An Overview of Modern Software Development Methodologies

Krešimir Fertalj, Marija Katić
Faculty of Electrical Engineering and Computing, University of Zagreb, Unska 3, 10000 Zagreb, Croatia
{kresimir.fertalj, marija.katic}@fer.hr

Abstract. *Business activities are rapidly changing nowadays and there are increasingly complex requirements set on programming solutions. That puts traditional software development methods behind and leads to the need for different approaches. Modern approach is called agile. This paper presents the process of software development and the methods that are applied to the process. An overview and comparison of traditional and modern methods of software development is given. Finally, there are some thoughts about modern research of software development methods and their application.*

Keywords. Software process, development methods, agile development

1 Introduction

Software development process comprises some key activities that cannot be avoided and should be implemented in phases. These are analysis, design, implementation, testing and maintenance. Different methods that apply this process have evolved so far. The sequence of given activities actually constitutes the software development life cycle (SDLC). Each method is characterized by its own life cycle which, except for specific techniques, makes the basis for major differences between the methods. Different life cycles have affected some basic aspects that have to be considered discussing each method. Besides the process, these are the structure of a project team, documentation developed during the project, practices applied by a method, software types and tools that can be used.

First of all, this paper gives a short history of development methodologies leading to the modern, i.e. agile software development methodologies. The following chapter presents a comparison of the traditional and modern methodologies. Four agile methodologies (extreme programming, scrum, dynamic system development methods and adaptive software development) are presented and compared according to the above mentioned criteria: process, project team structure, documentation, practices, software types and tools. Finally, there is a conclusion about the new breed of methods and contemporary research concerning them.

2 Why modern software development methodologies?

2.1 The history of emergence of software development methodologies

Numerous models for team and plan software or application development have been used over time. These are so called software development methods or methodologies. According to Avison and Fitzgerald [6], methodology is a collection of procedures, techniques, tools and documentation aids which will help system developers in their efforts to implement a new information system. In accordance with the conditions about 30 years ago and at the same time when software engineering was in inception, business activities were not influenced by frequent changes. It was logical that the development process should be deterministic and predictable. Such models are called traditional. A well known representative is the waterfall model. The process progresses linearly from analysis to the maintenance phase. Project plan is created in the analysis phase and all plans are strictly documented as well as results from the design phase. Developers are supposed to build the system according to the design documentation. They are not supposed to have contact with the end customer, which means that customer's requests can easily be misunderstood.

Endeavouring to understand customer's requirements, new models evolved, such as prototyping and the spiral model, which are iterative. Iterative means to build functional application in short cycles, based on incremental development. Evolutionary prototyping, as part of the spiral development model, tends to clarify misunderstandings between customer and developer through prototypes evaluated by customers. This approach enables refinement of the product in finite number of iterations.

Introducing the iterative and prototyping approach as well as rapid application development (RAD), has resolved some but has also introduced some new problems in software development. Iterative model has introduced a segmentation of the system, but linearity and predictability have remained applied in cycles. Prototyping has improved communication with the customer, while RAD has introduced fast system delivery, trying to avoid changing requirements.

The real modern methodologies are agile, trying to solve problems by changing software engineering approach. Traditional methodologies will not be replaced with the new ones, but it is necessary to rethink the conditions of software development and choose an appropriate method.

2.2 Agile software development methodologies

Although the emergence of agile methodologies relates to the time before 2000, their expansion was indicated in 2001 by the group of software consultants and experts. They created the Agile manifesto which recommends values and principles common for all agile methods. The most important values are [7]:

- individuals and interactions over processes and tools
- working software over comprehensive documentation
- customer collaboration over contract negotiation
- responding to change over following a plan;

It is easier to respond to conditions in the dynamic global market if applying these values. There are several items that need to be discussed in the context of the new methodological approach: process, project team structure, documentation, practices, software types and tools.

