International Journal of Engineering & Technology, 7 (2.28) (2018) 325-331

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Software process models: a review and analysis

Ramzi A. Haraty [1] *, Gongzhu Hu [2]

[1] Department of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon
[2] Department of Computer Science, Central Michigan University, Mount Pleasant, MI 48859, USA
**Corresponding author E-mail: rharaty@lau.edu.lb*

Abstract

Modeling of software process has been a very challenging problem and constantly debated in the software development community in the past 30+ years, largely due to the complex nature of the software development process that involves not only the technical knowledge and skills but also many other factors, such as human, management, quality assessment, and cost. Although the situations of creating software differ greatly from one case to another, there are some common themes shared by many of the situations, and hence various software process models have been emerged to address these common themes. In this paper, we present a review of the software process models commonly used in practice, from traditional to agile, and assessment of these models with metrics and case studies.

Keywords: *Software Process; Software Development Life Cycle; Traditional Models; Agile Models; Process Metrics; Model Evaluation*

1. Introduction

The process of creating software is similar to the process of making any other products that involves many components and factors. Only when the components are well designed and fit together, the product can have a better chance to be successful. Software processes, however, differ quite significantly from product processes in other industries. Product processes in other industries (auto industry, for example) are often stable, have long-term product design and development strategies with anticipated customer expectations that do not change frequently in any significant way. Software processes, however, are more likely conducted under dynamically changing environments, from product requirements and functionality, time and cost constraints, to users' satisfaction. Methodologies to address the problems specific to software development started in 1960's but only quite primitive at the beginning and became somewhat mature in the last 30 years. The evolution of software development methodologies can be roughly divided into two periods with traditional or plan-driven models in the first period and agile models in the second. As agile models are getting more and more acceptance, traditional models continue to be modified, improved, and used today.

All the software process models are based on the concept of software development life cycle (SDLC), but traditional models interpret SDLC in a much narrow sense. They divides the "life" of software development process into well defined

Phases and specifies the activities in each phase and the process flows between the phases. The most influential traditional model is the waterfall model proposed by Winston Royce in 1970 [25]. Many variations of the waterfall model were proposed during that period, such as spiral model [6], V-shaped model [12] and W model [13].

There are clear limitations and weaknesses of traditional models particularly for complex and large software projects, as reported in many publications in the literature. Agile methodologies started gaining momentum in early 2000's, particularly when the Agile Alliance published The Agile Manifesto in 2001 [1]. The agile idea is focused on repeated light-weight practices for rapid and continuous delivery of software in small chunks with close collaboration from the customer as well as among members of the development teams. No rigid plan or requirements is determined in advance, as these can change during the development process. Being flexible and adaptive to changes are in the DNA of agile methods while still achieve the ultimate goal of producing customer satisfied software within the time and cost framework. Various agile-like models have been proposed and developed in the last 10–15 years, such as Extreme Programming [4], Scrum [27], [9], Lean Software Development [23], and Kanban [2].

In this paper, we present a brief review of commonly used traditional and agile models, model evaluation metrics, case studies and trend in software development

In any medium, provided the original work be properly cited.

2. Traditional software process models

The process of creating a software contains these essential building blocks regardless of what model is used:

Functional and non-functional requirements, time and cost constraints, design of the software product, implementation, testing, delivery, and maintenance. Software development models differ in the ways these building blocks are structured and how they are related in the process. Traditional models are basically follow the idea of process flow between these components (called stages or phases), like a finite state machine [35], to form what is called the software life cycle (SLC). The most basic and influential SLC development model is the waterfall model. There are many traditional models, but we only briefly discuss the three classical and representative models in this section: waterfall, V-shaped, and spiral.

2.1. Waterfall model

Winston Royce proposed waterfall model in 1970 [25]. In this model, the components in the software process are arranged as a

Copyright © 2018 Ramzi A. Haraty, Gongzhu Hu. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

linear sequence, with the flow from the first component "down" to the last one, like waterfalls, as depicted in Fig. 1. The solid-arrow flow is the basic model and the dashed-arrow flow indicates iterative feedback.

