

SWAG:? A story point is an **arbitrary measure to understand the complexity of a user story.** e

1.

## WHAT IS SWAG

Swag stands for **Scientific Wild Ass Guess**, and is a slang word meaning a **rough estimate from an expert in the field**, based on his/her **experience and intuition**. A Scientific Wild Ass Guess is **more precise** than a Wild Ass Guess, since Swag is estimated by an expert

In the Agile world, **Swag is for portfolio items** AS IN what **story points are to backlog items**. Swag provides **a way to compare the size, time, and effort** that it will take to **complete a set of features** without going through **detailed backlog estimating activities**

Swag does not have to be the **same unit** as your backlog item estimate.

## 2. Forced Ranking

Stakeholders are asked to **prioritize all the PBI** in order forcing them to give a **unique increasing priority number** to each increment.

Highest priority is with lowest total

To make sure that no two product backlog items have the same priority

Sno		S1	S2	S3	S4	TOTAL
1	R1	1	4	2	3	10
2	R2	2	3	1	2	8
3	R3	3	2	4	2	11

- Forced ranking is a controversial **workforce management tool** that uses **intense yearly evaluations** to identify a **company's best and worst performing employees, using person-to-person comparisons**. In theory, each ranking will improve **the quality of the workforce**
- Forced ranking is a **business tool for managing limited resources**. This becomes especially **useful in software projects** for **scheduling features and functions, prioritizing bugs, scheduling resources for work activities**, selecting **infrastructure and organizing testing and deployment activities**
- All forced ranking of business requirements must be business driven
- Used in Scrum in Backlog Grooming
- It is also incorporated indirectly in the "minimum viable product" or MVP concept

- The **benefits of forced ranking** are to **manage limited resources** towards **achieving the essential parts of a system to meet primary goals**

### 3. Continuous Integration

#### 4. Different tools for Continuous Integration

Continuous integration (CI) is a **software engineering practice** in which **isolated changes are immediately tested and reported on when they are added to a larger code base**. The goal of CI is to **provide rapid feedback** so that if a defect is introduced into the code base, it can be identified and corrected as soon as possible. Continuous integration software tools can be used to automate the testing and build a document trail.

- Jenkins - It is an open source automation server written in **Java**. **Jenkins helps to automate** the non-human part of the **whole software development process**, with now common things like continuous integration, but by further empowering teams to implement the technical part of a Continuous Delivery.

- Bamboo - It is a continuous integration server from Atlassian, the makers of JIRA, Confluence and Crowd. It is used to **build, test and deploy applications automatically as per requirements and thus helps speed up the release process**. Bamboo supports builds in a number of programming languages

- Codeship – It is a **continuous deployment solution** that's focused on **being an end-to-end solution for running tests and deploying apps**.

### 5. BLOCKERS VS IMPEDIMENTS:

- An impediment is **anything that slows down or diminishes the pace of the Team**. When the Team is **confronted with impediment** (or obstacles), the Team **could move forward but in advancing they may generate waste**. Or the whole process of **making progress is more difficult** than it should be

- In contrast, **a blocker is anything that stops the delivery of the product**. Without the elimination of the blocker, the **Team cannot advance at all**. **Clearly, eliminating blockers is more important than resolving impediments**

### 6. User Story Mapping

A user story map arranges **user stories into a useful model** to help understand the **functionality of the system, identify holes and omissions in your backlog, and effectively plan holistic releases** that deliver *value to users and business with each release*

The main idea behind story mapping is that single list product backlogs are a terrible way to organize and prioritize the work that needs to be done.

Story mapping can be a **very effective way to communicate flow and priority on some projects.**

A story map is organized like this

- The horizontal axis represents usage sequence
- The vertical axis stands for criticality.
- Groups of related user stories can be grouped as Activities.

What does a good user story look like? What is its structure?