Process: Agile is focused on flexibility and it is not linear and deterministic. Basically, it tends to develop software in short time periods, actually iterations, doing refinement and reprioritization in every iteration. Iteration produces a new version of software which means that agile is incrementally oriented. It might seem like this kind of developing software is messy, because there is no fixed plan at the beginning of the project, but that is not true. The fixed plan does not exist, but the plan is continuously updated as the project progresses.

Project team structure: Project teams have to adopt different rules and apply different practices. These arguments lead to the necessity for a new team organization. For example, it can be the case when people work in pairs and use the same computer to solve difficult issues.

Documentation: Modern approach has changed the way the documentation is developed. The documentation is written throughout almost all phases of the life cycle and because of that it is subject to often changes. The significance of documentation is definitely reduced and therefore some agile methods produce little documentation. The most important fact which replaces this absence is the human being and ability to communicate and share knowledge.

Practices: There are sets of practices that agile methods use. It is important to state them because they refer to the process itself.

Software types: Agile methodology consists of methods that basically apply to different kinds of software, e.g. commercial, research, web or some other type. Suggestions as well as experiences concerning the most used methods are presented later in this work.

Tools: Agile movement does not necessarily require tools. It is possible to be agile and employ no more technology than a command line interface, a unit tester and some index cards on which to write requirements [13]. However, a lot of tools have evolved in order to accelerate the process cycle. Agile teams focus their investments on tools that the whole team can use, layering agile project management tools on top of testing tools on top of build management tools on top of software configuration management tools [9].

2.3 The comparison of the traditional and modern methodologies

Agile methods, with values and practices together, bring a new way of developing applications. However, a single methodology cannot work for the whole spectrum of different projects. Project management should identify the specific nature of project at hand and then select the best applicable development methodology [1]. Mostly, experts in this area share the same opinion that there is a need for both, traditional and modern methodologies, as there is no onesize-fits-all software development model that suits all imaginable purposes [1].

The main difference between agile and traditional approaches is that traditional methods tend to develop working software at the end of the process, while agile do it continuously, with support of continuous integration and test driven development. Test driven development is so important that removing a defect is the only type of work that takes priority over any new features, functionality or production [3]. Table 1 describes the differences between traditional and modern methodologies according to previously defined criteria.

Table 1. A comparison of traditional and modern methodologies

	Process	Practices	Project team structure	Documentation	Software types	Tools
Traditional approach	Heavyweight: defined plan in the beginning of the process which is frozen at that point – plan driven; Linear and predictive.	Not defined. Construct gradually and deliver once.	Distributed or collocated teams. Mostly big teams with strictly defined roles.	Well documented. First document then develop according to documents that were frozen at some point of time.	Any type, but with an increased risk; engineering and scientific software types; Big or small projects.	Different tool for each phase, with later integration. Eclipse, Toad, Microsoft Project.
Modern agile approach	Lightweight: No frozen plan, but planning throughout the cycles – planning and test driven; Non-linear and adaptive; Incrementally oriented	Seven basic [3]: self-organizing team, deliver frequently, plan to learn, communicate powerfully, test everything, measure value and clear the path	Rather small than big teams. Rather collocated than distributed teams.	Little or no documentation. Focused on tactic knowledge – sharing knowledge between team members.	Rather business software types with variable requirements. Rather small than big projects.	Integration tools from the beginning of the process. Microsoft Visual Studio Team System. Spreadsheets and wiki.

3 The most known agile methodologies

Agile methods that appeared first are eXtreme Programming, Crystal methods, Adaptive Software Development, Scrum and Dynamic Systems Development Method. Feature Driven Development, Lean Development, Open Source Software Development and others evolved afterwards. Four of them, the most frequently used, are discussed below.

3.1 Extreme Programming – XP

Extreme programming designated the expansion of agile software development methodologies. It has evolved from the traditional planning approach moving towards an adaptation approach. First of all, it is focused on the developer who makes technical decisions, while the customer makes business decisions. This is achieved by intensive customer and developer interaction. Extreme programming introduces a new way of applying some existing practices. XP strives for simplicity and not investing into future unless immediately needed [8].

