A Contrast and Comparison of Modern Software Process Models

Pankaj Vohra
Computer Science & Engineering Department
Thapar University, Patiala

Ashima Singh
Computer Science & Engineering Department
Thapar University, Patiala

ABSTRACT
Software Processes are the lifeline of any Software Development Model. Software Processes decide the survival of a particular software development model in the market as well as in software organization. The set of processes those proved to be effective and efficient for software development in one organization may or may not be followed in another organization. That is other organization finds another approach for software development more convenient to work with. This paper explains the progression and remarkable change in Software Processes and their respective models. It also summarizes a contrast of classical software processes with Agility and CBSE.

General Terms
Software Process Models, Software Engineering

Keywords
Software Processes, Agile Development, Change Driven Process Models, Iterative Process, Extreme Programming, Component Based Software Engineering, Software Process Improvement.

1. INTRODUCTION

1.1 Software Process Models
The primary function of software development process models is to "determine the order of the stages involved in software development and evolution and to establish the transition criteria for progressing from one stage to the next" [1]. In history various models were proposed. Figure 1 illustrates the evolution of process models in the past decades.

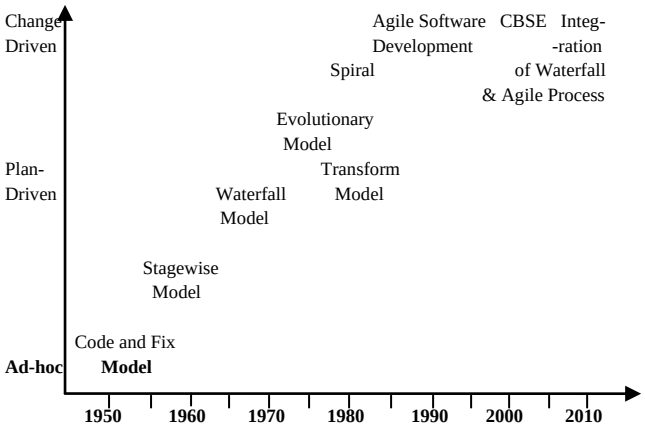

Fig 1: Progression of Process Models

It has also been suggested that the evolution of software development models originates from the problems of ad hoc programming that, at first, led towards traditional plan-driven models and towards iterative change-driven models of software development. The original meaning of the Latin term .ad hoc, refers to a methodology that has been designed for a special purpose (ad hoc = for the purpose of). However, in this context, as often in software engineering literature [2]. The term "ad hoc' is used to refer to the low degree of methodological discipline.

2. PLAN-DRIVEN MODELS FOR SOFTWARE DEVELOPMENT
The plan-driven approaches of software development have been defined as document-driven, code-driven, and traditional process models [1]. As the names suggest, a common feature for the plan-driven process models is their emphasis on defining the scope, schedule, and costs of the project upfront including, for example, an early fixing stage and extensive documentation of the end product requirements. One common characteristic could also be the recurrence of the software development phases only once during the development process, i.e., with only hints of iterativity [3]. The two-step process model of code-and-fix, used in the early days of software development, resulted in difficulties that necessitated explicit sequencing of the phases of software development [1]. In particular, the need to design prior to coding, to define requirements prior to design, and the need for early preparation for testing and modification were identified [1]. One of the first models to rise to that challenge was the stagewise model as early as in the middle of the 1950s [4]. This model evolved from the problems caused by the increasing size of software programs, which could not be handled by a single programmer [4]. In 1968, the NATO Science Committee held a software engineering conference in Garmisch, Germany, where the software crisis, or software gap, was discussed (NATO Science Committee 1969). A standardization of the software development process with an emphasis on quality, costs, and development practices was the key recommendation of the conference [6].

Soon after this, as refinement of the stepwise model, the waterfall model was introduced. The early version of the waterfall model was introduced in 1970 [5] and it has since evolved into a concept consisting of the sequential phases of requirements analysis, design, and development [4]. According to Boehm [1], the waterfall model provided two main advances over the stepwise model: it introduced prototyping to parallel the stages of requirements analysis and design, and provided feedback loops between the sequential stages. It should also be noted that, already in the early

waterfall model [5], it had been realized that it might be necessary to first build a pilot model of the system, i.e., to conduct two cycles of development and to obtain feedback to adjust the model. Thus, hints of iterativity in the model can be seen yet .this iterative feedback-based step has been lost in most descriptions of this model, although it is clearly not classic IID. [7]. Today, the waterfall model has been adopted for most software acquisition standards in government and industry [1]. While the waterfall model has solved various core problems in software development, it also includes features not appropriate for every software development context [1]. One central problem of the waterfall model has been identified as its .emphasis on fully elaborated documents as completion criteria for early requirements and design phases [1].