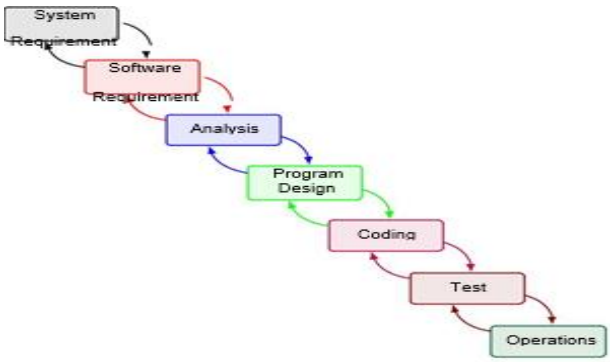


Fig. 1: The Waterfall Model [25].

The waterfall model has been perceived as an one-way flow of control in the software development process, where the process is denoted $S = (s_1; s_2; ; s_m)$, in which stage s_k starts only after stage s_{k-1} is complete. All the system and software requirements must be well specified at the beginning before analysis and design of the software product can begin, and cannot be changed during the entire process. In fact, the original waterfall model described in [25] had feedback loops as well as break-downs of these stages. Nevertheless, the one-way linear flow of the development process has been considered the main characteristics of the waterfall model. In this model, everything is well planned, the software product blueprint (design) strictly reflects the analysis of the requirements, and the blueprint is vigorously followed during implementation and testing to ensure the requirements are met.

The 7-stage waterfall in the original model has been massaged and adjusted into 5-stage, 6-stage, revised 7-stage and even more-stage waterfalls. Most noticeable revisions include combination of system requirements and and software requirements in to a single "Requirements" stage, and the addition of a "Maintenance and Support" stage at the end of the process. These revisions share the common strengths and weaknesses.

The primary strength of the waterfall model is the predictability of the product quality because "things are precise as planned."

The main drawback of this model is its inflexibility of adapting changes while changes are the game in software development. More often than not, even the customers are not sure about their needs at the beginning, hence the system and software requirements would change that have ripple effect on the later stages of the process. Because of this inflexibility, risk (project may fail) remains high throughout most of the development cycle until at the later stages (coding and integration) when problems are uncovered but it is "too late" while the functionality of the project is expected to rise, as illustrated in Fig. [2] given in [3]. This is also the main criticism of the waterfall model that led to the rise of agile models.

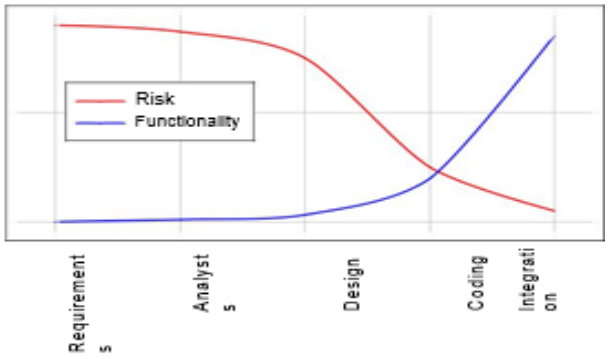


Fig. 2: Risk and Functionality Profile of Waterfall [3].

2.2. Spiral model

Barry Boehm introduced the spiral model in 1988 that was applied to the TRW Software Productivity System [6]. The model takes the waterfall stages into repetitive loops with the prototypes of the software design refined and improved in each loop, like the process going into a spiral, as shown in Fig. 3.