A good user story:

- has a description,
- defines acceptance criteria,
- can be **delivered within a single sprint**
- has all UI deliverables available?
- has all (probable) dependencies identified,
- defines performance criteria,
- defines tracking criteria, and
- is estimated by the team

## **7. FEATURE TEAMS:**

- A feature team, is a long-lived, **cross-functional, cross-component team** that completes many **end-to-end customer features—one by one**
- The characteristics of a feature team are listed below:
  - long-lived—the team stays together so that they can ‘jell’ for higher performance; they take on new features over time
  - cross-functional and cross-component
  - ideally, co-located
  - work on a **complete customer-centric feature, across all components and disciplines (analysis, programming, testing, ...)**
  - composed **of generalizing specialists**

in Scrum, typically  $7 \pm 2$  people

8.

### **THEMES, FEATURES, EPICS & USER STORIES:**

- **Themes** may be thought of as **groups of related stories**. Often the stories all contribute to a **common goal** or **are related in some obvious way**, such as all focusing on a **single customer**. However, **while some stories in a theme may be dependent on one another**
  
- “Theme” is a collection of user stories. We could put a rubber band around that group of stories I wrote about monthly reporting and we'd call that a “theme.” Sometimes it's helpful to think about a group of stories so we have a term for that. Sticking with the movie analogy above, in my DVD rack I have filed the James Bond movies together. They are a theme or grouping.
  
- A **feature** is a **distinct element of functionality which can provide capabilities to the business**. It generally takes **many iterations to deliver a feature**. A **user story is a part of the feature**. **By splitting a feature in smaller stories**, the user can give early feedback to the developers to issues quickly.

• My features might be:

- 
- Display informative Home screen
- User Registration
- User Login
- Display Products
- Display Shopping Cart
- Add products to Shopping Cart
- 

• And then, User stories would be developed off those:

- 
- Display informative Home screen
  - - As a User, I want to access the Home scree, so that I can enter the website
  - - As a user, I want to be able to see specials, so that I can see the current deals
- 
- User Registration
  - - As a user, I need to be able to register as a new user, so that I can have more access to the site
  - - As a user, I want to be able to register with my Google account
- 
- User Login
  - - As a user, I need to be able to login with my username/password, to gain access to the site
  - - As a user, I want to be able to login with my Google account, to gain access to the site

- Display Products
- -
- -
- Display Shopping Cart
- -
- Add products to Shopping Cart

- **Epics resemble themes in the sense that they are made up of multiple stories.** They may also resemble stories in the sense that, at first, many appear to simply be a "**big story**." As opposed to themes, however, **these stories often comprise a complete work flow for a user.** But there's an even more important difference between themes and epics. While **the stories that comprise, an epic may be completed independently, their business value isn't realized until the entire epic is complete.**
- **User Stories:**
  - It is a **software system requirement** formulated as **one or two sentences** in the everyday or business language of the user
  - Each story is limited; so, it fits on a **3x5in card**
- When are User Stories written?
  - **Throughout the Agile project.** Usually, story-writing workshop is held near the start of the agile project
  - **Everyone on the team** participates with the goal of **creating a product backlog** that **fully describes the functionality** to be added over the course of the project or a 3-6 months release cycle within
  - Some of these agile user stories will undoubtedly be epics. **Epics** will later be **decomposed into smaller stories** that fit more readily into a **single iteration**. Additionally, *new stories can be written & added to the product backlog at any time by anyone*
- How to write User Stories?
  - A user story briefly explains:
    - the person using the service (**actor**)
    - what the user needs the service for (**narrative**)
    - why the user needs it (**goal**)
  - As a <type of user>, I want <some goal> so that <some reason>

## 9. PBIs vs TASKS:

The PBI:

- is the requirement aka "the what"
- is **what you talk about with a customer.**

- It's **what shows up on the Daily Project Update (DPU) for the sprint....** again, because the DPU is customer facing.
- It's **what the customer will talk about and refer in terms of estimates and budget.**
- Might comprise one or more tasks.
- **Is business oriented and described in business oriented / domain style language that the customer understands.**
- Is what gets **tested and accepted in User Acceptance Testing (UAT)**

The Task:

- Is a piece of work **required to materialize the PBI** (requirement)
- Not something you talk about with a customer
- Doesn't show up on the DPU because you don't talk about them with customers
- Is estimated but has its estimates rolled up into the PBI
- Is a child to one and only one requirement.
- Can be described (and often is) using technical jargon
- **Tested internally and signed off by test**
- **Not accepted or tested in isolation by the customer** (they don't know they exist)

## 10. TEST DRIVEN DEVELOPMENT (TDD):

Negative test cases, positive cases

- "Test-driven development" refers to a style of programming in which **three activities are tightly interwoven: coding, testing (in the form of writing unit tests) and design (in the form of refactoring).**
- It can be succinctly described by the following set of rules:
  - write a "single" unit test **describing an aspect of the program**
  - run the test, which should fail because the **program lacks that feature**
  - write "**just enough**" code, the simplest possible, to make the test pass
  - "**refactor**" the code until it conforms to the simplicity criteria

repeat, "accumulating" unit tests over time, IT HELPS Understand all the possible invalid options

many teams report significant reductions in **defect rates**, at the, **IMPROVED DESIGN QUALITY**

## 11. ATDD (ACCEPTANCE TEST DRIVEN DEVELOPMENT):

- Acceptance Test Driven Development (ATDD) **involves team members with different perspectives** (customer, development, testing) **collaborating to write acceptance tests in advance of implementing the corresponding functionality.**
- These **acceptance tests** represent the **user's point of view** and **act as a form of requirements** to describe **how the system will function**, as well as serve as a way of **verifying that the system functions** as intended. In some cases, the team automates the acceptance tests

the creation of interfaces specific to functional testing

Concord ion

12. **BDD (BEHAVIOR DRIVEN DEVELOPMENT)**: This simple technique, however, **can often quickly direct you to the root of the problem**. So, whenever a system or process isn't working properly, give it a try before you embark on a more in-depth approach.

5 Whys in troubleshooting, **quality improvement and problem solving**, but it is best for simple or moderately difficult problems

Behavior Driven Development (BDD) is a **synthesis and refinement of practices stemming from Test Driven Development (TDD) and Acceptance Test Driven Development (ATDD)**. **BDD augments TDD and ATDD with the following tactics:**

#### **Test outside in**

- Apply the "**Five Why's**" principle to each proposed user story, so that its purpose is clearly related to business outcomes
- **in other words, implement only those behaviors which contribute most directly to these business outcomes**, so as to minimize waste
- **describe behaviors, customers' expectations are taken into account**
- Talked about behavior than implementation
- write failing test cases, code to pass, refactor, acceptance, add behaviors write test case check , customer is the king

describe behaviors in a **single notation which is directly accessible** to domain experts, testers and developers, so as to improve common

#### 13. **APPROACHES TO RELEASE PLANNING:**

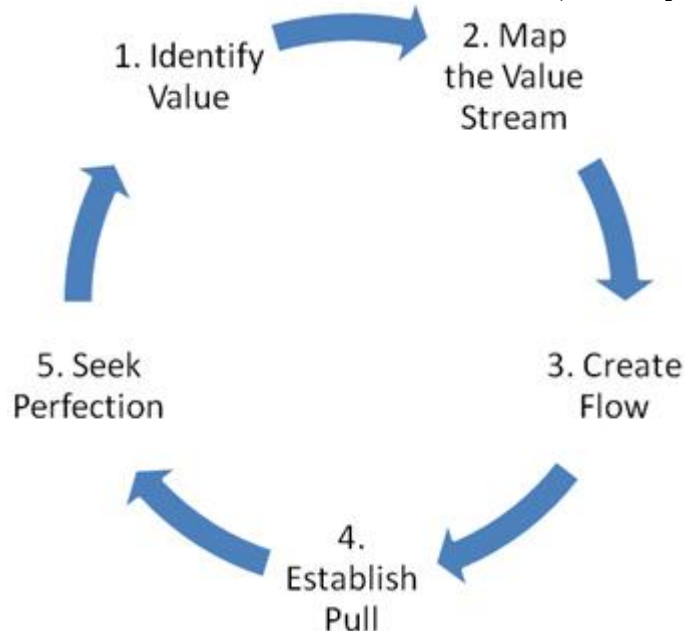
It is a guideline that reflects expectations about ***which features will be implemented and when they are completed***. It also serves as a *base to monitor progress within the project*. Releases can be **intermediate deliveries** done during the project or the **final delivery at the end**