XP life cycle is based on the evolutionary prototyping and consists of six phases: exploration, planning, iterations to release, productionizing, maintenance and death.

The process begins with the exploration phase where the customer writes out the story cards and iteratively continues to the planning phase. Developer and customer both use story cards. According to them, developers plan the time needed for their realization and customers do the prioritization and reprioritization. Development is time boxed and performs in increments from fifteen minutes to a couple of hours. It allows for reprioritization within a time box.

XP as a high-discipline methodology that calls for tight adherence to strict coding and design standards, strong unit test suites that must pass at all times, good acceptance tests, constant working in pairs, vigilance in keeping the design simple, and aggressive refactoring [4]. XP is a test driven method and the developer is responsible for unit tests, while the customer is responsible for acceptance or functional tests. Unit tests verify the functionality of units of code like classes or components in object oriented development. XP is considered one of the lightest of the Agile approaches for managing the projects and therefore there is lot of research on blending these methods with the other agile methods such as Scrum, DSDM or Crystal methods.

Recently, XP has been used in combination with the other agile methods such as Scrum or ASD. A lot of case studies have been done in the area of XP applicability on web projects showing that XP can be adopted for such projects, but that is not trivial and it can easily fail if not adopted well [14].

3. 2 Scrum

The Scrum method has evolved by recognizing that fundamental empirical changes cannot be solved with the traditional approach. The term of Scrum appeared in 1986 when H. Takeuchi and I. Nonaka observed that small, cross-functional teams are often the best performers, and likened these teams to the Scrum formation in rugby [10]. That is adaptive, fast, self-organizing method focused on project management originating from Japan. It does not include specific system implementation techniques.

Scrum life cycle is based on time boxed development and consists of three phases: pre-game, development and post-game. In pre-game phase there is a constantly updated list of requirements named Product Backlog. Development phase is an agile part of Scrum method [1] in which the system is being developed throughout iterative cycles so called 30 Day Sprints. Integration, testing and other activities required for system delivery are accomplished in the post-game phase.

Because it is concentrated on management, this method is successfully used in combination with extreme programming which is technically oriented. It is better documented than XP, but still not too much. There is a person - technical writer who is responsible for documentation writing. He or she follows the development from scratch and keeps an eye on the big picture - that is, how all the pieces fit together [10].

Schwaber and Beedle [1] identify two types of situations in which Scrum can be adopted: an existing project and a new project. These projects are usually small, but it is possible to apply it in big structures if teams are reduced and isolated. According to the recent research on project for a complex Integrated Library System (ILS) similar to a vertical market ERP system, where over 50 developers from U.S., Canada and Russia were involved, the Scrum methodology was used [12]. This practice showed that distributed teams can be very successful if they follow the Scrum principles. Some of the best practices for distributed Scrum observed on the ILS project are the daily Scrum meetings of all developers from multiple sites, hourly automated build from one central repository and seamless integration of XP practices like pair programming with Scrum [12].

3.3 Dynamic Systems Development Method – DSDM

Non-profit organization DSDM Consortium has developed a framework for RAD development because of high increase of various development tools what had changed the development process. The main idea is to determine the time and resources and subsequently adapt the system functionalities, so that it would not be done more than could be done.

DSDM life cycle is based on evolutionary prototyping and consists of five phases: feasibility study, business study, functional model iteration, design and build iteration and implementation. The first two phases are sequential and done only once, while the others are iterative and incremental [1].

DSDM is focused on business processes and therefore is successfully used with XP that puts emphasis on programming. Both DSDM and XP include continuous user involvement resulting in more precise user requirements. Combining the two, gives a controlled framework with robust programming practices [5]. In an organisation already familiar with Scrum but struggling to fit development into a more traditional corporate culture, a blending of the two approaches might be the best solution [5].

DSDM is also better documented than XP because each life cycle phase produces specific documents, but still the documentation is absent. This method is appropriate for small and large projects where the accent is on speed. When considering application domain, Stapleton [1] observes that DSDM is more easily applied to business systems than to engineering or scientific applications.