It can be argued that the plan-driven models of software development can and should be applied in a dynamic way by repeating the phases or even the entire process, if necessary. However, the original purpose of these process models was not to welcome changes during the development, but rather to try to fix factors, such as scope, time and money, up-front in order to eliminate change which was considered a risk factor.

3. ITERATIVE CHANGE-DRIVEN MODELES FOR SOFTWARE DEVELOPMENT

The software development models, developed after the waterfall model, seem to have the common aim of enabling, at least to some degree, the evolution of product requirements during the process of software development. This contributed one main modification to the earlier software development models: the adoption of the iterative and incremental approach. Iterative development refers to the overall lifecycle model in which the software is built in several iterations in sequence [8]. According to [8], each iteration can be considered as a mini-project in which the activities of requirements analysis, design, implementation and testing are conducted in order to produce a subset of the final system, often resulting in internal iteration release. An iteration release has been defined as "a stable, integrated and tested partially complete system" [8]

A development approach where the system is developed in several iterations is called iterative and incremental development (IID), yet it is often referred to as iterative development. [8].

Even though agile software development has recently brought the IID approach of developing software into the spotlight, the history of these approaches is, in fact, considerably longer [7]. Among the first models that focused on increasing the possibility of determining product improvements throughout the development process, was the evolutionary development (Evo) model. This concept was first introduced in 1981 [9] and has been expanded by Gilb [10], [11].

The spiral model of the late 1980s [1] typically consists of four iteratively repeatable steps: 1) determining the objectives, alternatives, and constraints, 2) evaluating alternatives, and identifying and resolving risks, 3) development and verification, and 4) planning the next phase. [1] Defined the spiral model as a risk-driven approach for software development.

Agile software development, which emerged in the mid-1990s, can also be classified as an iterative and change-driven software development approach. It could be argued that at present there is no common agile process model with specified phases, but there is rather a set of fundamentals [12] common to the methods claiming to be agile. However, Extreme Programming (XP) [14], which is probably the best-known among the first agile methodologies, contains an underlying process model for agile software development that has been adopted and adapted by its successors. Figure 2 illustrates how Beck [13] has compared the agile development model of XP with the waterfall model and with the iterative processes.

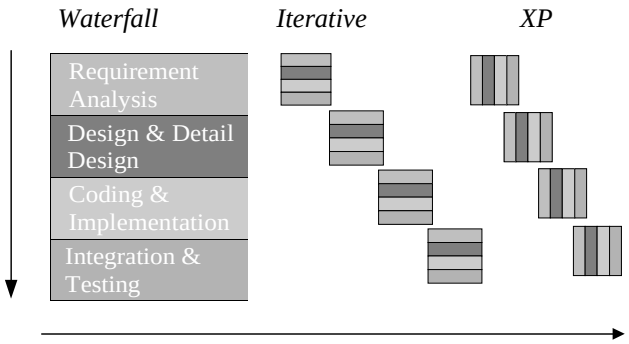

Fig 2: A Contrast: Waterfall, Iterative, Agility and XP

According to [13], XP aims at combining the activities of analysis, design, implementation and testing, a little at a time, throughout the entire software development process. The common feature of agile methods is the recognition that software development cannot be considered to be a defined process, but rather an empirical (or nonlinear) one due to the constant changes that are welcomed during the development of the software product [16].

4. HISTORY OF AGILE SOFTWARE DEVELOPMENT

Agile methodologies were emerged in the mid- 1990s, when software methodologies and techniques such as Extreme Programming (XP) [13], Scrum [15], eXtreme testing [18], Crystal Family of Methodologies [19], Dynamic Systems Development Method (DSDM) [20], Adaptive Software Development (ASD) [17], and Feature-Driven Development (FDD) [21] began to emerge. The emergence of agile methodologies is defined in more detail in [22].

