An Overview of Software Engineering

Version 1.0

February 28, 2015

Rohan Gadad

MS Candidate, Information Technology

Rensselaer Polytechnic Institute

Prepared for

ITWS\_6940, Software Engineering

Instructor: Ingrid Liu, Ph.D.

Spring 2015

Contents

[Abstract 4](#_Toc413802816)

[Software Engineering 4](#_Toc413802817)

[Sub disciplines of software engineering 4](#_Toc413802818)

[Requirements Engineering 4](#_Toc413802819)

[Software design 5](#_Toc413802820)

[Software Maintenance 6](#_Toc413802821)

[Software Testing 7](#_Toc413802822)

[Software Validation 7](#_Toc413802823)

[Software Verification 7](#_Toc413802824)

[Target of the test are - 7](#_Toc413802825)

[Manual Vs Automated Testing 7](#_Toc413802826)

[Testing Approaches 8](#_Toc413802827)

[Software Design Strategies 8](#_Toc413802828)

[Structured Design 8](#_Toc413802829)

[Function Oriented Design 8](#_Toc413802830)

[Object Oriented Design 8](#_Toc413802831)

[Computer Aided Software Engineering 9](#_Toc413802832)

[Software Development Life Cycle (SDLC) Models 11](#_Toc413802833)

[Waterfall Model 11](#_Toc413802834)

[Iterative Model 11](#_Toc413802835)

[Spiral Model 12](#_Toc413802836)

[V – Model 13](#_Toc413802837)

[Big Bang Model 14](#_Toc413802838)

[Agile Model 15](#_Toc413802839)

[Extreme 16](#_Toc413802840)

[Scrum 16](#_Toc413802841)

[Kanban 17](#_Toc413802842)

[REFERENCES 19](#_Toc413802843)

List of Figures

[Figure 1: Top Level Project Requirements 6](#_Toc413802806)

[Figure 2: Costs associated with a software project 7](#_Toc413802807)

[Figure 3: Waterfall Model 12](#_Toc413802808)

[Figure 4: Iterative Model 13](#_Toc413802809)

[Figure 5: Spiral Model 14](#_Toc413802810)

[Figure 6: V-Model 15](#_Toc413802811)

[Figure 7: Big Bang Model 16](#_Toc413802812)

[Figure 8: Agile Model 16](#_Toc413802813)

[Figure 9: Scrum 18](#_Toc413802814)

[Figure 10: Kanban 19](#_Toc413802815)

# Abstract

*Software Engineering is the study of the design, development, testing and maintenance of software systems. In this paper, I will review the field of software engineering, software project management, software lifecycle, tools used in software engineering etc.*

# Software Engineering

Software engineering (SE) is concerned with developing and maintaining software systems that behave reliably and efficiently, are affordable to develop and maintain, and satisfy all the requirements that customers have defined for them. It is important because of the impact of large, expensive software systems and the role of software in safety-critical applications. It integrates significant mathematics, computer science and practices whose origins are in engineering. [1]

Software engineering can be subdivided into the following sub disciplines.

1. Requirements Engineering
2. Software design
3. Software construction
4. Software testing
5. Software maintenance
6. Software configuration management
7. Software engineering process
8. Software quality management

The next section of the paper will give an introduction to a few of the disciplines mentioned above.

# Sub disciplines of software engineering

## Requirements Engineering

A typical project has requirements at several levels. The top level is a set of stakeholder requirements that expresses product requirements using a problem domain perspective. These should be completely independent of any proposed solution.

System requirements are at the second level, where an abstract solution is postulated but the solution still allows many different possible implementations. In other words, these system requirements should be as abstract as possible and should not preempt the final design solution.

