International Journal of Research in Signal Processing, Computing & Communication System Design
4 (1), 2018, 16-20
http://www.publishingindia.com/ijrsp

A Study on Agile Software Development

T. Sasi Vardhan[1*], V. Anitha[2], S. P. Anand Raj[3] and C. V. P. R. Prasad[4]

[1]Associate Professor, Dept. of CSE, MRECW, Hyderabad, Telangana, India.
Email: sasi.mrecw@gmail.com
[2]Assistant Professor, Dept. of CSE, BVRIT, Narsapur, Medak, Telangana, India.
[3]Professor, Dept. of CSE, MRECW, Hyderabad, Telangana, India.
[4]Professor & HOD, Dept. of CSE, MRECW, Hyderabad, Telangana, India.
*Corresponding Author

Abstract: Agile software development has rapidly gained a lot of interest in the field of software engineering. Agile software development, despite its novelty, is an important domain of research within software engineering discipline. Agile software development methods have caught the attention of software engineers and researchers worldwide. Scientific research is yet scarce, there has been little detailed reporting of the usage, penetration and success of agile methodologies in traditional, professional software development organizations. Agile development focuses on cross functional teams empowered to make decisions, versus big hierarchies and splitting by function. The field is relatively nascent and research is in its initial stages.

Keywords: Agile methodologies, Agile software development, Software engineering.

I. Introduction

System Engineering is the process of analyzing and designing an entire system, including the hardware and the software. A lot of systems today have a mix of hardware and software that is tightly integrated, like modern smartphones, tablets, etc. To create such systems, involve a lot of different disciplines. Software is any set of machine readable instructions that directs a computer's processor to perform specific operations. Computer hardware and software require each other and neither can be realistically used without the other. In the Fig. 1 we see the different phases involved in the Software Development Lifecycle (SDLC).

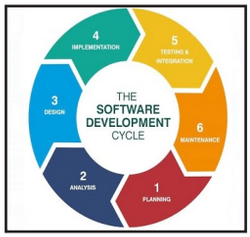

Fig. 1: Software Development Lifecycle (SDLC)

The main parts or phases in the software development process are:

- Planning
- Requirements Analysis
- Design
- Implementation
- Testing
- Deployment and Maintenance

In Fig. 2 we see examples of some of the different activities involved in the different phases of software development.

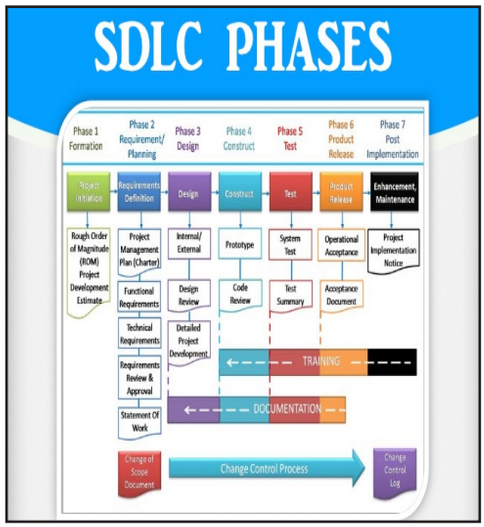

Fig. 2: Activities Involved in the Different Software Phases

Software Development also involves different roles, which are organized in different teams.

Typical roles are:

- Project Manager
- System Architect
- UX Designer

- Programmer, System Developer
- Tester
- Customer

It is crucial that the different roles and teams can work together and collaborate. The Programmer or System Engineer must deal with that there exists hundreds of different Programming Languages. Each language has pros and cons, so it is important to find out which programming language is best suited in each situation. When working with software development it is important to have good tools. The developer needs of course to use a programming language and proper IDE (Integrated Development Environment). In addition, a so-called ALM Tool should be used. ALM is short for Application Lifecycle Management. An ALM tool typically facilitate and integrate things like:

- Requirements Management
- Architecture
- Coding
- Source Code Control (SCC)
- Testing
- Bug Tracking
- Release Management etc.

There exist a lot of such tools, e.g. Team Foundation Server (TFS), Jira, etc. We will take a closer look at TFS in this document. TFS from Microsoft, since it is tightly integrated with Visual Studio. Typically, you need to share the code with others developers or testers in your team or other teams, so it is crucial that you have tools that can be used to share your code, that makes sure that old versions of your code will be stored, and can be restored, etc. Such a system is called a Source Code Control (SCC) system. In Fig. 3 we see a typical software project with different platforms and frameworks involved.