In software development, the agile "movement" was launched in 2001 when the various originators and practitioners of these methodologies met to identify the common aspects of these methods that both combined old and new ideas, and clearly shared some particular ideologies in common. As a result, the Manifesto for Agile Software Development was drafted and the term "agile" was chosen to combine the methods and techniques that would share the values and principles of agile software development. The values and principles of the Agile Manifesto [12] set out the central elements of agility that should be embedded in any method claiming to be agile.

5. COMPARISON OF CLASSICAL SOFTWARE PROCESSES WITH AGILE PROCESSES

Table 1 describes the comparison of Classical Software Processes and Agile Processes in terms of delivery of product to customer, on basis of requirement specified by customer,

request for changes, involvement of customer in development of product and Risk factor as follows:

Table 1. Comparison of Classical Software Processes and Agile Processes

Criterion	Classical Software Processes	Agile Processes
Request for Changes at any time during development of product	Here change request is always rejected throughout development.	Changes are acceptable at any time during development.
Delivery of product in time/on time/early	Usually, deadlines are not meet and mostly impossible to deliver product before estimated deadline.	Delivery of product is as per estimated deadlines i.e. always delivered in or on time.
Quality of product is a major concern	In Waterfall model quality of product is not as desired by customer, because if user want some other changes then it is not possible in one go (during development time)	Quality is built-in; delivered product always satisfies the requirement or need of customer
Involvement of customer throughout development	After submitting the requirements in 1st phase, customer gets involved only on delivery of product.	Customer must be present at each and every phase of development.
Requirements	This model is used, if requirements of customer are clear and well defined	Agile model is used if requirements of customer are not clear or changes frequently.
Pattern	Waterfall model is a sequential model, means phases are always followed in consecutive manner.	As change occur frequently, so we can revisit any phase at any time.
Development time	Development life cycle is longer as compare to agile model.	If requirements are not so clear, are gathered on daily basis, then adopting agile makes sense.
Risk factor	There is a lot of risk of not meeting customer's requirement	Risk is less in Agile development because customer is involved in each and every phase of development.

6. SOFTWARE PROCESS IMPROVEMENT

A software process can be defined as "the sequence of steps required to develop or maintain software" [23], aiming at providing the "technical and management framework for applying methods, tools, and people to the software task" [23]. Software Process Improvement (SPI) aims at providing software development organizations with mechanisms for evaluating their existing processes, identifying possibilities for improving as well as implementing and evaluating the impact of improvements [24].

6.1 Software Process Improvement Models

There are various standard process models, such as CMM® [25], CMMI® [26], ISO 15504, i.e., SPICE (Software Process Improvement and Capability Determination), Trillium [27], and Bootstrap [29] that provide a reference process model against which organizational processes can be assessed and improved. Standard software development process models provide a top-down approach for SPI which offer a framework against which the organization can evaluate and improve its own processes and identify practices that would increase the maturity of the current processes [26].

6.1.1 ISO vs CMM vs Agile

Table 2 describes the comparison of ISO, CMM and Agile model:

Table 2. ISO vs CMM vs CMM

ISO	CMM	Agile
Emphasizes minimal quality criteria	Emphasizes process improvement and maturity	Emphasizes individual and interactions
Set of documented procedures that cover all aspects of business	Set of processes practices and behavior that deliver predicted outcomes	Set of methodologies which helps Rapid and Continuous delivery of useful software

6.1.2 Areas where Agile model suits best:

Agile software development methods are now being widely used in the IT sector and are increasingly being advocated as preferable to the traditional development model. Agile models are applicable at every area of software development. It is best

suited for Web-Based application in order to remove bugs in iterative manner.

6.1.2.1 Why to use Agile model for web based development:

Changes; Adding new features: This is major factor that effects Web based application most. Web Based applications must be flexible to welcome changes or add new features. So it's better to adopt Agile model for development of web based application in order to provide flexibility of handling future changes.

Reduces Risk: In web based development risk of not meeting user requirement is very high because of lack of user involvement. Agile model reduces such risk as it requires high involvement of user in order to deliver high quality product as per user requirement.

Scrum: In order to deliver high quality Web application scrum is must so that requirements can be gathered on daily-basis.

