SOFTWARE PROCESS METHODOLOGIES AND A COMPARATIVE STUDY OF VARIOUS MODELS

Mr. S. Manivannan  
Research Scholar  
Anna University of Technology Coimbatore  
E-mail: smanivannan2003@yahoo.com

Dr. S. Balasubramanian  
IPR Consultant & Research Supervisor  
Anna University of Technology Coimbatore  
E-mail: s_balasubramanian@rediffmail.com

ABSTRACT:

The largely growing body of software development organizations implement process methodologies. Many of them are in the defense industry, which in the U.S. requires a rating based on 'process models' to obtain contracts.

The international standard for describing the method of selecting, implementing and monitoring the life cycle for software is ISO 12207.

A decades-long goal has been to find repeatable, predictable processes that improve productivity and quality. Some try to systematize or formalize the seemingly unruly task of writing software. Others apply project management techniques to writing software. Without project management, software projects can easily be delivered late or over budget. With large numbers of software projects not meeting their expectations in terms of functionality, cost, or delivery schedule, effective project management appears to be lacking.

Organizations may create a Software Engineering Process Group (SEPG), which is the focal point for process improvement. Composed of line practitioners who have varied skills, the group is at the center of the collaborative effort of everyone in the organization who is involved with software engineering process improvement.
This paper is dealing about various software process methodologies, advantages / disadvantages and the scenarios / project types where they have to be adopted.

Following are the software development activities to be followed in any software development and maintenance process. There are several models to represent this process.

**PLANNING**

The important task in creating a software product is extracting the requirements or requirements analysis. Customers typically have an abstract idea of what they want as an end result, but not what software should do. Incomplete, ambiguous, or even contradictory requirements are recognized by skilled and experienced software engineers at this point. Frequently demonstrating live code may help reduce the risk that the requirements are incorrect.

Once the general requirements are gleaned from the client, an analysis of the scope of the development should be determined and clearly stated. This is often called a scope document.

Certain functionality may be out of scope of the project as a function of cost or as a result of unclear requirements at the start of development. If the development is done externally, this document can be considered a legal document so that if there are ever disputes, any ambiguity of what was promised to the client can be clarified.

**DESIGN**

Domain Analysis is often the first step in attempting to design a new piece of software, whether it be an addition to an existing software, a new application, a new subsystem or a whole new system. Assuming that the developers (including the analysts) are not sufficiently knowledgeable in the subject area of the new software, the first task is to investigate the so-called "domain" of the software. The more knowledgeable they are about the domain already, the less work required. Another objective of this work is to make the analysts, who will later try to elicit and gather the requirements from the area experts, speak with them in the domain's own terminology, facilitating a better understanding of what is being said by these experts. If the analyst does not use the
proper terminology it is likely that they will not be taken seriously, thus this phase is an important prelude to extracting and gathering the requirements. If an analyst hasn't done the appropriate work confusion may ensue: "I know you believe you understood what you think I said, but I am not sure you realize what you heard is not what I meant."[1]

ARCHITECTURE

The architecture of a software system or software architecture refers to an abstract representation of that system. Architecture is concerned with making sure the software system will meet the requirements of the product, as well as ensuring that future requirements can be addressed. The architecture step also addresses interfaces between the software system and other software products, as well as the underlying hardware or the host operating system.

IMPLEMENTATION, TESTING AND DOCUMENTING

Implementation is the part of the process where software engineers actually program the code for the project.

Software testing is an integral and important part of the software development process. This part of the process ensures that bugs are recognized as early as possible.

Documenting the internal design of software for the purpose of future maintenance and enhancement is done throughout development. This may also include the authoring of an API, be it external or internal.

DEPLOYMENT AND MAINTENANCE

Deployment starts after the code is appropriately tested, is approved for release and sold or otherwise distributed into a production environment.