A distinctive feature of the spiral model is the added risk analysis into the loops. Here risk means the conditions and events that may make it harder for the development team to achieve its goals. In each iteration, the prototype of the project is evaluated and risk is assessed for the next iteration in the process. Sometimes the spiral model is categorized as a risk-driven model. The product prototype is iteratively updated (and hopefully improved) as the spiral spins outward and ends up with an operational prototype for implementation. A more detailed description of the

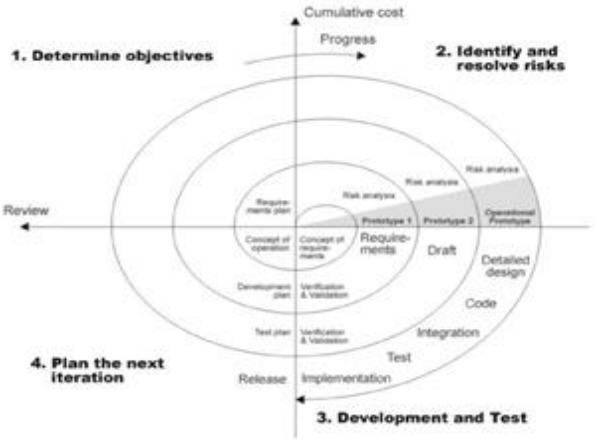


Fig. 3: Spiral Model [6].

Spiral model with enhancement was published in 2014 by the originator of the mode [7], that provided principles and guidelines of using spiral model for software projects to be successful.

The spiral model provides some flexibility to the process, such as allowing additional functionality to be added at the later stage of the development. However, the model is not widely used mainly because it is quite costly to use that requires skilled risk assessment professionals throughout the process and the model does not work well for small projects.

2.3. V-shaped model

The "Vee" model was introduced by Kevin Forsberg and Harold Mooz in 1991 [12] that followed the basic outlines of the "Vee" charts developed by NASA. The model emphasizes on integration and verification (testing). Just like the waterfall model, the software development life cycle is also a sequential process, but envisions the cycle as a V. The left side consists of the earlier phases of the cycle (requirement, specification, analysis, design) and descents just like in the waterfall model, while later phases (integration and verification) of the project cycle ascent on the right side of the V. In this model, detailed work can start early and even in parallel. For example, testing procedures can be developed early in the life cycle before coding is done. A V-shaped model is shown in Fig. 4 that is from [11] but with the dashed lines added to indicate the relationships of the phases in the planning.

Researchers have proposed and developed some extensions of the V-shaped models, such as the W model [13, 29]. In W model, a second V is added to have the two V's intertwining together to allow parallel execution of the phases of the project cycle. The second V emphasizes on testing that is integrated into the model. A W model for component-based development (CBD) was proposed by Kung-Kiu Lau et al. in 2011 [16]. In their model, one V is defined for the component development process and the other V is defined for system development process, and the two processes are conjoined into a single CBD process.

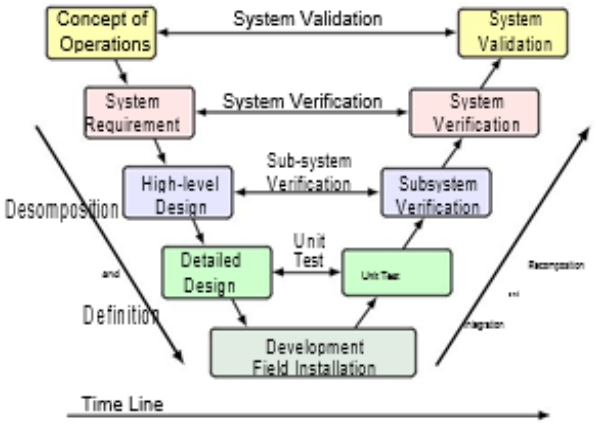


Fig. 4: The V-Shaped Model.

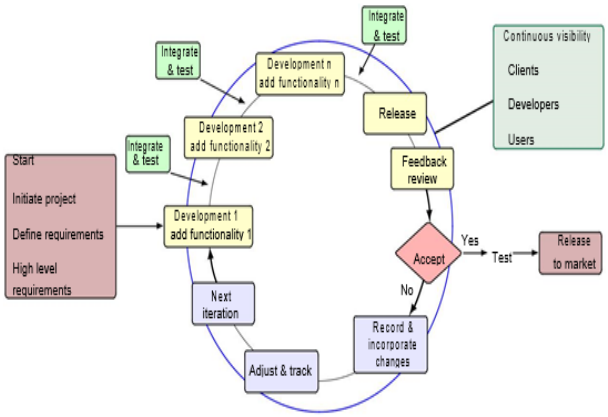


Fig. 5: Agile Life Cycle [20].