At the third level are subsystem requirements. The number of levels below this level depends on the scale and complexity of the systems developed. [2]

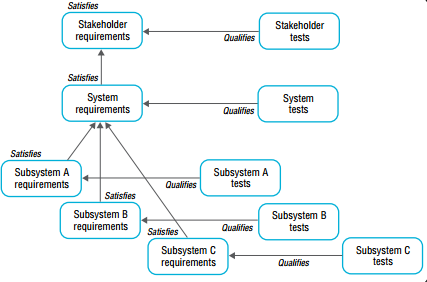


Figure 1: Top Level Project Requirements

Requirements engineering usually consists of the following activities

1. Requirements identification
2. Requirements analysis
3. SRS (Software Requirements Specification)
4. Requirements Validation
5. Requirements Management

## Software design

Software design is the process by which an agent creates a specification of a software artifact, intended to accomplish goals, using a set of primitive components and subject to constraints. Software design may refer to either "all the activities involved in conceptualizing, framing, implementing, commissioning, and ultimately modifying complex systems" or "the activity following requirements specification and before programming, as ... [in] a stylized software engineering process." [3]

Software design yields three levels of results:

* **Architectural Design -**The architectural design is the highest abstract version of the system. It identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of proposed solution domain.
* **High-level Design-**The high-level design breaks the ‘single entity-multiple component’ concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other. High-level design focuses on how the system along with all of its components can be implemented in forms of modules. It recognizes modular structure of each sub-system and their relation and interaction among each other.
* **Detailed Design-**Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs. It is more detailed towards modules and their implementations. It defines logical structure of each module and their interfaces to communicate with other modules.

## Software Maintenance

In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:

* **Corrective Maintenance** - This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.
* **Adaptive Maintenance** - This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.
* **Perfective Maintenance** - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.
* **Preventive Maintenance** - This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

**Cost of Maintenance**

Reports suggest that the cost of maintenance is high. A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.

On an average, the cost of software maintenance is more than 50% of all SDLC phases.

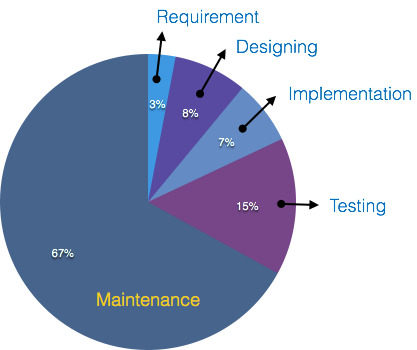


Figure 2: Costs associated with a software project

## Software Testing

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation and Verification.

### Software Validation

Validation is process of examining whether or not the software satisfies the user requirements. It is carried out at the end of the SDLC. If the software matches requirements for which it was made, it is validated.

* Validation ensures the product under development is as per the user requirements.
* Validation answers the question – "Are we developing the product which attempts all that user needs from this software ?".
* Validation emphasizes on user requirements.

### Software Verification

Verification is the process of confirming if the software is meeting the business requirements, and is developed adhering to the proper specifications and methodologies.

* Verification ensures the product being developed is according to design specifications.
* Verification answers the question– "Are we developing this product by firmly following all design specifications ?"
* Verifications concentrates on the design and system specifications.

### Target of the test are -

* **Errors** - These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, is considered as an error.
* **Fault** - When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail.
* **Failure**- failure is said to be the inability of the system to perform the desired task. Failure occurs when fault exists in the system.

### Manual Vs Automated Testing

Testing can either be done manually or using an automated testing tool:

* **Manual** - This testing is performed without taking help of automated testing tools. The software tester prepares test cases for different sections and levels of the code, executes the tests and reports the result to the manager.

Manual testing is time and resource consuming. The tester needs to confirm whether or not right test cases are used. Major portion of testing involves manual testing.

* **Automated** This testing is a testing procedure done with aid of automated testing tools. The limitations with manual testing can be overcome using automated test tools.

A test needs to check if a webpage can be opened in Internet Explorer. This can be easily done with manual testing. But to check if the web-server can take the load of 1 million users, it is quite impossible to test manually.

There are software and hardware tools which helps tester in conducting load testing, stress testing, regression testing.

### Testing Approaches

Tests can be conducted based on two approaches –

* Functionality testing
* Implementation testing

When functionality is being tested without taking the actual implementation in concern it is known as black-box testing. The other side is known as white-box testing where not only functionality is tested but the way it is implemented is also analyzed.

Exhaustive tests are the best-desired method for a perfect testing. Every single possible value in the range of the input and output values is tested. It is not possible to test each and every value in real world scenario if the range of values is large.

# Software Design Strategies

## Structured Design

Structured design is a conceptualization of problem into several well-organized elements of solution. It is basically concerned with the solution design. Benefit of structured design is, it gives better understanding of how the problem is being solved. Structured design also makes it simpler for designer to concentrate on the problem more accurately.