Software Training and Support is important because a large percentage of software projects fail because the developers fail to realize that it doesn't matter how much time and planning a development team puts into creating software if nobody in an organization ends up using it. People are often resistant to change and avoid venturing into an unfamiliar area, so as a part of the deployment phase, it is very important to have training classes for new clients of your software.
Maintenance and enhancing software to cope with newly discovered problems or new requirements can take far more time than the initial development of the software. It may be necessary to add code that does not fit the original design to correct an unforeseen problem or it may be that a customer is requesting more functionality and code can be added to accommodate their requests. It is during this phase that customer calls come in and you see whether your testing was extensive enough to uncover the problems before customers do. If the labor cost of the maintenance phase exceeds 25% of the prior-phases' labor cost, then it is likely that the overall quality, of at least one prior phase, is poor. In that case, management should consider the option of rebuilding the system (or portions) before maintenance cost is out of control.

Bug Tracking System tools are often deployed at this stage of the process to allow development teams to interface with customer/field teams testing the software to identify any real or perceived issues. These software tools, both open source and commercially licensed, provide a customizable process to acquire, review, acknowledge, and respond to reported issues.

WATERFALL MODEL

The waterfall model is a sequential software development process, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design (validation), Construction, Testing and maintenance.

The waterfall development model has its origins in the manufacturing and construction industries; highly structured physical environments in which after-the-fact changes are prohibitively costly, if not impossible. Since no formal software development methodologies existed at the time, this hardware-oriented model was simply adapted for software development.
The waterfall model shows a process, where developers are to follow these steps in order:

1. Requirements specification (AKA Verification or Analysis)
2. Design
3. Construction (AKA implementation or coding)
4. Integration
5. Testing and debugging (AKA validation)
6. Installation (AKA deployment)
7. Maintenance

After each step is finished, the process proceeds to the next step, just as builders don't revise the foundation of a house after the framing has been erected.

There is a misconception that the process has no provision for correcting errors in early steps (for example, in the requirements). In fact this is where the domain of requirements management comes in, which includes change control. The counter argument, by critics to the process, is the significantly increased cost in correcting problems through introduction of iterations. This is also the factor that extends delivery time and makes this process increasingly unpopular even in high risk projects.
ITERATIVE MODEL

Iterative development prescribes the construction of initially small but ever larger portions of a software project to help all those involved to uncover important issues early before problems or faulty assumptions can lead to disaster. Iterative processes are preferred by commercial developers because it allows a potential of reaching the design goals of a customer who does not know how to define what they want.

Agile software development processes are built on the foundation of iterative development. To that foundation they add a lighter, more people-centric viewpoint than traditional approaches. Agile processes use feedback, rather than planning, as their primary control mechanism. The feedback is driven by regular tests and releases of the evolving software.

XP: EXTREME PROGRAMMING

Extreme Programming is successful because it stresses customer satisfaction. Instead of delivering everything you could possibly want on some date far in the future this process delivers the software you need as you need it. Extreme Programming empowers your developers to confidently respond to changing customer requirements, even late in the life cycle.

Extreme Programming emphasizes teamwork. Managers, customers, and developers are all equal partners in a collaborative team. Extreme Programming implements a simple, yet effective environment enabling teams to become highly productive. The team self-organizes around the problem to solve it as efficiently as possible.

Extreme Programming improves a software project in five essential ways; communication, simplicity, feedback, respect, and courage. Extreme Programmers constantly communicate with their customers and fellow programmers. They keep their design simple and clean. They get feedback by testing their software starting on day one. They deliver the system to the customers as early as possible and implement changes as suggested. Every small success deepens their respect for the unique contributions of each and every team member. With this foundation Extreme Programmers are able to courageously respond to changing requirements and technology.
In XP, the phases are carried out in extremely small (or "continuous") steps compared to the older, "batch" processes. The (intentionally incomplete) first pass through the steps might take a day or a week, rather than the months or years of each complete step in the Waterfall model. First, one writes automated tests, to provide concrete goals for development. Next is coding (by a pair of programmers), which is complete when all the tests pass, and the programmers can't think of any more tests that are needed. Design and architecture emerge out of refactoring, and come after coding. Design is done by the same people who do the coding. (Only the last feature — merging design and code — is common to all the other agile processes.) The incomplete but functional system is deployed or demonstrated for (some subset of) the users (at least one of which is on the development team). At this point, the practitioners start again on writing tests for the next most important part of the system.

**ISO**

ISO 9000 describes standards for formally organizing processes with documentation.