3. Agile software process models

The waterfall model and other traditional models were considered too rigid in practice. People started to realize that more flexible and practical models are needed. Quite a few software professionals put into practice in the 1990's the idea of Agile. In 2001, 17 agile practitioners gathered in Utah trying to find common goals. The Agile Software Development Alliance was formed at the meeting and all the participants signed the Manifesto for Agile Software Development [1] that has become the guiding principles for agile development models. At the core, "Agile Methodologies is about the mushy stuff of values and culture" to deliver good products to the customers in an environment that the acts of people are most important [14].

In this section, we review the basic characteristics of agile development and four agile models: Extreme Programming (XP), Scrum, Lean, and Kanban.

3.1. Agile characteristics and life cycle

There are long and short definitions for agile development, and all share these points [21]:

- Requirements are assumed to change.
- Systems evolve during a series of short iterations.
- Customers participate during each iteration.
- Documentation is developed only as needed.

The key characteristics of agile can be summarized into three phrases: adapting change, rapid delivery, constant user involvement.

The process in the agile life cycle is illustrated in Fig. 5 adapted from [20]. The product goes through a series of short iterations (Development i; i = one; n) with new functionality added in each iteration. During each iteration, design, coding, integration and testing are carried out to achieve incremental improvements. The client then releases the product for approval. If not accepted (client may request changes, for example), the development team will incorporate changes, adjust and track system features, and start the next cycle. The system is continuously visible to all stakeholders who are closely participating in the process for the entire development period. Agile is now the dominating method used today for software development.

3.2. Extreme programming (XP)

Extreme programming (XP) model was created by Kent Beck in 1996 when he was working on the Chrysler Corp.'s payroll application. The methodology was further defined and explained in a book [4]. Extreme programming is a "lightweight discipline of software development" methodology that emphasizes on customer satisfaction by delivering software as the customer needs it rather than delivering the software far in the future that satisfies all the requirements planned in advance. XP model recognizes five values: communication, simplicity, feedback, courage, and respect. A set of rules was established as the guide for software development using XP to reflect these values. Those rules [36] that are the most critical are summarized in Table 1.

Table 1: Rules of Extreme Programming [36]

Aspect	Rule
Planning	User stories are written Release plan creates release schedule make frequent small releases Project is divided into small iterations
Managing	Open work space standup meeting starts each day Measure project velocity
Designing	Fix XP when it breaks Keep it simple Create spike solutions to reduce risk Refactor whatever and whenever possible
Coding	Customer is always availabe Code standards Code unit test first All code is pair programmed Integrate often Code must pass all unit tests before release
Testing	Tests are created when a bug is found Acceptance tests are run often

There was a heated debate in the early 2000's, when XP (and agile in general) was just getting momentum, about the future Direction of software development. In addition to pointing out some weaknesses of XP, some criticism (such as [24]) argued that in extreme programming there is only coding but no requirements, no schedules, no documentation. A decade later, agile methodologies including XP has matured to the point that they are the dominating models in the software development landscape.

3.3. Scrum

Hirotaka Takeuchi and Ikujiro Monika first used the term Scrum in 1986 to explain the need for new approaches for product development [32]. They argued that a fast and flexible process is needed rather than sticking with the old sequential approach to developing new products, as they were calling for "stop running the relay race and take up rugby."

Scrum as a software development methodology was first introduced in 1995 by Ken Schwaber [27]. According to [27], Scrum is a loose set of activities going through an unpredictable system development process. Being flexible to respond to the unpredictability and incorporate controls and risk management can significantly increase the probability of system success.

Scrum is a framework within which various processes and techniques can be employed [28]. In the Scrum framework, three key roles (product owner, development team, Scrum master) closely work together to go through the process. The life cycle of the project consists of these steps:

1) Product backlog: documents about all the features to be developed with priority order.
2) Sprint backlog: list of things the team thinks can be done in the current sprint.
3) Sprint: project is divided into a series of 1-to-4-week periods (sprints), with design, coding, testing, and documentation in each sprint. No changes are allowed mid-sprint.
4) Scrum meeting: 15-minute daily meeting to review what was done yesterday, brainstorm what to do today and what are the blocking factors.