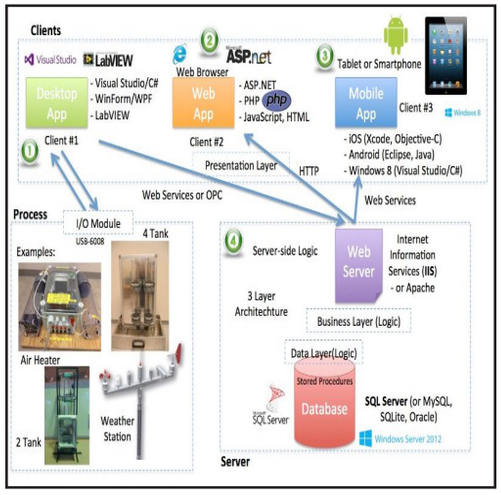

Fig. 3: Typical Software Project with Different Platforms and Frameworks Involved

Typically, software needs to be installed and be running on different devices, such as PCs, tablets, smartphones, etc. We also need to store the data, typically in a database, such as Microsoft SQL Server, MySQL, etc. All these devices and the data also need to communicate with each other over a network, either an internal network (LAN, Local Area Network) or over Internet (WAN, Wide Area Network). All these things make it very complicated to develop, test, deploy and install such systems. That's the reality for a modern software developer.

II. Approaches for Software Development

There are different approaches (Software Development Processes) that deal with these phases, such as:

A. Waterfall Model

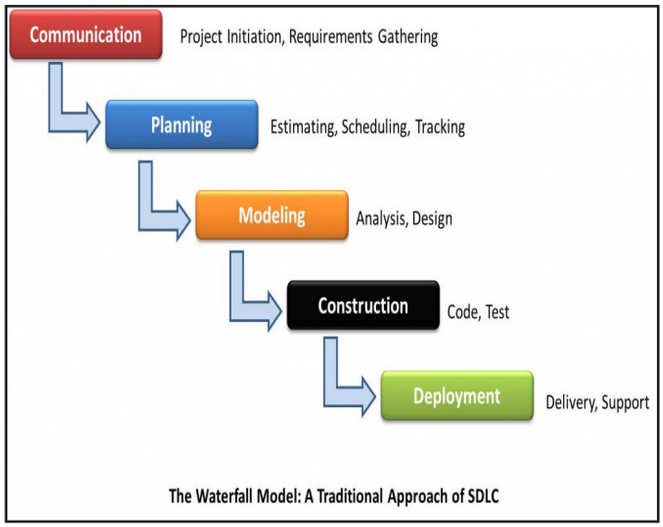

Fig. 4: Waterfall Model

The Waterfall model consists of the following phases:

- Requirements specification (Requirements analysis)
- Software design
- Implementation and Integration
- Testing (or Validation)
- Deployment (or Installation)
- Maintenance

Traditionally with the Waterfall model, you can only start on the next phase when the previous phase is finished. Therefore, it is called the Waterfall method. In practice, there is impossible to create perfect requirements and design before you start implementing the code, so it is common to go back and update these phases iteratively.

B. V Model

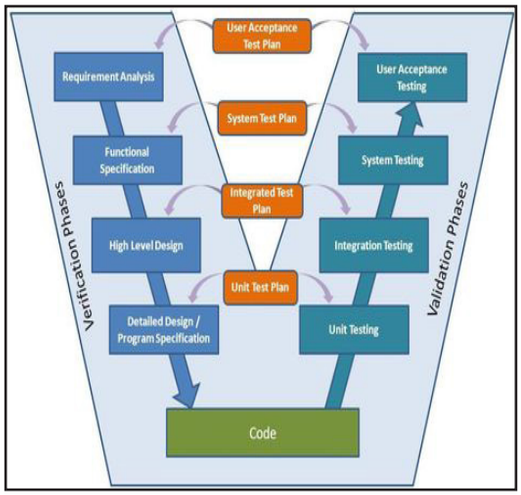

Fig. 5: V-Models

As we see in Fig. 5, the left side is about requirements and design, while the right-side of the model is about testing and validating. It is also known as Verification and Validation model. So there are Verification phases on one side of the .V. and Validation phases on the other side.

