Software Development Methodologies

Article · August 2013

CITATIONS
12

READS
78,116

1 author:

David Young
GDIT & Alabama Supercomputer Center
13 PUBLICATIONS **1,952** CITATIONS

Software Development Methodologies

About this white paper: This paper was written by David C. Young, an employee of CSC. It was written as supplemental documentation for use by the HPC account holders at the Alabama Supercomputer Center (ASC). This was originally written in 2012, and updated in 2013.

Software development methodologies have traditionally been covered little or not at all in some of the IT degree curriculums. However, individuals working for professional software development organizations find that it is a big part of their work environment. There is currently a trend of managing other types of work following the general scheme of a software development management practice.

A software development methodology is a way of managing a software development project. This typically address issues like selecting features for inclusion in the current version, when software will be released, who works on what, and what testing is done.

No one methodology is best for all situations. Even the much maligned waterfall method is appropriate for some organizations. In practice, every organization implements their software development project management in a different way, which is often slightly different from one project to the next. None the less, nearly all are using some subset or combination of the ones discussed here.

Choosing an appropriate management structure can make a big difference in achieving a successful end result when measured in terms of cost, meeting deadlines, client happiness, robustness of software, or minimizing expenditures on failed projects. As such, it is worth your time to learn about a number of these and make your best effort to choose wisely.

There are definitely trends in the project management field. The following is a discussion of methodologies that were getting a fair amount of exposure when this was written in 2012. This document is intended to give you just enough comparison to choose which to investigate further.

Waterfall - This is the original, traditional method of software development. It approaches software development like you would approach building a house... with the view that changes after the fact are prohibitively expensive. This is a linear method in which there is a big emphasis on collecting requirements and designing the software architecture before doing development and testing. The advantage of this is that the project is well planned, minimizing on mid-project costs for changing requirements, and that these projects tend to be well documented. This typically results in major version releases with a significant number of new features every few years. The disadvantage is that it is very hard to adjust the feature set in the middle of development, which often happens as problems are uncovered in development or changing business environments change what is needed in the software. This is such a problem that many organizations put in a place a "feature freeze" in which they refuse to alter the features to be included in a given version once software writing begins, and thus needed features get pushed to later major versions forcing the users of the software to wait years for those features. Anyone who has worked on a waterfall managed project has experienced the frequent flaps over feature changes suggested by software developers, management, and clients which often necessitate an inefficient micromanagement format... all of which are arguments against this process.

In the current lexicon, "**Agile**", "**Crystal**" and "**Unified Process**" are general terms for families of similar software development methodologies.

Agile family - Agile methods are meant to adapt to changing requirements, minimize development costs, and still give reasonable quality software. Agile projects are characterized by many incremental releases each generated in a very short period of time. Typically all members of the team are involved in all aspects of planning, implementation, and testing. This is typically used by small teams, perhaps nine or fewer, who can have daily face-to-face interaction. Teams may include a client representative. There is a strong emphasis on testing as software is written. The disadvantages of the Agile methods are that they work poorly for projects with hundreds of developers, or lasting decades, or where the requirements emphasize rigorous documentation and well documented design and testing.

- **SCRUM** - is currently the most popular implementation of the agile ideals. Features are added in short sprints (usually 7-30 days), and short frequent meetings keep people focused. Tasks are usually tracked on a scrum board. The group is self-organizing and collaboratively managed, although there is a scrum master tasked with enforcing the rules and buffering the team from outside distractions.
- **Dynamic Systems Development Model (DSDM)** - is an agile method that sets time, quality, and cost at the beginning of the project. This is accomplished by prioritizing features into musts, shoulds, coulds, and won't haves. Client involvement is critical to setting these priorities. There is a pre-project planning phase to give the project a well-considered initial direction. This works well if time, cost, and quality are more important than the completeness of the feature set.
- **Rapid Application Development (RAD)** - is a minimalist agile method with an emphasis on minimizing planning, and a focus on prototyping and using reusable components. This can be the best choice when a good prototype is good enough to serve as the final product. RAD has been criticized because the lack of structure leads to failed projects or poor quality products if there is not a team of good developers that feel personally committed to the project.
- **Extreme Programming (XP)** - is a frequent release development methodology in which developers work in pairs for continuous code review. This gives very robust, high quality software, at the expense of twice the development cost. There is a strong emphasis on test driven development.
- **Feature-Driven Development (FDD)** - is an iterative development process with more emphasis on planning out the overall architecture, followed by implementing features in a logical order.
- **Internet-Speed Development** - is an iterative format that emphasizes daily builds. It is tailored to the needs of open source projects where volunteer developers are geographically distributed, and working around the clock. The project is built from a vision and scope statement, but there are no feature freezes. Development is separated into many small pieces that can be developed in parallel. The down side of this process is that the code is constantly in flux, so there are not necessarily stable release points where the code is particularly well tested and robust.

