. The company develops precision turnkey robotic systems for steel cutting applications. Specification requirements on each project are different, mainly because different types of cuts are required. The application software associated with the robotic systems is becoming more central to the solution. One of the reasons for this is that PC based solutions are preferred over embedded. The software departments within ASHCON have experienced these changes and are critical in keeping life-cycle costs down. The company's objective in this project was to implement a process that would add structure to the development and maintenance of the software.

Reducing Life Cycle Costs

ASHCON tackled the areas of Re-usability of source, documentation, and maintenance to solve the problem. The overall goal for ASHCON was to reduce the life-cycle costs of customized software by 30%. The Robot department of ASHCON carried out the experiment and was motivated in implementing a software process that would reduce life cycle costs and ultimately bring the following benefits:

1. Quick time to market
2. Software flexibility
3. Software tractability
4. Product cost reduction
5. Increased profit margins

The aim of the project was to create a framework (tools), techniques and training for handling customized software.

Work Performed

The CUSTOMIZE project was performed by the Robot department which consisted of 9 experienced software and robot engineers. The manager of the Robot department was given the sole responsibility of running the project. All personnel working on the project had to directly report to this person. The project focused more on acquiring new programming methods and not on the organizational issues. The goal was to reduce the life-cycle costs of customized software. Firstly, tools and techniques were selected on the basis of their merits by performing a features and benefits study. A lot of this activity was

subcontracted to SINTEF (a Norwegian independent non-profit research agency). This working relationship proved to work well because SINTEF possessed the right level of experience. The following methods and tools were selected:
1) Literate programming together with version control software was used to improve the quality of the software documents and to maintain a consistent source code.
2) Data driven configuration files and SQL (Structured Query Language) databases for a common data storage format. This reduced the number of customized software modules. Procedures were implemented and put in place to ensure that a quality process was in place. Procedures for development management, maintenance and non-conformance for this class of software were implemented either as new procedures or as appendices or parts of existing procedures. Actual method and tool details were included in the programmer's handbo ok to meet quality auditing needs. The new tools and techniques were then implemented on the baseline product. The baseline product was the development of a PC based CAD robot interface software for automatic off-line programming of a Motoman SK16 robot based on NUPAS CAD/CAM files.
1. All source code was written using literate programming techniques in Microsoft Word using the HTML file format. The HTML files were then processed by an in-house version of CLIP which extracted the source code part of the HTML files and created standard C++ and included files (*.Hand *.HPP files).
2. All data storage was performed in configurable SQL tables.
3. An analysis on where mutant software may occur had been carried out. ASHCON believe they have made the system configurable with respect to customer variations.
4. The data import module reads data from the NUPAS file to a run-time configurable SQL database. The imported data was validated automatically by using a data driven configuration file which was run-time configurable from the same file. One of the by-products from the work performed was the 'Programmers Handbook'. The handbook is seen as the 'bible' and used by all of the software development team. It contains guidelines for developing and operating the customized software. It addresses methods and tools, programming conventions, rules for documentation, interdependencies and re-use of source code, testing, release, and software maintenance.
The findings in this project have been actively disseminated externally in conferences related to shipbuilding. These were received with interest. Internal dissemination included presenting to the Sales and Marketing team as well as top management. The aim here was to make use of results to cost efficiently sell and commission customized software.

Project Results

Below are the main findings of the experiment:
1. Literate programming (programming Microsoft Word using the HTML file format and export the source code to the compiler using an in-house version of Clip) have improved the readability of the source code documents. Literate programming have also enabled the programmers to write data logically belonging to each other in one document and in a logical sequence, instead of placing the object definition to one file (e.g. a*.h file), the

object initialization in another file (e.g. a *.cpp file) and the data management in a third file (e.g. another*.cpp file).
2. The use of a program launcher has reduced the time for testing, debugging and maintaining the source code and finished programs. For instance after a revision of dynamic link library etc.
3. ASHCON now have a uniform data storage file format (i.e. SQL tables), a uniform way of inputting and retrieving data, a uniform way of presenting the data (using Microsoft access) as a result of the use of SQL database for all data storage. It is easy for third parties to interface to ASHCON software by using Open Database Connectivity (ODBC) and Application Programming Interface (API). Data sharing is obtained free and ASHCON do not have to maintain all the different file format modules they used before.
4. The use of database driven configuration files have increased ASHCON's ability to tailor make GUIs without the need of mutant versions. This method has already reduced the number of mutant software versions at ASHCON. Since then they have implemented this method in two of their version upgrades.
5 Simplification has been achieved in the number of files and source code lines as a result of introducing literate programming. Literate programming has been of most benefit. The number of files and source code lines are the driving cost factors with respect to software maintenance and upgrades. Other measured results indicate.
6. Reduction in the number of bugs from 45 to 3 three months after the first release. From a business prospective, the following targets have been met:
7. Delivery time on customized software has been reduced by more than 35%.
8. Increased reliability as a result of improved quality. One of the negative aspects of the results was that the interface card had to be upgraded if changes to data were required. This meant that changes had to be carried out at client site.

Lesson Learnt

From a technical prospective, the tools and methods selected in this project have worked well. The use of literate programming has been of most benefit with regards to cost reductions. Although, the software team within the Robotics department is highly experienced in software development, they have acquired the following skills as a result of the project:
1. Literate programming
2. Configuration management (already possessed but now is knowledge documented)
3. Configurable SQL databases
4. Data driven configuration file
As previously mentioned, one of the negative aspects of the results technically is that the interface card has to be upgraded if changes to data are required. This means that the database has to be upgraded at the client site. The project has brought many business benefits to the software product. The obvious one being the 30% reduction in life-cycle costs. ASHCON are confident in seeing improvements in other business areas as the process becomes more stable. The department as a whole is more process oriented, and the project has also made them more technology driven. There were some concerns on

the issue of project management, and that some developers felt they wanted to play a more active role in the project by making suggestions and recommendations.

Future Plans Of ASHCON

ASHCON would focus on the following as future work:
1. Other software products, which are now being upgraded and updated with literate programming documents, configurable SQL based data storage and data driven configuration files.
2. Automating the 'Programmers Handbook' on a network

Conclusion

Always it is necessary to choose the best practices to create best software for efficient use. Hence efficient software development plays and key role in software development.

By, K. V. Natarajan
Department of Information Technology
Mepco Schlenk Engineering College
Anna University
India