Structured design is mostly based on ‘divide and conquer’ strategy where a problem is broken into several small problems and each small problem is individually solved until the whole problem is solved.

## Function Oriented Design

In function-oriented design, the system is comprised of many smaller sub-systems known as functions. These functions are capable of performing significant task in the system. The system is considered as top view of all functions.

Function oriented design inherits some properties of structured design where divide and conquer methodology is used.

## Object Oriented Design

Object oriented design works around the entities and their characteristics instead of functions involved in the software system. This design strategies focuses on entities and its characteristics. The whole concept of software solution revolves around the engaged entities.

The important concepts of Object Oriented Design:

* **Objects -**All entities involved in the solution design are known as objects. For example, person, banks, company and customers are treated as objects. Every entity has some attributes associated to it and has some methods to perform on the attributes.
* **Classes -**A class is a generalized description of an object. An object is an instance of a class. Class defines all the attributes, which an object can have and methods, which defines the functionality of the object. In the solution design, attributes are stored as variables and functionalities are defined by means of methods or procedures.
* **Encapsulation -**In OOD, the attributes (data variables) and methods (operation on the data) are bundled together is called encapsulation. Encapsulation not only bundles important information of an object together, but also restricts access of the data and methods from the outside world. This is called information hiding.
* **Inheritance -**OOD allows similar classes to stack up in hierarchical manner where the lower or sub-classes can import, implement and re-use allowed variables and methods from their immediate super classes. This property of OOD is known as inheritance. This makes it easier to define specific class and to create generalized classes from specific ones.
* **Polymorphism -**OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned same name. This is called polymorphism, which allows a single interface performing tasks for different types. Depending upon how the function is invoked, respective portion of the code gets executed.

# Computer Aided Software Engineering

**Diagram tools**

These tools are used to represent system components, data and control flow among various software components and system structure in a graphical form. For example, Flow Chart Maker tool for creating state-of-the-art flowcharts.

**Process Modeling Tools**

Process modeling is method to create software process model, which is used to develop the software. Process modeling tools help the managers to choose a process model or modify it as per the requirement of software product. For example, EPF Composer

**Project Management Tools**

These tools are used for project planning, cost and effort estimation, project scheduling and resource planning. Managers have to strictly comply project execution with every mentioned step in software project management. Project management tools help in storing and sharing project information in real-time throughout the organization. For example, Creative Pro Office, Trac Project, Basecamp.

**Documentation Tools**

Documentation in a software project starts prior to the software process, goes throughout all phases of SDLC and after the completion of the project.

Documentation tools generate documents for technical users and end users. Technical users are mostly in-house professionals of the development team who refer to system manual, reference manual, training manual, installation manuals etc. The end user documents describe the functioning and how-to of the system such as user manual. For example, Doxygen, DrExplain, Adobe RoboHelp for documentation.

**Analysis Tools**

These tools help to gather requirements, automatically check for any inconsistency, inaccuracy in the diagrams, data redundancies or erroneous omissions. For example, Accept 360, Accompa, CaseComplete for requirement analysis, Visible Analyst for total analysis.

**Design Tools**

These tools help software designers to design the block structure of the software, which may further be broken down in smaller modules using refinement techniques. These tools provides detailing of each module and interconnections among modules. For example, Animated Software Design

**Configuration Management Tools**

An instance of software is released under one version. Configuration Management tools deal with –

* Version and revision management
* Baseline configuration management
* Change control management

CASE tools help in this by automatic tracking, version management and release management. For example, Fossil, Git, Accu REV.

**Programming Tools**

These tools consist of programming environments like IDE (Integrated Development Environment), in-built modules library and simulation tools. These tools provide comprehensive aid in building software product and include features for simulation and testing. For example, Cscope to search code in C, Eclipse.

**Prototyping Tools**

Software prototype is simulated version of the intended software product. Prototype provides initial look and feel of the product and simulates few aspect of actual product.

Prototyping CASE tools essentially come with graphical libraries. They can create hardware independent user interfaces and design. These tools help us to build rapid prototypes based on existing information. In addition, they provide simulation of software prototype. For example, Serena prototype composer, Mockup Builder.

**Web Development Tools**

These tools assist in designing web pages with all allied elements like forms, text, script, graphic and so on. Web tools also provide live preview of what is being developed and how will it look after completion. For example, Fontello, Adobe Edge Inspect, Foundation 3, Brackets.