Evo - is an older, less known, evolutionary system developed at Hewlett-Packard. It bears significant similarity to agile development, and adds in a link to sales and manufacturing cycles. Unlike Agile, it puts more emphasis on having a technical manager to assign tasks. There should also be a user liaison, sometimes someone who is a subject matter expert in their own right. Evo is reportedly very successful, and probably only lacks the quantity of publicity that Agile has received. If your old school management is reticent to adopt the self-organizing, self-managing Agile format, consider Evo.

Unified Process family - An interactive development process for larger, often more bureaucratic, development teams. There is a strong focus on use cases, which in turn suggest requirements. There is also an emphasis on choosing the best architecture. The highest risk tasks are done first in order to give an early break point where the project can be cancelled if it is doomed to failure. The labor pool is used efficiently by often having various percentages of requirements, design, implementation, and testing all being performed in parallel.

- **Rational Unified Process (RUP)** - (name owned by IBM) This implementation of the unified process is an IBM product consisting of documentation, management software tools, training, and certifications.
- **Open Unified Process (OpenUP)** - An open source implementation developed by the Eclipse foundation. It targets small, collocated teams that want the structure of a unified process.
- **Essential Unified Process (EssUP)** - EssUP is a list of processes from unified process and other methodologies. The project leader/group then picks from this list to tailor the process to their needs.
- **Agile Unified Process** - is an attempt to simplify unified process and add in a couple Agile principles such as test driven development.
- **Enterprise Unified Process** - for VERY BIG projects that will have a long-term commitment for support and the eventual retirement of the software.

PRINCE2 (PRojects IN Controlled Environments 2) is the project management format mandated by the UK government for public projects. It is common in Europe. It is not specific to software development. PRINCE2 focuses on the process of how things are to be done. It is based on seven principles (continued business justification, learn from experience, defined roles and responsibilities, manage by stages, manage by exception, focus on products, and tailored to suit the project environment). This process emphasizes management of resources, but may not be appropriate for small projects.

Project Management Body of Knowledge (PMBOK) - is a generic project management framework set forth by the Project Management Institute (PMI), and adopted as a standard by ANSI and IEEE. It focuses more on what is to be done (tools and techniques). This is used as a framework for software development when complying with those standards is a mandatory requirement.

Capability Maturity Model Integration (CMMI) - is a process improvement approach. It focuses on the organization's needs, bureaucracy, and structure. This is not a software development methodology in itself, but it provides a framework for reviewing and improving the software development methodology. It defines a maturity level to quantify how well the organization is running. CMMI was developed in the software engineering community, but has since then been generalized and become extremely abstract.

Microsoft Solutions Framework (MSF) - This is a generic project management methodology focused on IT including software development, and deployment of equipment. MSF uses two models; a team model describing roles of individuals in the software development group, and a governance model describing the stages in the development process. MSF contains templates for Agile and Capability Maturity Model Integration.

Crystal Methods Methodology - Crystal methods are a family of related methodologies, each named with a color. The philosophy of crystal methods is that the process should be designed around the strengths and weaknesses of individuals on the team. The various levels of crystal method vary in how light or heavy weight they are. The lightweight methods might be best for small, short-term projects. The heavy weight methods are better suited for cases where human life is involved and thus bugs cannot be tolerated. All incorporate frequent deliveries, close communication with expert users, and personal safety.

Joint Application Design (JAD) is also referred to as **Joint Application Development** - This is a process for requirements gathering in which various parties including multiple members of the client organization spend days participating in a requirements gathering workshop. The empirical observation is that having this better-defined set of requirements up front gives a modest reduction in development costs.

Lean Development (LD) is sometimes called **Lean Software Development (LSD)**. This process is intended to give the absolute minimum cost. Developers focus on 80% today, not 100% tomorrow. Expect poor quality, minimally featured, software made quick and cheap. If you are lucky, it will come out just good enough to get the job done.

Spiral Model - is a software development process for creation of new technology where failure is a big risk. Spiral projects start small, first investigating the highest risk issues then slowly expand the project once those key components are functioning. Two or more phases of prototyping are done before the final implementation. The spiral model is considered to be better than waterfall for large, expensive, complicated projects. Spiral development is generally considered inappropriate for small projects.

Systems Development Life Cycle (SDLC) - Is a more formalized process for handling large projects where documentation, training, integrity, and security are vital to the project success. SDLC projects typically use object oriented analysis and design. Multiple models will be prepared for use cases, relational data, user interface, and a more abstract conceptual model. Multiple types of software testing are employed. It assumes that different management teams may be handling requirements, implementation, deployment, and monitoring.