3.4 Adaptive Software Development – ASD

ASD has evolved as an attempt to solve the problem of developing complex and large systems where is necessary to integrate markets, organizations, development and customers. Fundamentally, ASD is about "balancing on the edge of chaos" – its aim is to provide a framework with enough guidance to prevent projects from falling into chaos, but not too much, which could suppress emergence and creativity [1].

ASD life cycle is based on evolutionary prototyping and consists of three phases: speculate, collaborate and learn. Speculation is the discussion of what is to be done in an iteration. Collaboration means component based development through adaptive development cycles. A review and a preparation for the next phase are being done in the learning phase. Each iteration, called adaptive cycle, has the following properties [8]:

- it is mission-driven, based on the project vision;
- it is component rather than task-based (result-driven);
- it is limited in time;
- each time-box is only one iteration in a larger set of iterations;
- it is risk-driven;
- it is change-tolerant.

This method is appropriate for complex and big systems that are influenced by often changes, but it might be most effective in combination with other methods. For example, because ASD is focused on collaboration practices, it is successfully used with XP which puts emphasis on software development practices.

ASD uses a new programming approach to develop software that can adapt to an environment of rapidly changing user requirements. This is adaptive programming as a special case of aspect oriented programming. In contrary to object oriented programming, which encapsulates data and functionality in classes where applications may suffer from frequent changes under class hierarchy, adaptive programming encapsulates class hierarchies using traversal strategies and visitors so enabling the applications to have an interface to the class hierarchy [2].

4 The comparison of agile methodologies

Although all agile methods basically apply some common concepts such as intensive communication with the customer, test driven development, iterative and incremental development, and minimalistic documentation, Table 2 shows differences among some of them. Not all of them are applicable to the same project types, but they can be combined, allowing for wider usage.

There is some research indicating that methodologies may learn from each other. D. Riehle [8] has done the research comparing ASD and XP on the value system; actually a system of beliefs, what constitutes the fundamental aspects of software development. He has made the basis for comparing further methodologies with ASD and XP and with each other. D Strode [11] has also performed research on combinations of agile methods. She proposed DSDM as a framework that can be used with any other agile method, XP with Scrum and Crystal, and that Scrum, ASD and Crystal can be used with any techniques as long as they achieve the goals of the methodology.

The agile methodology is definitely the future of software development, and as Steve McConnell said: "The future of Agile with a capital "A" is the same as the past of Object Oriented or Structured".

Table 2. A comparison of agile methodologies

	Process	Project team structure	Documentation	Practices	Software types	Tools
XP (Kent Beck, 1999)	Evolutionary prototyping – iterative and incremental; short cycles; time boxed; test driven	Small to medium collocated teams from 3 to 20 members; 7 possible roles	Absence of documentation is replaced with tactic knowledge and different CASE tools	14 practices; The most important are: pair programming, test-driven development, simple design, coding standards and on-site customer	Object oriented projects; web applications	Refactoring tools for Java, C++, relational databases, object database, concurrent systems. CM tools for fast builds; unit testing framework e.g. Junit, HttpUnit; planning tools e.g. Xplanner
Scrum (Ken Schwaber, 1999)	Evolutionary delivery (time boxed – 30 day Sprint); iterative and incremental;	Small teams, but recently applied to big distributed teams; 6 possible roles, scrum master is the most responsible – as project manager	Each iteration produces a document; it is developed from the bottom up; written by technical writer; it is not emphasised	Product Backlog, effort estimation, Sprint, Sprint planning meeting, Sprint backlog, daily scrum and Sprint review meeting	Object oriented projects; web applications; business oriented applications	Integrated suite of lifecycle tools e.g. Conchango Scrum plug-in for Microsoft Visual Studio Team System; planning tools e.g. Xplanner; lifecycle management tools e.g. ScrumWorks
DSDM (DSDM Consortium, 1995)	Evolutionary prototyping (time boxed); iterative and incremental; test driven	Small teams from 2 to 6 members; 15 possible roles: ambassador, visionary, advisor…	Each iteration produces specific documents, but not necessarily	9 practices; some of them: active user involvement, empowered teams; frequent deliveries; continuous testing;	Large-scale enterprise systems; any analysis, design and build techniques; eBusiness, eCommerce	CASE tools, rapid development tools, suite of lifecycle tools for Eclipse e.g. composer plug-in (new)
ASD (James A. Highsmith III, 2000)	Evolutionary prototyping (time boxed); iterative and incremental; risk driven;	Small teams, but structure not completely defined; no need for collocated teams; some roles: executive sponsor, customer, facilitator	Each iteration produces specific documents, but not necessarily	Not focused on practices, but there are some: component based development, customer focus group reviews	systems that involve interaction with an external environment that are hard to model accurately; adaptive programming approach	Project management and collaboration tools, rapid development tools; Demeter tools for aspect-oriented programming