- **Date Driven**
  - When you calculate **the amount of work you can finish by a particular date**
- **Feature Driven**
  - When you calculate **how long it will take to finish a specific amount of work**

#### 14. SAFE

Scaled Agile Framework (or SAFe) is an Agile software development framework designed by Scaled Agile, Inc. It consists of a knowledge base **of integrated patterns intended for enterprise-scale Lean-Agile development**. Its proponents consider SAFe to be **scalable and modular**, allowing an organization **to apply it in a way that suits its need**.

SAFe is based on a number of **immutable, underlying [Lean](#) and [Agile](#) principles**



#### 15. FORMAT FOR USER STORY & ACCEPTANCE CRITERIA:

- How to write User Stories?
  - A user story briefly explains:
    - the person using the service (**actor**)
    - what the user needs the service for (**narrative**)
    - why the user needs it (**goal**)

As a <type of user>, I want <some goal> so that <some reason

- *Acceptance Criteria as “**Conditions** that a software product must satisfy to be accepted by a user, customer or other stakeholder.”* Google defines them as “Pre-established **standards or requirements** a product or project must meet.”
- Acceptance Criteria are a **set of statements, each with a clear pass/fail result**, that specify both functional (e.g., minimal marketable functionality) and non-functional (e.g., minimal quality) requirements applicable at the current stage of project integration. These requirements represent “**conditions of satisfaction**.” **There is no partial acceptance: either a criterion is met or it is not.**



- These criteria define the boundaries and parameters of a User Story/feature and determine when a story is completed and working as expected. They add certainty to what the team is building.
  - **Functional Criteria:**
  - **Non-functional Criteria:**
  - **Performance Criteria:**
- Clearly defined Acceptance Criteria are crucial for **timely and effective delivery** of the functionality defined in the User Stories, which ultimately determines the success of the project

## 16. ORID

### **ORID Technique:**

ORID stands for Objective, Reflective, Interpretive and Decisional.

Interpreted as a **consecutive order of stages**, it reflects the **natural order in which humans think through an issue**

Engage the team in a **series of questions that help slow down decisions and gather insights before moving to recommendations:**

- O: Ask objective questions first. **"What was happening? What did you notice?"**  
Have team members work in small groups to gather these items.
- R: Ask reflective questions next. **"What reaction did you have to that? What was challenging or helpful?"**
- I: Ask interpretive questions. **"What does that say about how we work? What might be some recommendations for our work?"**
- D: Ask decisional questions. **"Given what we have recorded here, what new agreements or practices might we invite into our next iterations?"**

## 17. PSEUDO SOLUTIONS:

- Pseudo Solutions are the **solutions that were initially developed for a different PBI or Task, but can be applied to another one**
- For e.g. if a PBI has two tasks, and the development of one task leads to the solution of another.
- Pseudo Solutions may or may not be beneficial to the team

## 18. Advantages of SCRUM:

- Helps company save time & money
- Enables project where the **business requirements documentation is hard to quantify to be successfully developed**
- **Fast moving, cutting-edge developments** can be quickly coded & tested using this method, as a mistake can be easily rectified
- **Tightly controlled method** which insists on **frequent updating of progress in work** through regular meetings. Thus, there is clear visibility of project development
- Like any other agile methodology, this is **also iterative in nature**. It requires **continuous feedback from the user**
- Due to short sprints and constant feedback, it becomes **easier to cope with the changes**
- Daily meetings make it possible to measure **individual productivity**. This leads to the **improvement in the productivity of each of the team members**
- **Issues are identified well in advance** through the daily meetings & hence can be resolved speedily
- It is **easier to deliver** a quality product in a scheduled time
- Agile Scrum can work with any technology/programming language but is particularly useful for fast moving web 2.0 or new media projects
- The **overhead cost in terms of process & management is minimal** thus leading to a quicker, cheaper result