Apple design process. The Apple corporation has made an enormous amount of money creating innovative software and hardware products. Apple's product development process is somewhat unique. Apple does a lot of design specification before beginning work on a product development project. They produce three documents that must be approved by senior management before the project can begin. The first document is an engineering specifications document, which details the product dimensions, weight, battery life, features, etc. The second document is a marketing plan, which details who the target consumer is, how much technical experience they have, what price they are willing to pay, how to advertize and sell to them, etc. The third document is the user experience document. Apple is very much a design company that thinks about every aspect of the user experience from the packaging to the first thing you see when you open the box, to how many functions are too many for the product to be conveniently usable. Once these documents are approved, the project manager has the very difficult job of sticking to the design plans. Most companies work on a consent model… the sales department will sign off on the product if you put in functions they suggest, the software development group signs off if you put in their favorite features, the VP signs off if you put in her favorite feature, and the product goes to market late with a bloat of extra features. Apple works on a consultancy model… everyone gets to give their opinion, but it's the project manager's job to throw out all of those suggestions unless everyone agrees that an extra feature is critically important. Keeping this glut of extra features out of the products is what gives Apple products a clean, easily used design. Apple also pushes engineering and manufacturing to find ways to follow idealistic designs and aggressive quality goals, rather than making something similar to the initial design that was easier to engineer, program, and manufacture. Within these aggressive guidelines, Apple has a fairly iterative process for attacking engineering challenges. No product development organization can get everything perfect all the time, but Apple certainly strives for perfection on the first try. Another unique aspect of the Apple culture is that they don't do market research... if you aren't an industry visionary with good design sense and taste then you should not be on the senior design team at Apple.

These management practices are all designed for managing various size groups of software developers. However, individuals writing software (a team of one) can still pull relevant ideas from them. These include using use cases to define requirements, making prototypes for experimental features, test driven development, using a source code management tool like Subversion or Mercurial, using management tools to track progress, features, & bugs, and more.

Software testing

The creation of computer software is a unique activity. It the design of other products, such as building a house or writing a book, small mistakes may only have a minor impact. However, computer software is very integrated so small mistakes can have a cascading impact on many functions in the software package. Also, software source code is so readily modified that there can be a lot of pressure to get major features working first, then go back to the same sections of code to fix the behavior for unusual circumstances. Because of these factors, the testing and debugging cycle is key to getting reliable, robust, bug free software.

Software testing is generally broken into two broad categories. **Unit tests** are test of an individual section of code, as independently from the rest of the software as possible. These sections of code may be termed methods, functions, subroutines, objects, or procedures depending upon the programming language. **Functional tests** are tests of the entire piece of software.

These tests may be designed to test various aspects of the software including; correct behavior, security, performance, robustness, handling heavy loads, standards compliance, and usability. Occasionally, the same set of tests may be done multiple times to check for compatibility with various operating systems, web browsers, or other interdependent components.

Many of the currently popular software development methodologies focus on **test driven development (TDD)**. This is a process in which unit tests are designed before or while creating the software to independently ensure that each section of code is working correctly. Key tests may even be specified in advance in the design specification documents. This cuts down on the amount of time that will be spent on testing and debugging later. This is key to being able to frequently release new versions of the software with some level of confidence in how well they will work.

Comparison of software development methodologies

Method	Deadline emphasis	Cost emphasis	Quality emphasis	Project size	Release frequency	Planning emphasis	Coordination tightness	Notes
LD	H	H	VL	S	H	L	L	
RAD	H	H	VL	S	H	L	L	
DSDM	H	H	M	S	H	M	L	
Intenet-Speed Dev	H	H	L	S	VH	M	L	
SCRUM	H	H	M	M	H	M	M	
FDD	M	M	M	M	M	H	H	
Agile Unified Process	M	M	M	M	M	H	H	
Evo	M	M	M	M	M	H	H	
MSF	L	M	H	L	L	H	H	
OpenUP	L	L	H	S	L	H	H	
XP	H	L	VH	M	M	H	H	Product quality emphasis
Crystal Methods	L	L	VH	L	L	H	H	Team member utilization
Spiral	L	L	H	L	M	M	M	Experimental projects
PRINCE2	L	M	H	L	L	H	H	Focus on process
JAD		M				H	H	Requirements gathering
Waterfall	L	L	H	L	L	H	H	High cost of rework
PMBOK	L	L	H	L	L	H	H	Standards compliance
RUP	L	L	VH	VL	L	H	H	Docs, training, certs
CMMI	L	L	VH	L	L	H	H	Process improvement
SDLC	VL	VL	VH	VL	VL	VH	VH	Formalized process
Enterprise Unified	VL	VL	H	VL	VL	VH	VH	Long-term projects

The table above gives a sense of how the various software development methodologies are typically characterized. Since every development team tends to adapt these methodologies to their needs, there are probably exceptions to all of these. The following are descriptions of the table columns.