Testing/Removing Conflicts: Scrum also suits for testing web based application. As Scrum involves each and every member of development team as well as clients. This is the efficient way to solve conflicts if any.

Extreme Programming(XP): XP aims at combining the activities of analysis, design, implementation and testing, a little at a time, throughout the entire development process. It is the best way to make web application available in shorthand and to improve it in next iteration. Improvement is done by removing the bug in iteration process.

7. COMPONENT-BASED SOFTWARE ENGINEERING

Component-based software engineering emerged in the late 1990s as an approach to software systems development based on reusing software components.

Definition: Component-Based Software Engineering is a process that emphasizes the design and construction of computer-based systems using reusable software "components" [29].

Dictionary meaning of "Component": A unit of, part of a model.

7.1 Problems of Software Engineering

1. Size & Complexity increases rapidly.
2. Software is upgraded mostly after development
3. Time-to-market must decrease significantly.
4. The cost of product increases according to our predictions

7.2 Issues with Traditional Software Models

1. Lack of Reusability: Due to development based in specific requirements.

2. Lack of standardized component interface between components: Components interfaces are designed for a specific project. No consistent mechanism for supporting component interactions

3. Lack of Customization: Customization is not possible.

4. Lack of Component Interoperability: Due to lack of consistent data exchange mechanism between components; Due to lack of consistent interaction mechanism between components

7.3 Component-based Development

All the above issues can be resolved by Component-based development. As, component based development provides the idea: to build Software system form pre-existing components. Example – building furniture from existing components, for building components that can be reused in different applications. In CBSE maintenance is done by replacing of component and introducing new components into system.

7.3.1 CBSE vs Traditional Software Engineering

1. CBSE life cycle is shorter as compare to waterfall model.

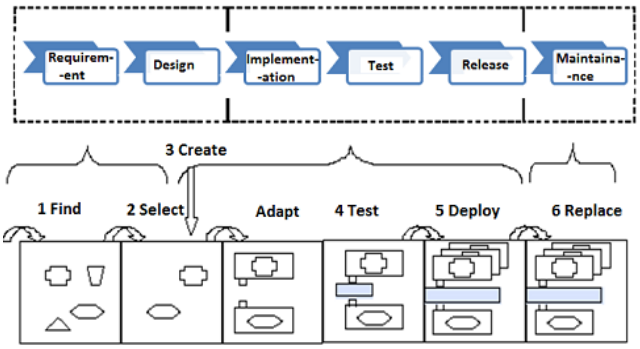

Fig 3: Waterfall vs CBSE Development Cycle [30]

2. CBSE develops architecture.

3. CBSE is less expensive because in this high quality & certified components are reused to form system, which also reduces risk of failure.

4. In CBSE maintenance is easily done by replacing components and introducing new component to system

5. Time-to-market is less in Component based development.

8. CONCLUSION

The timely adopted and adapted changes in software development approaches slowly changed the face of software development. Different flavors of software development originating from classical to agile and component based models showcased distinct ways of software development. As Adhoc software development approach gave birth to software crisis, middle-aged plan driven models introduced process structure. Now the era of change driven development which is moving towards implementing change in requirement, design and code at any point of time. One of the Agile development technique is SCRUM that supports implementing frequent requirement change. This characteristic of SCRUM made it first choice of software developers. But SCRUM approach compromised on software reusability. Whereas CBSE delivers reusable components. This paper compares and contrasts traditional software processes with modern software development approaches like Agile, XP and CBSE. Software Development with simultaneous process improvement still remains a challenge for many software organizations.

9. REFERENCES

[1] Boehm, B. 1988. A Spiral Model of Software Development and Enhancement. Computer, Vol. 21, 5 (5), May 1988, pp. 61-72.

[2] Basili, V. R. & Reiter, R. 1981. A Controlled Experiment Quantitatively Comparing Software Development Approaches. IEEE Transactions on Software Engineering, Vol. 7, 3 (3), pp. 299-320.

[3] Larman, C. & Basili, V. R. 2003. Iterative and Incremental Development: A Brief History. IEEE Software, Vol. 20, pp. 47-56.

[4] Benington, H. D. 1983. Production of Large Computer Programs. Annals of the History of Computing, Vol. 5, 4 (4), October, pp. 350-361.