III. Introduction to ASD

Agile readiness for motion, nimbleness, activity, dexterity in motion. Agility means the ability to both create and respond to change in order to profit in a turbulent business environment. Companies need to determine the amount of agility they need to be competitive.

Chaordic: Exhibiting properties of both chaos and order. The blend of chaos and order inherent in the external environment and in people themselves, argues against the prevailing wisdom about predictability and planning. Things get done because people adapt, not because they slavishly follow processes which may last from one to four weeks.

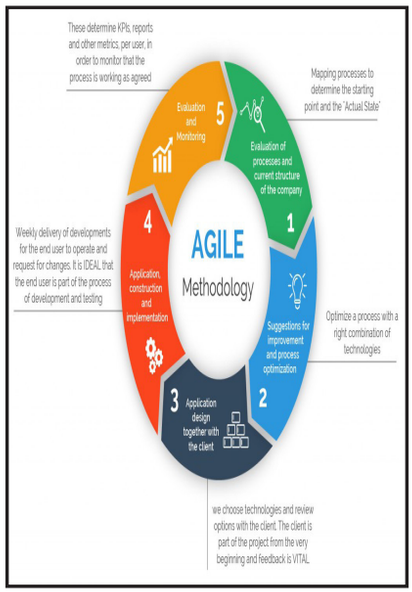

Fig. 6: Agile Methodology

Characteristics of Agile Software Development: Light Weighted methodology, Small to medium sized teams, vague and/or changing requirements, vague and/or changing techniques, Simple design, Minimal system into production.

Agile Alliance: Several individuals, motivated to constrain activities, such that certain outputs and artifacts are predictably produced, around 2000, these notables got together to address common development problems.

IV. Agile Software Development Manifestation

Through this work we have come to the value:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.

That is, while there is value in the items on the right, we value the items on the left more."

Strong player: not necessarily an 'ace;' work well with others, Communication and interacting is more important than raw talent. 'Right' tools are vital to smooth functioning of a team. Start small. Find a free tool and use until you can demo you've outgrown it. Don't assume bigger is better. Flat files before going to a huge database.

Building a team more important than building environment.

- Some managers build the environment and expect the team to fall together.
- Doesn't work.
- Let the team build the environment on the basis of need.

Too much documentation is worse than too little: Take time; more to keep in sync with code.

Short rationale and structure document: Keep this in sync, only highest level structure in the system kept.

Two essentials for transferring info to new team members.

Code is the only unambiguous source of information.

Best way to transfer info- interacts with them.

Rule: Produce no document unless need is immediate and significant.

Customers cannot just cite needs and go away. Best contracts are NOT those specifying requirements, schedule and cost. Become meaningless shortly. Details ideally not specified in contract. With frequent deliverables and feedback, acceptance tests never an issue.

Course of a project cannot be predicted far into the future. Too many variables; not many good ways at estimating cost.

Tempting to create a PERT or GNANT chart for whole project. This does not give novice managers control. But the structure of the chart will degrade. Once plan is made, difficult to change due to momentum and commitment. But rest of plan remains flexible.

V. Agile Principles

The following principles are those that differentiate agile processes from others**.**

Principle 1: The more frequent the deliveries, the higher the final quality.

Principle 2: Welcome Changing Requirements, even late in Development. Agile Processes harness change for the Customer's Competitive Advantage: This is a statement of attitude.

Participants in an agile process are not afraid of change.

Principle 3: Deliver Working Software Frequently: We deliver working software. Deliver early and often. Be not content with delivering bundles of documents, or plans. Don't count those as true deliverables. The goal of delivering software that satisfies the customer's needs.

Principle 4: Business People and Developers Must Work Together Daily throughout the Project: For agile projects, there must be significant and frequent interaction between the customers, developers, and Stakeholders. An agile project must be continuously guided.

Principle 5: Build Projects around Motivated Individuals: An agile project has people the most important factor of success. All other factors, process, environment, management, etc., are considered to be second order effects, and are subject to change if they are having an adverse effect upon the people. Example: if the office environment is an obstacle to the team, change the office environment. If certain process steps are obstacles to the team, change the process steps.

Principle 6: In agile projects, developers talk to each other. The primary mode of communication is conversation.