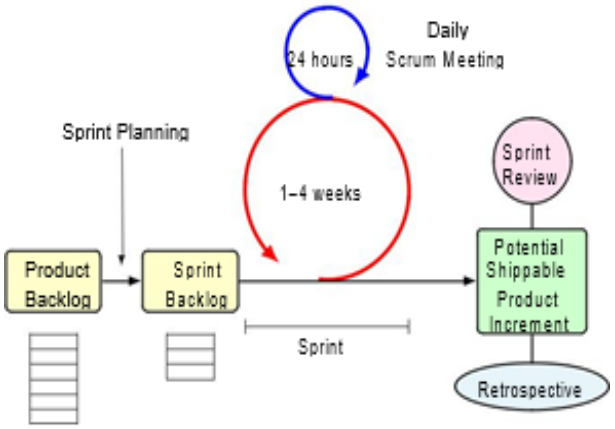


Fig. 6: Software Development Process in the Scrum Framework.

The artifacts, events, and steps in the Scrum development process are illustrated in Fig. 6 (adapted from [9]). The development team works closely with the Scrum master and the product owner throughout the process and creates an increment of potentially shippable product at the end of each sprint, ensuring fast delivery of high-quality product.

3.4. Lean development

The core concept of lean development is reduce waste and deliver quality products fast, namely, achieving leanness in the development process of a product. Lean development was originated from the manufacturing industry with Toyota Product Development System leading the effort successfully. When applying to the software development, the lean method focuses on seven principles [23]:

1) Eliminate waste: Develop and deliver exactly what the customer wants. Do not do anything more.
2) Amplify learning: Learning from the experience in the previous cycles is critical to the improvement of the product in the next cycle.
3) Decide as late as possible: Late decision making is more effective in the environment involving uncertainty. Better decision can be made if delayed.
4) Deliver fast: Rapid development enables reliable feedback and learning.
5) Empower the team: Develop team is involved in the details of technical decisions as they know better.
6) Build integrity in: Software needs to have a coherent architecture, fit its purpose, be maintainable, adaptable, and extensible.
7) See the whole: Software system consists of interdependent and interactive parts. "The ability of a system to achieve its purpose depends on how well the parts work together, not just how well they perform individually."

The concepts of Lean has been successfully applied to software development (e.g. case studies in [18]), but a more recent study concluded, "Advices to industry professionals to apply lean principles to large-scale software development is scarce" [22].

3.5. Kanban

Kanban is another agile method that emphasizes on continual delivery while lets the development team to balance their work load. Kanban is a Japanese word meaning "sign," "signal card," or "visual board." It was used to name a software development method for a distinctive feature of the method that emphasizes on visual display of the development process so that everyone in the team is fully aware of what is going on. An example of a kanban board is shown in Fig. [7] (adapted from [2]), in which each box on the board is a work item.

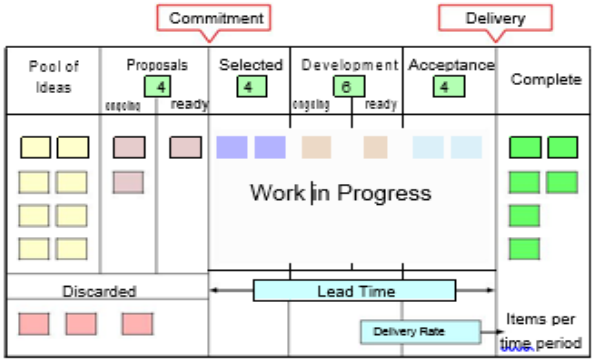


Fig. 7: Example of a Kanban Board [2].

The figure shows the two metrics: lead-time and delivery rate. Lead time L of an item is the duration between the commitment and delivery of the item. The average delivery rate R is the ratio of average work-in-progress over average lead-time according to Little's Law:

$$R = \frac{\overline{WiP}}{\overline{L}}$$

Delivery rate is the number of completed work items per time.