**Quality Assurance Tools**

Quality assurance in a software organization is monitoring the engineering process and methods adopted to develop the software product in order to ensure conformance of quality as per organization standards. QA tools consist of configuration and change control tools and software testing tools. For example, SoapTest, AppsWatch, JMeter.

**Maintenance Tools**

Software maintenance includes modifications in the software product after it is delivered. Automatic logging and error reporting techniques, automatic error ticket generation and root cause Analysis are few CASE tools, which help software organization in maintenance phase of SDLC. For example, Bugzilla for defect tracking, HP Quality Center.

# Software Development Life Cycle (SDLC) Models

The software development paradigm helps developer to select a strategy to develop the software. A software development paradigm has its own set of tools, methods and procedures, which are expressed clearly and defines software development life cycle. A few of software development paradigms or process models are defined as follows:

## Waterfall Model

Waterfall model is the simplest model of software development paradigm. It says the all the phases of SDLC will function one after another in linear manner. That is, when the first phase is finished then only the second phase will start and so on.

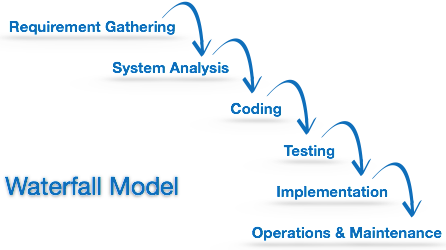


Figure 3: Waterfall Model

This model assumes that everything is carried out and taken place perfectly as planned in the previous stage and there is no need to think about the past issues that may arise in the next phase. This model does not work smoothly if there are some issues left at the previous step. The sequential nature of model does not allow us go back and undo or redo our actions.

This model is best suited when developers already have designed and developed similar software in the past and are aware of all its domains.

## Iterative Model

This model leads the software development process in iterations. It projects the process of development in cyclic manner repeating every step after every cycle of SDLC process.

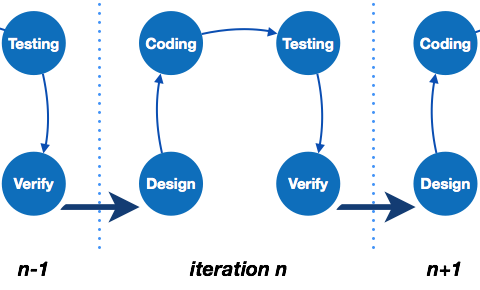


Figure 4: Iterative Model

The software is first developed on very small scale and all the steps are followed which are taken into consideration. Then, on every next iteration, more features and modules are designed, coded, tested and added to the software. Every cycle produces a software, which is complete in itself and has more features and capabilities than that of the previous one.

After each iteration, the management team can do work on risk management and prepare for the next iteration. Because a cycle includes small portion of whole software process, it is easier to manage the development process but it consumes more resources.

## Spiral Model

Spiral model is a combination of both, iterative model and one of the SDLC model. It can be seen as if you choose one SDLC model and combine it with cyclic process (iterative model).

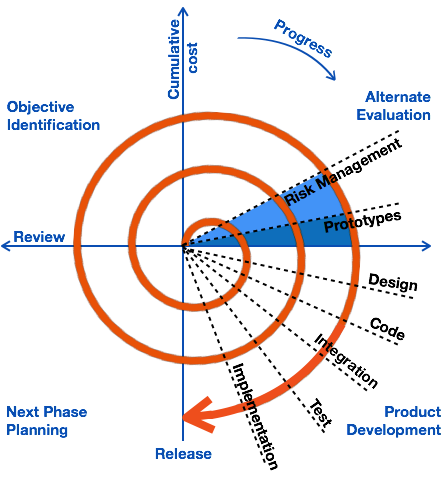


Figure 5: Spiral Model

This model considers risk, which often goes un-noticed by most other models. The model starts with determining objectives and constraints of the software at the start of one iteration. Next phase is of prototyping the software. This includes risk analysis. Then one standard SDLC model is used to build the software. In the fourth phase of the plan of next iteration is prepared.

## V – Model

The major drawback of waterfall model is we move to the next stage only when the previous one is finished and there was no chance to go back if something is found wrong in later stages. V-Model provides means of testing of software at each stage in reverse manner.