Principle 7: Agile projects measure their progress by measuring the amount of working software. Agile teams are 30% done when 30% of the necessary functionality is working**.**

Principle 8: An agile project is not run like a 50 yard dash; it is run like a marathon Rather they run at a fast, but sustainable, pace. Running too fast leads to burnout, shortcuts, and debacle. Agile teams pace them. They don't allow themselves to get too tired. They don't borrow tomorrow's energy to get a bit more done today.

Principle 9: Continuous Attention to Technical Excellence and Good Design enhances Agility: High quality is the key to high speed. They do not make messes and then tell themselves they'll clean it up when they have more time.

Principle 10: They don't anticipate tomorrow's problems and try to defend against them today.

Principle 11: Responsibilities are not handed to individual team members from the outside. Agile team members work together on all project aspects. Each is allowed input into the whole. The team shares those responsibilities and each team member has influence over them.

VI. Plan-Driven and Agile Development

Plan-driven Development: A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.

Agile Development: Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.

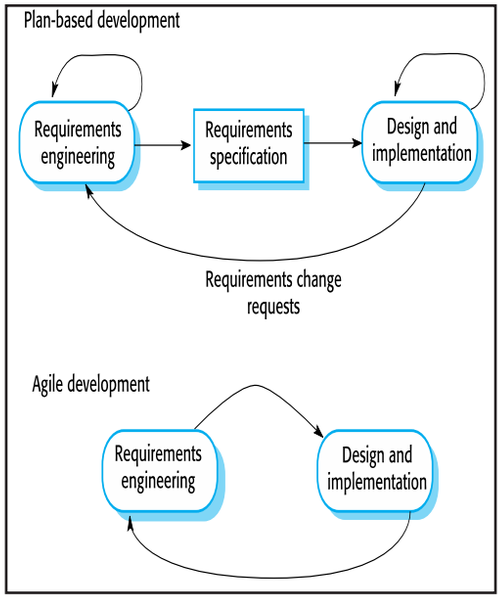

Fig. 7: Plan Based and Agile Based Development

VII. Conclusion

Software development methodologies have evolved since the 1970s. Agile software development methods have evoked a substantial amount of literature and debates. However, academic research on the subject is still scarce, as most existing publications are written by practitioners or consultants. The analyzed studies point towards the researchers current concerns with the challenges in Agile Software Development (ASD). The aim of this paper is to attempt to make sense out of the jungle of emerged agile software development methods.

References

[1] ISO/IEC/IEEE 42010:2011, Systems and software engineering - Architecture description.

[2] J. Highsmith, *Agile Project Management: Creating Innovative Products*, 2nd ed.

[3] IEEE Std 1471, IEEE recommended practice for architectural description of software - Intensive systems, October 2000.

[4] N. Rozanski, and E. Woods, *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*, 2nd ed.

[5] Manifesto for Agile Software Development.

[6] V. P. Eloranta, and K. Koskimies, "Lightweight architecture knowledge management for agile software development," Tampere University of Technology, Tampere, Finland.

[7] J. Highsmith, "The great methodologies debate: Part2," *Cutter IT Journal*, vol. 15, 2002.

[8] A. Begel, and N. Nagappan, "Usage and perceptions of agile software development in an industrial context: An exploratory study," Microsoft Research One Microsoft Way Redmond, WA 98052.

[9] A. Agrawal, Md. A. Atiq, and L. S. Maurya, "A current study on the limitations of agile methods in industry using secure Google forms," *Procedia Computer Science*, vol. 78, pp. 291 297, 2016.

[10] V. P. Eloranta, K. Koskimies, and T. Mikkonen, "Exploring ScrumBut - An empirical study of Scrum anti-patterns," *Information and Software Technology*, vol. 74, pp. 194-203, 2016.

[11] P. Gregory, L. Barroca, H. Sharp, A. Deshpande, and K. Taylor, "The challenges that challenge: Engaging with agile practitioners' concerns," *Information and Software Technology*, vol. 77, pp. 92-104, 2016.

[12] V. T. Heikkila, M. Paasivaara, K. Rautiainen, C. Lassenius, T. Toivola, and J. Jarvinen, "Operational release planning in large-scale Scrum with multiple stakeholders - A longitudinal case study at F-Secure Corporation," *Information And Software Technology*, vol. 57, pp. 116-140, 2015.