Deadline emphasis (High, Medium, Low, Very Low) – a methodology with a high deadline emphasis is one constructed to meet release deadlines, often frequently, even at the expense of something else such as feature completeness.

Cost emphasis (High, Medium Low, Very Low) – a methodology with a high cost emphasis has a strong focus on keeping costs down, or staying within a fixed cost, even if something else suffers.

Quality emphasis (Very High, High, Medium, Low, Very Low) – a methodology with a high quality emphasis is designed to produce robust, bug free software, even if it goes over time and budget targets to achieve that goal.

Project size (Small, Medium, Large, Very Large) – a small project requires nine or fewer development and testing staff members, medium tens of people, large over fifty, and very large in the hundreds. The majority of software development projects in the world fall in the small category. Also, smaller projects tend to have a shorter development time, from days to a couple years. Some of the largest software development projects in existence are building on a code base where the earliest lines of code were written in the 1950s or 1970s.

Release frequency (Very High, High, Medium, Low, Very Low) – a very high release frequency project will essentially use each days build as a new version available to the public, high weeks or months between versions released to the users, and on down to very low release frequencies that might be 5-10 years between version releases and each version may be only marginally different from the previous version. Note that some methodologies use an iterative implementation process inside the software development group, but are categorized as low release frequency because they less often release new versions to the user community.

Planning emphasis (Low, Medium, High, Very High) – a low planning emphasis is where each component is designed as it is built often without any overall design plan. A very high planning emphasis might involve a couple years of specifications, use cases, algorithm design, etc. before beginning to write the final production code.

Coordination tightness (Low, Medium, High, Very High) – a low coordination tightness is when each developer works independently to build a component without an overall plan for what order components should be built and how the components will interconnect. A very high coordination tightness is typified by a master plan for which developer will work on which component in what order and how all of the components will interconnect.

Why software development projects fail

There are a number of reasons that software development projects fail. Here are the most typical problems to be aware of.

- The most frequent cause of projects being over time or over budget is incomplete or changing requirements. This problem can be minimized by a more formalized requirements gathering process, and by having a member of the development team who is a subject matter expert.
- Feature creep is a process by which more and more functionality is put into the product by developers, managers, and users. This delays the release of products, or worse yet products may be rushed to market before sufficient testing and debugging has occurred. Frequent release development methodologies and test driven development can be good ways to minimize these problems.
- Underestimating labor involved. If there are unknowns in how key components will work, it is best to start small and prototype those sections of the project. There is a natural psychological tendency to think that tasks will be easier than they really are. Once a development group has been together for a while, the manager can figure out how to estimate the expected time to complete work as some factor times what the development staff estimates (2X – 5X are common).
- Lack of funding. If this is a concern, consider using one of the methodologies built around a fixed cost or time.
- Poor knowledge of the technical field. Programmers right out of school can be eager and productive, but years of correcting mistakes as these early career employees learn from experience is a lot more expensive than hiring one person with years of experience in the field.
- Poor knowledge of the market. Even great software products can fail in the market place if they come to market too soon, too late, marketed incorrectly, to a market that is too small, or a market with too much competition. Across all business startups, failure to market the product well enough to bring in a revenue stream that can sustain the company before startup funds are exhausted is the cause of failure far, far more often than failure to create a usable product.

No project management system is best for all projects in all fields with all employees in all companies. A wise choice of software development methodology will make the process significantly less painful, and more likely to successfully accomplish the task within time and budget constraints.

Further reading

http://www.itinfo.am/eng/software-development-methodologies/

http://www.noop.nl/2008/07/the-definitive-list-of-software-development-methodologies.html

http://en.wikipedia.org/wiki/Software_development_process

The Wikipedia articles on the individual methodologies generally give a good, if minimal, overview.

Some notes on Apple product design are at
http://uxmovement.com/resources/8-things-to-know-about-the-company-culture-at-apple/
http://www.businessweek.com/the_thread/techbeat/archives/2008/03/apples_design_p.html
https://developer.apple.com/library/mac/#documentation/userexperience/conceptual/applehiguidelines/UEGuidelines/UEGuidelines.html

An example of Agile methods to create software for the Obama election campaign is in the following article. This illustrates how a hard deadline and well-prioritized requirements can point to a choice of software development methodology. Aspects of both DSDM and RAD are evident in the description of the project. They boldly cut out some best practices based on their vision that the software would not be reused since available data sources and the popularity of internet communication venues will have changed and thus made the software obsolete before the next election.
http://www.drdobbs.com/architecture-and-design/software-development-in-the-obama-campai/240146307