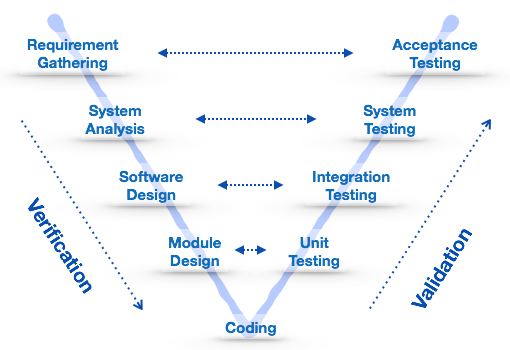


Figure 6: V-Model

At every stage, test plans and test cases are created to verify and validate the product according to the requirement of that stage. For example, in requirement gathering stage the test team prepares all the test cases in correspondence to the requirements. Later, when the product is developed and is ready for testing, test cases of this stage verify the software against its validity towards requirements at this stage.

This makes both verification and validation go in parallel. This model is also known as verification and validation model.

## Big Bang Model

This model is the simplest model in its form. It requires little planning, lots of programming and lots of funds. This model is conceptualized around the big bang of universe. As scientists say that after big bang lots of galaxies, planets and stars evolved just as an event. Likewise, if we put together lots of programming and funds, you may achieve the best software product.

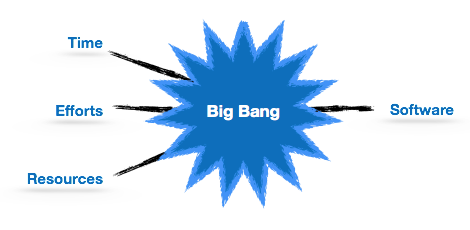


Figure 7: Big Bang Model

For this model, very small amount of planning is required. It does not follow any process, or at times the customer is not sure about the requirements and future needs. So the input requirements are arbitrary.

This model is not suitable for large software projects but good one for learning and experimenting.

## Agile Model

Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In agile the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

Here is a graphical illustration of the Agile Model:

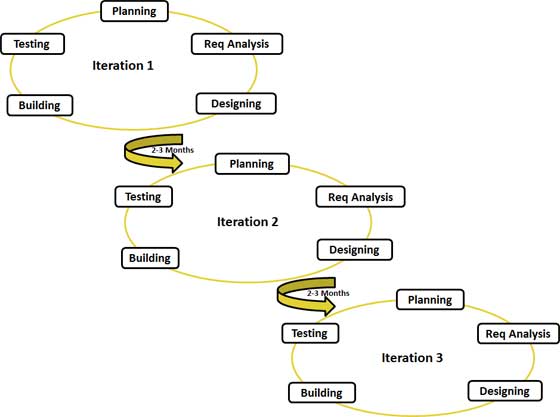


Figure 8: Agile Model

Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability.

### Extreme

In extreme project management methodology, there are no fixed project phases and fixed set of guidelines on how to execute the project activities. Extreme methodology adapts to the situation and executes the project activity the best way possible.

By nature, extreme project management methodology does not have lengthy deadlines or delivery dates. The delivery cycles are shorter and usually they are 2 weeks. Therefore, the entire project team is focused on delivering the scope of the delivery in short term. This allows the team to welcome any scope or requirement changes for the next delivery cycle.

In extreme approach, things are done totally a different way, when compared to tradition approaches. Therefore, changing the mindset of the project team is one of the main requirements and responsibilities of the management team. When it comes to changing the mindset, consider the following rules as the ground rules for extreme approach for project management.

* Requirements and project activities being chaotic is normal
* Uncertainty is the most certain characteristic of an extreme project
* This type of projects are not fully controllable
* Change is the king and you need to welcome it every possible way
* The feeling of security is increased by relaxing the project controls

### Scrum

Scrum is an agile way to manage a project, usually software development. Agile software development with Scrum is often perceived as a methodology; but rather than viewing Scrum as methodology, think of it as a framework for managing a process.

In the agile Scrum world, instead of providing complete, detailed descriptions of how everything is to be done on a project, much of it is left up to the Scrum software development team. This is because the team will know best how to solve the problem they are presented.