ISO 15504, also known as Software Process Improvement Capability Determination (SPICE), is a "framework for the assessment of software processes". This standard is aimed at setting out a clear model for process comparison. SPICE is used much like CMMI. It models processes to manage, control, guide and monitor software development. This model is then used to measure what a development organization or project team actually does during software development. This information is analyzed to identify weaknesses and drive improvement. It also identifies strengths that can be continued or integrated into common practice for that organization or team.

**CMMI**

The Capability Maturity Model Integration (CMMI) is one of the leading models and based on best practice. Independent assessments grade organizations on how well they follow their defined processes, not on the quality of those processes or the software produced. CMMI has replaced CMM.
SIX SIGMA

Six Sigma is a methodology to manage process variations that uses data and statistical analysis to measure and improve a company's operational performance. It works by identifying and eliminating defects in manufacturing and service-related processes. The maximum permissible defects is 3.4 per one million opportunities. However, Six Sigma is manufacturing-oriented and needs further research on its relevance to software development.

FORMAL METHODS

Formal methods are mathematical approaches to solving software (and hardware) problems at the requirements, specification and design levels. Examples of formal methods include the B-Method, Petri nets, Automated theorem proving, RAISE and VDM. Various formal specification notations are available, such as the Z notation. More generally, automata theory can be used to build up and validate application behavior by designing a system of finite state machines.

Finite state machine (FSM) based methodologies allow executable software specification and by-passing of conventional coding (see virtual finite state machine or event driven finite state machine).

Formal methods are most likely to be applied in avionics software, particularly where the software is safety critical. Software safety assurance standards, such as DO178B demand formal methods at the highest level of categorization.

AGILE MODEL

Agile software development refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. The term was coined in the year 2001 when the Agile Manifesto was formulated.

Agile methods generally promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices that allow for rapid delivery of high-quality software, and a business approach that aligns
development with customer needs and company goals. Conceptual foundations of this framework are found in modern approaches to operations management and analysis, such as lean manufacturing, soft systems methodology, speech act theory (network of conversations approach), and Six Sigma.

COMPARISON WITH OTHER METHODS

Agile methods are sometimes characterized as being at the opposite end of the spectrum from "plan-driven" or "disciplined" methods. This distinction is misleading, as it implies that agile methods are "unplanned" or "undisciplined". A more accurate distinction is that methods exist on a continuum from "adaptive" to "predictive". Agile methods lie on the "adaptive" side of this continuum.

Adaptive methods focus on adapting quickly to changing realities. When the needs of a project change, an adaptive team changes as well. An adaptive team will have difficulty describing exactly what will happen in the future. The further away a date is, the more vague an adaptive method will be about what will happen on that date. An adaptive team can report exactly what tasks are being done next week, but only which features are planned for next month. When asked about a release six months from now, an adaptive team may only be able to report the mission statement for the release, or a statement of expected value vs. cost.

Predictive methods, in contrast, focus on planning the future in detail. A predictive team can report exactly what features and tasks are planned for the entire length of the development process. Predictive teams have difficulty changing direction. The plan is typically optimized for the original destination and changing direction can cause completed work to be thrown away and done over differently. Predictive teams will often institute a change control board to ensure that only the most valuable changes are considered.

Agile methods have much in common with the "Rapid Application Development" techniques from the 1980/90s as espoused by James Martin and others.
CONTRASTED WITH OTHER ITERATIVE DEVELOPMENT METHODS

Most agile methods share other iterative and incremental development methods' emphasis on building releasable software in short time periods. Agile development differs from other development models: in this model time periods are measured in weeks rather than months and work is performed in a highly collaborative manner. Most agile methods also differ by treating their time period as a timebox.

CONTRASTED WITH THE WATERFALL MODEL

Agile development has little in common with the waterfall model. As of 2004, the waterfall model is still in common use. The waterfall model is the most structured of the methods, stepping through requirements-capture, analysis, design, coding, and testing in a strict, pre-planned sequence. Progress is generally measured in terms of deliverable artifacts: requirement specifications, design documents, test plans, code reviews and the like.