The essential activities in a Kanban system [2] are:

1) Visualize Use kanban board and other tools to visualize the work and policies.
2) Limit work in progress (WiP): WiP should be minimized. Too much on-going work is wasteful.
3) Manage flow: Managing flow of work should maximize the delivery of value of work items. The value of a work item is a function of cost of delay.
4) Make policies explicit: Policies like WiP limit, capacity allocation, definition of "done" may be clearly stated and stick at the top of each column on the kanban board.
5) Implement feedback loops: Cadences (cyclical meetings and reviews) should be properly scheduled for feedback.
6) Improve collaboratively, evolve experimentally: Seek continuous and incremental improvement without endpoint.

4. Model evaluation metrics

Evaluation of software process models is just as important as the models themselves. In this section, we give a brief reviewof some evaluation metrics.

4.1. Metrics in agile development

Agile methods have been widely adopted in the software industry, and evaluation of the usage of these models is an important factor for the organizations to make decisions for software development in the future. Kupiainen et al. published a systematic study [15] about the use of metrics in agile software development in industries. In 30 primary students from a large list of over 774 publications, a total of 102 metrics were found. These metrics are closely related to the motivations of using agile in the organizations and expected benefits. Here we just list a few of these metrics: business value delivered, customer satisfaction, defect count, velocity, effort estimate, percentage of stories prepared for sprint, lead time, cost performance index, work in progress, cycle time, implemented vs wasted requirements, etc. The top reasons for using metrics, the most popular and important metrics in their study are listed in Table 2.

Table 2: Top Reasons and Most Important Metrics [15]

	Metric
Reason for using metrics	sprint planning
	progress tracking
	software quality measurement
	fixing software process problems
	motivating people
Quantitative metrics	velocity
	effort estimate
	defect count
	customer satisfaction
Qualitative metrics	technical debt
	build status
	progress as working code

Table 3: Metrics for process models [8]

Metric	Measure
NOA	Number of activities in a process
NOAC	Number of activities and control-flow elements
NOAJS	Number of activities, joins, and splits
CFC	Control-flow complexity
HPC	Halstead process complexity
IC	Interface complexity
CNC	Coefficient of network complexity
RT	Restrictiveness estimator

Where C_{and} (a) = f about (a); Cor (a) = 2 $^{f\ anout\ (a)}$ (1)

And

C_{xor} (a) = 1

Halstead process complexity HPC is a measure based on the number of activities, splits, joins, and control-flow elements.
Interface complexity IC of an activity a is determined by the number of inputs and outputs of a as

IC = length (N_{input} $N_{out\ put}$) ^{2}

Coefficient of network complexity CNC is the ratio of number of edges vs the number of nodes in the graph:

$CNC = N_{edge} = N_{activity;\ join;\ split}$

Restrictiveness estimator RT measures the number of feasible sequences in the graph:

$$RT = 2 \underset{i\ j}{\text{å}} r_{i\ j}\ 6(n\ 1) = (n\ 2)\ (n\ 3)$$

Where $r_{i\ j}$ is the reachability matrix (i.e. transitive closure of adjacent matrix) and n is the number of nodes.

4.2. Model Complexity Metrics

Complexity of software process models has direct effects on the success or failure of applying the models on software products. In general, the more complex of a model, the harder for the model to be understood and maintained. There are many metrics for measuring model complexity suggested in the literature, from model size, structure, to comprehensiveness. Cardoso et al. provided a survey [8] in 2006, in which complexity metrics of process models are reviewed as an analogy to the metrics of programs. The NOA, NOAC, and NOAJS metrics are based on the line-of-code measure for programs, and the other metrics are based on the graph structure of the process model. A summary of the metrics is given in Table 3.
Control-flow complexity CFC is similar to the Cyclomatic measure for program complexity, defined based on the flow graph of a process P. CFC(p) is the sum of complexity measures of activity a 2 p, where a may be an and-split, or-split, or an xor-split in the graph:

$$CFC\ (p) = \underset{a2p}{\text{å}} C_{and}\ (a) + \underset{a2p}{\text{å}} C_{or}\ (a) + \underset{a2p}{\text{å}} C_{xor}\ (a)$$

5. Case studies

Some case studies have been researched on applying various software process models to real-world software projects, and and results were reported showing the pros and cons of these models applied in different project environments. The Scrum Standish Group CHAOS Report [30] showed that the success rates of traditional software projects (i.e. using the Waterfall model) and projects using an agile approach are 11% and 39%, respectively. Table 4 shows the comparison of the projects traced by the CHAOS project database.