Additional read - <http://www.dummies.com/careers/project-management/10-key-benefits-of-scrum/>

#### **Disadvantages of SCRUM:**

- Agile Scrum is one of **the leading causes of scope creep** because unless there is definite **end date**, the project management stakeholders will **be tempted to keep demanding new functionality is delivered**
- **If a task is not well defined**, estimating project **costs and time will not be accurate**. In such a case, the task can be spread over several sprints
- If the **team members are not committed**, the **project will either never complete or fail**
- It is **good for small, fast moving projects** as it works well only with **small team**
- This methodology **needs experienced team members only**. If the team consists of people who are novices, the project cannot be completed in time
- Scrum works well when the **Scrum Master trusts** the team they are managing. If they practice **too strict control over the team members**, it can be **extremely frustrating** for them, **leading to demoralization and the failure** of the project
- If any of the **team members leave** during a development it can have a **huge inverse effect on the project development**

#### **WATERFALL vs SCRUM:**

1. Linear sequential software development
2. Project scope cannot be adjusted during project as it may kill the project
3. Requirements churn is not allowed
4. detailed documentation is required during the beginning and closing of the phases
5. No working software is developed until development phase

1. Iterative incremental timeboxed software development tool
2. project scope/sprint goal can be adjusted during the project
3. Enables and welcomes requirements churns
4. user stories invest in business language is needed
5. PSPI is developed in short sprint

### **WATERFALL-SCRUM HYBRID:**

1. **Integration Longer development cycles and constantly changing requirements**
2. ***Implementing team work “agile” while the hardware development and product managers use traditional waterfall approach***
3. **Tight, continuous integration from product concept until validation and production**
4. Hybrid model is best for reusing codes

Purpose:

This hybrid approach, however, is for coexistence sake; to be able to satisfy both processes for the foreseeable life of the project.

The hybrid Scrum methodology proceeds as follows:

- **Release Planning** - a hybrid process element - is a series of planning sessions conducted prior to each **release cycle**. Planning is initially accomplished with **IEEE 830 requirements only**.
- **Predefined Release Schedule** - a hybrid artifact - is a **regularly refined product of each Release Planning**. At a high level, this shows **the entire release schedule for the project**.
- **Product Backlog** - a hybrid artifact - is a change to the normal Scrum Product Backlog. It is populated with **IEEE 830 Requirements**, and **it is groomed each release cycle through the Release Planning**.
- **User Story Workshop** - a normal Scrum process element.
- **Release Backlog** - a hybrid artifact - is an additional backlog which is used **to define the scope of the current release**. It is populated by the Scrum Team and PMO in **User Story Workshops**, and it serves the purpose that a Product Backlog serves in normal Scrum.

- **Sprint Planning Session** - a normal Scrum process element.
  - **Sprint Planning Review Artifact** - a hybrid artifact - is a product of the Sprint Planning Session which captures the scope of the current sprint which was just captured
  - **Sprint Backlog** - a normal Scrum process element.
  - **Sprint and Daily Scrums** - are normal Scrum process elements.
- 
- **Traceable UAT Artifact** - a hybrid artifact - is a **clearly defined UAT for each user story** which is *used to test features and trace their verification from the signed UAT back to the IEEE 830 requirements which they satisfy.*
- 
- **Sprint Review** - a normal Scrum process element.
  - **Sprint Review Artifact** - a hybrid artifact - is a product of the Sprint Review which captures the *satisfactory accomplishment and acceptance of each of the stories in the sprint*, with traceability back to original IEEE 830 requirements.
  - **Shippable Product** - a normal Scrum process element

## 21. Principles of agile methodology

1. Individuals and interactions over process and tools
2. Working software than comprehensive document
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