The main problem with the waterfall model is the inflexible division of a project into separate stages, so that commitments are made early on, and it is difficult to react to changes in requirements. Iterations are expensive. This means that the waterfall model is likely to be unsuitable if requirements are not well understood or are likely to change in the course of the project.

Agile methods, in contrast, produce completely developed and tested features (but a very small subset of the whole) every few weeks. The emphasis is on obtaining the smallest workable piece of functionality to deliver business value early, and continually improving it and adding further functionality throughout the life of the project. If a project being delivered under the waterfall method is cancelled at any point up to the end, there is nothing to show for it beyond a huge resources bill. With the agile method, being cancelled at any point will still leave the customer with some worthwhile code that has likely already been put into live operation.

In this respect, agile critics may assert that these features are not placed in context of the overall project, concluding that, if the sponsors of the project are concerned about completing certain goals with a defined timeline or budget, agile may not be appropriate.
Proponents of agile development counter that adaptations of Scrum[9] show how agile methods are augmented to produce and continuously improve a strategic plan.

Some agile teams use the waterfall model on a small scale, repeating the entire waterfall cycle in every iteration. Other teams, most notably Extreme Programming teams, work on activities simultaneously.

CONCLUSION

Of all the models, Agile model is highly recommended for the following reasons.

1. Revenue

The iterative nature of agile development means features are delivered incrementally, enabling some benefits to be realized early as the product continues to develop.

2. Speed-to-market

Research suggests about 80% of all market leaders were first to market. As well as the higher revenue from incremental delivery, agile development philosophy also supports the notion of early and regular releases, and 'perpetual beta'.

3. Quality

A key principle of agile development is that testing is integrated throughout the lifecycle, enabling regular inspection of the working product as it develops. This allows the product owner to make adjustments if necessary and gives the product team early sight of any quality issues.

4. Visibility

Agile development principles encourage active 'user' involvement throughout the product's development and a very cooperative collaborative approach. This provides excellent visibility for key stakeholders, both of the project's progress and of the product itself, which in turn helps to ensure that expectations are effectively managed.
5. Risk Management

Small incremental releases made visible to the product owner and product team through its development help to identify any issues early and make it easier to respond to change. The clear visibility in agile development helps to ensure that any necessary decisions can be taken at the earliest possible opportunity, while there's still time to make a material difference to the outcome.

6. Flexibility / Agility

In traditional development projects, we write a big spec up-front and then tell business owners how expensive it is to change anything, particularly as the project goes on. In fear of scope creep and a never-ending project, we resist changes and put people through a change control committee to keep them to the essential minimum. Agile development principles are different. In agile development, change is accepted. In fact, it's expected. Because the one thing that's certain in life is change. Instead the timescale is fixed and requirements emerge and evolve as the product is developed. Of course for this to work, it's imperative to have an actively involved stakeholder who understands this concept and makes the necessary trade-off decisions, trading existing scope for new.

7. Cost Control

The above approach of fixed timescales and evolving requirements enables a fixed budget. The scope of the product and its features are variable, rather than the cost.

8. Business Engagement/Customer Satisfaction

The active involvement of a user representative and/or product owner, the high visibility of the product and progress, and the flexibility to change when change is needed, create much better business engagement and customer satisfaction. This is an important benefit that can create much more positive and enduring working relationships.

9. Right Product

Above all other points, the ability for agile development requirements to emerge and evolve, and the ability to embrace change (with the appropriate trade-offs), the team build the right product. It's all too common in more traditional projects to deliver a "successful" project in IT terms and find that the product is not what was expected,
needed or hoped for. In agile development, the emphasis is absolutely on building the right product.

10. More Enjoyable

The active involvement, cooperation and collaboration make agile development teams a much more enjoyable place for most people. Instead of big specs, we discuss requirements in workshops. Instead of lengthy status reports, we collaborate around a task-board discussing progress. Instead of long project plans and change management committees, we discuss what's right for the product and project and the team is empowered to make decisions. In my experience this makes it a much more rewarding approach for everyone. In turn this helps to create highly motivated, high performance teams that are highly cooperative.

REFERENCES

- Larman, Craig and Basili, Victor R. Iterative and Incremental Development: A Brief History IEEE Computer, June 2003