Table 4: CHAOS Resolution by Agile vs Waterfall [30]

Model	Successful	Challenged	Failed
Waterfall	11%	60%	29%
Agile	39%	52%	9%

5.1. FBI's sentinel project

The Sentinel project was initiated in 2006 to be an internal system used by over 30,000 FBI employees, with original estimated budge of $451 million, to be deployed by end of 2009 [26]. It started using the traditional software
development model, but only delivered two of the four phases of the system in summer 2010 after spending $405 million. FBI shifted the process to agile model in 2010, and project was completed in summer 2012 with reduced development team and small budget. The comparison of using different models in the project is given in Table 5.

Table 5: Models Used for the FBI Sentinel Project [26]

Measure	Model	
	Waterfall	Agile
duration	4 years	12 months
staff size	400	45
budget (million)	$405	$30
result	half done	completed

5.2. Shift from waterfall to scrum at intel

Software development at Intel went from the traditional approaches, mostly waterfall, to agile or scrum approach. Intel has benefited from the shifting, such as reduced cycle time by 66%, uncovered software bugs, weak tools, and poor engineering habits [10]. The effort of adopting scrum method has changing the Product Development Engineering unit at Intel "from a command-and-control, plan-based organization into an inspecting and adapting, self-organizing, empirical planning-based organization."

5.3. Agile at yahoo

Yahoo started its Scrum pilot program in 2005 with four teams and has grown rapidly to 40 teams in just one year, and to over 150 teams using agile approaches [5]. In comparison of using Scrum vs the methods previous used, the responses from the team members overwhelmingly in favor of the Scrum method, shown in Table 6.

Table 6: Percentage of Responses [5]

How much was done in 30 days?	2%	24%	74%
Clarity of goals	6%	14%	80%
Business value produced in 30 days.	2%	34%	64%
Overall quality and "rightness" produced.	5%	41%	59%
Collaboration and cooperation within team?	0%	11%	89%
Amount of time wasted / work thrown out?	19%	13%	68%
Overall feelings about using Scrum.	9%	14%	77%

In addition, 81% of team members said they would continue to use Scrum.

5.4. Agile Practice at Google

Google has many very successful customer-oriented products such as search and gmail developed in the "startup culture" environment where most decisions were made by the engineering teams themselves without much interference from the management. However, a lot problems arose when the project AdWords (Google's online advertising service software) grown to become very big with very high rate. Of changes in the project. AdWords is a B2B application, quite different from most of other customer-oriented products, requiring much more business involvement. To address these problems, agile practices was carefully introduced into several project teams at Google. As stated in [31], the agile approach had resistance initially simply because of the googley way of developing software that many engineers did not believe any formal process can be helpful. After "trying it out" of the agile approach, the project teams had adopted it and put it into daily practice [31].

5.5. Lean management at BBC worldwide

A case study of applying lean ideas to managing software development of the BBC Worldwide Webmedia Department's software processes by a 9-person development team was reported in [19]. The data collected from a 12-month period, three months after implementation of lean started, contains 84 features (52% were small) in the first 5 months and 64 features (75% were small) in the last 5 months. The study shows that using lean method can actually improve software development, as shown in Table 7.

Table 7: Improvement by Using Lean at BBC Worldwide [19]

Measure	Change
Lead time to deliver software	+37%
Consistency of delivery	+47%
Defects reported by customers	24%

6. Trend

In this section, we review several recent surveys that show the growing trend of agility in the software industry, as well as in many other industries.