5 Conclusion

Agile methods provide different ways of developing software. In lots of cases they proved to be more successful than traditional ones, but the problem is in increased emergence of different methods, and still there is not sufficient research on their application. Therefore, they are often exposed to criticism, especially due to absence of documentation and due to their flexible process structure.

This paper gives a brief overview of agile methodology as a whole comparing it with the traditional one, and also a brief overview of four agile methods and their mutual comparison. A brief comparison between agile and traditional approaches demonstrates opposite directions within all of the compared criteria. A comparison of agile methods proves similarity in evolutionary development, small teams, and reduction of documentation. On the other hand, software development techniques and supporting software tools make the difference between these methods.

6 References

[1] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen & Juhani Warsta: Agile software development mehods: Review and Analysis, VTT Publications 478, 2002

[2] Adaptive programming, available at http://www.datakon.cz/datakon04/d04_it_kroha.pdf, Accessed: 10th 2008

[3] Mishkin Berteig: The Seven Core Practices of Agile Work, available at http://www.agileadvice.com/archives/2006/09/practices_of_ag.html, Accessed: 10th June 2008

[4] Alistair Cockburn: Agile Software Development, Addison Wesley, Boston, USA, 2000

[5] DSDM Consortum, available at http://www.dsdm.org/, Accessed: 10th June 2008

[6] Juhani Iivari, Jari Maansaari: The usage of systems development methods: are we stuck to old practices?, Information and Software Technology, 1998, Linnanmaa, Finland, 1998, pp. 501-510

[7] Manifesto for Agile Software Development, available at http://agilemanifesto.org/, Accessed: 10th June 2008

[8] Dirk Riehle: A comparison of the value systems of adaptive software development and extreme programming: how methodologies may learn from each other, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2001, pp. 35-50

[9] Carey Schwaber, Gene Leganza, David D'Silva: The Truth About Agile Processes, available at http://www.forrester.com/Research/Document/0,7211,41836,00.html, Accessed: 10th June 2008

[10] Christine M. Sigman, Adapting to Scrum: Challenges and Strategies, available at http://www.stc.org/intercom/PDFs/2007/20070708_16-19.pdf, Accessed: 10th June 2008

[11] Diane Strode: An analytical comparison of five agile methods and an investigation of their target environment, available at http://www.slideshare.net/StrDia/agile-methods-101-bar-camp-2007-196957, Accessed: 10th June 2008

[12] Jeff Sutherland, Anton Viktorov, Jack Blount, Nikolai Puntikov: Distributed Scrum: Agile Project Management with Outsourced Development Teams, 40th Annual Hawaii International Conference on Software Systems, Big Island, Hawaii, 2007, pp. 274a-274a

[13] Jack Vaughan: Agile tools for agile development, available at http://searchsoftwarequality.techtarget.com/news/article/0,289142,sid92_gci1287345,00.html, Accessed: 10th June 2008

[14] XP123 - Exploring Extreme Programming, available at http://www.xp123.com/, Accessed: 10th June 2008