Best practices for software development projects

Mike Perks

August 10, 2006
(First published June 16, 2003)

This article provides a list of best practices for improving the success of your software development projects.

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. In our experience, software projects using modern technologies such as Java, J2EE, XML, and Web Services are no exception to this rule.

This article contains a summary of best practices for software development projects. Industry luminaries such as Scott Ambler, Martin Fowler, Steve McConnell, and Karl Wiegers have documented many of these best practices on the Internet and they are referenced in this article. See also the Related information section at the end of this article. The companion article, Guide to Running Software Development Projects, describes the top ten factors that help improve the success of your project.

Best practices

1. 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), IBM® 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. In addition, the RUP has a series of books that describe the...
best practices for using RUP [1][2][3][4] although if you do not choose to use RUP, these books still provide an excellent source of best practices. It is also possible to add plugins to the RUP.

2. 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 are 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. A definitive book on the subject of use cases is by Armour and Miller [5]. 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.

3. Architecture - Choosing the appropriate architecture for your application is key. Many times IBM is asked to review a project in trouble and we 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 IBM. Our 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 [6] patterns, Java patterns [7], to EJB design patterns [8]. Sun's equivalent is the Core J2EE Patterns catalog [9]. 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.

4. 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. There are many books on UML, but we recommend UML User Guide[11] and Applying UML and Patterns[12]. 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.

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

6. 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,
7. 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 or even more effective than testing. Any artifact from the development process is reviewed, including plans, requirements, architecture, design, code, and test cases. Karl Wiegers paper on the Seven Deadly Sins of Software Reviews explains the correct ways to perform peer reviews. Peer reviews are helpful in trying to produce software quality at top speed.

8. 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.

9. 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.

10. 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 [13] for configuration management.

11. 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.
12. 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.

13. 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.

14. 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 this article.

15. Project management - Project management is 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. Our recommended bible for project management is Rapid Development by Steve McConnell [14]. 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.

16. Measuring success - You can measure your development process against an industry standard known as the Capability Maturity Model (CMM) from the Software Engineering Institute at Carnegie Mellon University. Most projects are at level 1 (initial). If you implement the best practices described above and the guidelines in the companion article, Guide to Running Software Development Projects, then you could be well on the way to achieving a higher maturity level and a successful project.

Conclusion
This article provided a list of best practices that help improve the success of a software development project. By following these best practices, you have a better chance of completing your project successfully.

Related information
10. IBM Redbooks. Search for "patterns AND e-business".

© Copyright IBM Corporation 2003, 2006
Trademarks
(www.ibm.com/developerworks.ibm/trademarks/)