6.1. Agility in the software industry – HP survey

A 2015 survey [33] of 601 software developers and IT professionals conducted by HP shows that agile is on the rise and accelerated more rapidly since 2010. The percentages of the companies using waterfall-agile development methods in the survey is given in Table 8(a) and the main reasons for adopting for those whose companies use agile are given in Table 8(b).

Among those 475 responders with some adoption of agile methods in 2017, only 4% using agile in 2004, and the adoption sharpy accelerated during the 2009–2010 period, as shown in Fig. 8.

6.2. Agility across industries – version one survey

Version One conducts annual world-wide survey of the state of agile for since 2006 across a wide spectrum of industries, education, government, and non-profit. Software is the largest industry in the survey. The results of the surveys [34] were released in the year after the survey was conducted. The sizes and demographics of the annual surveys are given in Table 9.
The survey result reveals that enterprise agility continues8

Table 8: HP Survey Result: Waterfall vs Agile [33] (A) Primary Development Method Used in Organization

Method	%
Pure Waterfall	2
Leaning towards Waterfall	7
Hybrid	24
Leaning towards Agile	51
Agile	16

(B) Key Motivators

Key motivators	%
Enhance collaboration between teams	54
Increase the level of software quality	52
Results in increased customer satisfaction	49
Shortens time to market	43
Reduces cost of development	42

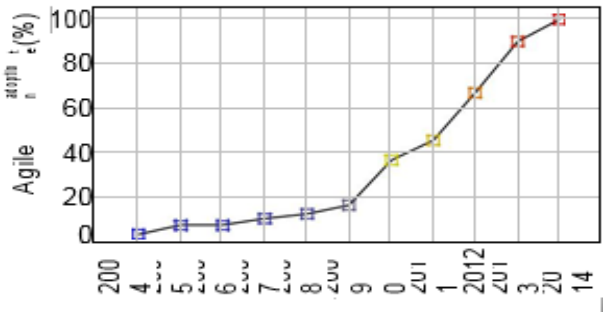


Fig. 8: Agile Adoption over Time [33].

to increase (e.g. 94% of responders' organizations practice agile), and there is still a lot of opportunity for growth (e.g. 60% of responders said less than half of teams use agile, and 80% said their organizations are at or below a "still maturing" level). An overwhelming majority of 98% responders stated that their organization has realized success from agile projects.
We collected the data from the VersionOne annual state of agile reports from 2007 (2nd) to 2016 (11th) to find the top responses regarding the following three areas of using agile methodologies, and if these top responses have changed over the years.

1) Agile methods used
2) Agile techniques employed
3) Benefits of adopting agile

Since some choices given in a few questions in the annual surveys varied from year to year, we could only select those that are common in most years. The trends of the above three survey items are shown in Fig. 9.
The agile methods used by survey responders' organizations are plotted in Fig. 9(a) measured in the percentage of the responses. Scrum is by far the most commonly used agile method throughout the years (50-58%), while the use of ScrumBan and Kanban has steadily increased although still a small minority. XP decreased significantly from 12% in 2007.

6.3. Traditional models are still useful

Although the survey and some other reports show that the traditional software development life cycle models have gradually faded out of favor as more and more companies and institutions are adopting agile approaches, the traditional methods are still well and live, and considered a reliable approach to software development. A 2013 case study of a successful home health component of a hospital healthcare system found that the traditional approach "is still as useful today as it ever was" [17].

6.4. Co-exist of traditional and agile methods

The software development community accepts agile methodologies rapidly in the last 15 years and many agile-like models have been introduced and practiced. This does not mean that the traditional models will be totally taken over anytime soon. Just like old programming languages COBOL and FORTRAN still comprise a large chunk of shares in today's software systems, traditional software process models, like the waterfall model, are well-defined and developed, and have shown strength in relatively stable software systems such that the one mentioned in the previous section 6.3. As stated in the SEI technical report [21], traditional and agile has some very basic similarities while possess significant differences. We have reviewed their differences in this paper; here we only summarize their similarities as identified in [21]:

1) Share the same goal: deliver a quality product in a