[5] Royce, W. W. 1970. Managing the Development of Large Software Systems. In: The proceedings of the WESCON. San Francisco. IEEE CS. Pp. 328-339.

[6] Lycett, M., Macredie, R. D., Patel, C. & Paul, R. J. 2003. Migrating Agile Methods to Standardized Development Practice. Computer, Vol. 36, 6 (6), June, pp. 79-85.

[7] Larman, C. & Basili, V. R. 2003. Iterative and Incremental Development: A Brief History. IEEE Software, Vol. 20, pp. 47-56.

[8] Larman, C. 2004. Agile and Iterative Development: A Manager's Guide. Pearson Education, Inc. Boston. 342 p.

[9] Gilb, T. 1981. Evolutionary Development. ACM SIGSOFT Software Engineering Notes, Vol. 6, 2 (2). April, pp. 17.

[10] Gilb, T. 1988. Principles of Software Engineering Management. Addison-Wesley. Wokingham, UK, 464 p.

[11] Gilb, T. 2005. Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage. Butterworth-Heinemann. 480 p.

[12] Agile Alliance Manifesto for Agile Software Development.2001

[13] Beck, K. 1999. Embracing Change with Extreme Programming. IEEE Computer, Vol. 32, 10 (10), pp. 70-77.

[14] Beck, K. 2000. Extreme Programming Explained: Embrace Change. Addison Wesley Longman, Inc. 190 p.

[15] Schwaber, K. 1995. Scrum Development Process. In: The proceedings of the OOPSLA'95 Workshop on Business Object Design and Implementation. Springer-Verlag. Pp. 117-134.

[16] Williams, L. & Cockburn, A. 2003. Agile Software Development: It's about Feedback and Change. IEEE Computer Society, Vol. 36, 6 (6), June, pp. 39-43.

[17] Highsmith, J. A. 2000. Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. Dorset House Publishing. New York, NY. 358 p.

[18] Jeffries, R. E. 1999. eXtreme Testing: Why Aggressive Software Development Calls for Radical Testing Efforts. Software Testing & Quality Engineering, Vol. March/April, pp. 23-26.

[19] Cockburn, A. 1998. Surviving Object-Oriented Projects. Addison-Wesley. Reading, Mass. 250 p.

[20] Stapleton, J. 2003. DSDM: Business Focused Development. Second Edition. Addison Wesley. London. 239 p.

[21] Coad, P., LeFebvre, E. & De Luca, J. 1999. Java Modeling In Color With UML: Enterprise Components and Process. Prentice Hall. 221 p.

[22] Abrahamsson, P., Salo, O., Ronkainen, J. & Warsta, J. 2002. Agile Software Development Methods: Review and Analysis. VTT Publications 478. VTT Electronics. Espoo. 107 p. ISBN 951-38-6009-4; 951-38-6010-8.

[23] Humphrey, W. S. 1995. A Discipline for Software Engineering. Addison Wesley Longman, Inc. 242 p.

[24] Florac, W. A., Carleton, A. D. & Barnard, J. R. 2000. Statistical Process Control: Analyzing a Space Shuttle Onboard Software Process. IEEE Software, Vol. 17, 4 (4). July–August, pp. 97-106.

[25] Paulk, M., Curtis, B., Chrissis, M. & Weber, C. 1993. Capability Maturity Model for Software (Version 1.1). CMU/SEI-93-TR-024. Software Engineering Institute (SEI). February. 65 p.

[26] SEI, C. M. S. E. I. Capability Maturity Model® Integration (CMMISM), Version 1.1. Carnegie Mellon Software Engineering Institute. 2001.

[27] Bell Canada Trillium: Model for Telecom Product Development & Support Process Capability. Release 3.0. Bell Canada. December, 1994. 118 p.

[28] Kuvaja, P. & Bicego, A. 1993. Bootstrap: Europe's Assessment Method. IEEE Software, Vol. 10, 3 (3), May, pp. 93-95.

[29] Pressman, R. S., Software Engineering—A Practitioner's Approach, New York: McGrawHill International Ltd., 2010. 847 p

[30] Mili, Mili, Yacoub, Addy Edward, Reuse-Based Software Engineering, AWiley-Interscience Publication, John Wiley & Sons,INC.,2002. 540 p.