This is why in Scrum development, for example, a sprint planning meeting is described in terms of the desired outcome (a commitment to a set of features to be developed in the next sprint) instead of a set of Entry criteria, Task definitions, Validation criteria, Exit criteria (ETVX) and so on, as would be provided in most methodologies.

Scrum relies on a self-organizing, cross-functional team. The scrum team is self-organizing in that there is no overall team leader who decides which person will do which task or how a problem will be solved. Those are issues that are decided by the team as a whole.

And in Scrum, a team is cross functional, meaning everyone is needed to take a feature from idea to implementation.

Within agile development, Scrum teams are supported by two specific roles. The first is a ScrumMaster, who can be thought of as a coach for the team, helping team members use the Scrum process to perform at the highest level.

The product owner (PO) is the other role, and in Scrum software development, represents the business, customers or users, and guides the team toward building the right product.

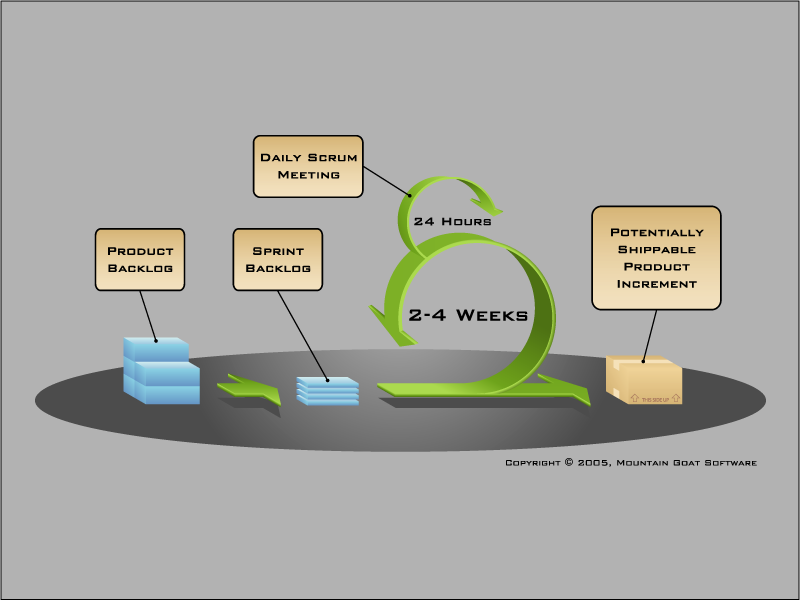


Figure 9: Scrum

### Kanban

Kanban is another framework used to implement agile. Back in the 1940s, Toyota optimized its engineering process by modeling it after how supermarkets stock shelves. Supermarkets stock just enough product to meet consumer demand, a practice that optimizes the flow between the supermarket and the consumer. Because inventory levels match with consumption patterns, the supermarket gains significant efficiency in inventory management and optimizing for the customer. When Toyota brought that idea to its factory floors, teams (such as the team that attaches the doors to the car's frame) would deliver a card, or "kanban", to each other (say, to the team that assembles the doors) to signal that they have excess capacity and are ready to pull more materials. Although the signaling technology has evolved, this system is still at the core of "just in time" manufacturing today.

Kanban does the same for software teams. By matching the amount of work in progress to the team's capacity, kanban gives teams more flexible planning options, faster output, clear focus, and transparency throughout the development cycle.



Figure 10: Kanban

|  | SCRUM | KANBAN |
| --- | --- | --- |
| Cadence | Regular fixed length sprints | Continuous flow |
| Release methodology | At the end of each sprint if approved by the product owner | Continuous delivery or at the team's discretion |
| Roles | Product owner, scrum master, development team | No existing roles. Some teams enlist the help of an agile coach. |
| Key metrics | Velocity | Cycle time |
| Change philosophy | Teams should strive to not make changes to the sprint forecast during the sprint. Doing so compromises learnings around estimation. |  |

# REFERENCES

1. Software Engineering - http://computingcareers.acm.org/?page\_id=12
2. Requirements Engineering - <http://www.biglever.com/extras/RE_for_SPL.pdf>
3. Software Design - <http://en.wikipedia.org/wiki/Software_design>
4. Software Design - <http://www.tutorialspoint.com/software_engineering/software_implementation.htm>
5. Scrum - <http://www.mountaingoatsoftware.com/agile/scrum/overview>
6. Kanban - <https://www.atlassian.com/agile/kanban>