## 22. APPROACHES TO SPRINT PLANNING:

- **Velocity driven Sprint Planning**
- **Commitment driven Sprint Planning**

### VELOCITY DRIVEN PLANNING:

- Velocity-driven sprint planning **is based on the premise that the amount of work a team will do in the coming sprint is roughly equal to what they've done in prior sprints.** This is a very valid premise
- The steps in velocity-driven sprint planning are as follows:
  - Determine the **team's historical average velocity.**
  - Select a **number of product backlog items equal to that velocity.**
  - Some teams stop there. Others include the additional step of:

- Identify the **tasks involved in the selected user** stories and see if it feels like the **right amount of work**. Some go further & ..
- Estimate the tasks and see if the sum of the work is in line with past sprints

### **COMMITMENT DRIVEN PLANNING: (Read from Mountain Goat):**

A commitment-driven sprint planning meeting involves the product owner, ScrumMaster and all development team members. The product owner brings the **top-priority product backlog items into the meeting and describes them to the team**, usually starting with an overview of **the set of high-priority items**

it is an approach in which the team selects a Pbi and asks the commitment of the team and then they select the next pbi and so on while sanity test is conducted by the scrum master to calculate the velocity and capacity and they compare it with the current one

Commitment driven approach: The steps involved are Selecting Items or PBIs, Determining the Tasks and Calculating hours to finish each task (rough estimate), Team members ask for commitment (can we commit?), repeat the process with other stories or PBIs, Sanity check done by the scrum-master to find out the story points and velocity which will be compared with average or recent velocity.

A mock up is about the realistic rep, wireframe is about a functionality

## 23. REQUIREMENTS CHURCH

### **REQUIREMENTS CHURN & WHO IS RESPONSIBLE?**

- Requirements churn is the ***change to backlog items from the time they are entered until the product goes into production***
- **10-15%** churn is not a problem. But **50-200%** churn means **someone is wasting a lot of time putting stuff** in the backlog that is going to change, they do not have much of a **sense of what is certain and what is not**
- Either the *Product Owner* could be responsible for this, as he/she would be adding any incoming requirements to the Product Backlog. Or it could be someone from the Development team adding items that do not add value, which shows that there is uncertainty about requirements

**Shorter release cycles** , **set a limit then re access**, reduce review fatigues , improve visibility as in show the impact to change

- Capacity of a Team

24. Vertical Splitting is more useful. If a **user story is split vertically**, it is **split in such a manner that the smaller items still generate some business value**. The **functionality** will not be **split across** technical layers or tasks, but **across functional layers**

To give customers the essence of a **walking skeleton**, deliver value to the end customers , not db layer is what we want but finds value in UI, WS, DB layer

Split by workflow steps

Split by business rules

Split by happy / unhappy flow

Horizontal Splitting involves **breaking up user stories by the kind of work** that needs to be done or the application layers that are involved. So, work that has to be done for the UI, databases, a front-end, a back-end and testing is **split up into groups of tasks**.

This is typically how teams work in a more traditional (waterfall) development context. Although this approach has its use in waterfall development, it is most certainly not recommended for breaking up user stories when Scrumming.

**User stories belong to one architecture layers of the system, refer to specific layer or operation**

25. Roles:

Product owner, scrum development team, Scrum master

26. Meetings:

SPM, SEM(DS), SRM, SRM, BRM

27. Artifacts:

PB, PBI, SB, ST, SBC, P/RBC

28. PRIORITIZATION TECHNIQUES

**MoSCoW:**

- **M - MUST** have this

- S - SHOULD have this if at all possible
- C - COULD have this if it does not affect anything else
- W - WON'T have this time but would like in the future

### **KANO MODEL: [Refer online]**

The **Kano model** is a theory of **product development and customer satisfaction** developed in the 1980s by Professor Noriaki Kano, which classifies **customer preferences** into four categories.

- **Performance**  
Some product features behave as what we might intuitively think that Satisfaction works: the more we provide, the more satisfied our customers become.
- **Must-be**  
Other product features are simply **expected** by customers. If the product doesn't have them, it will be considered to be incomplete or just plain bad. This type of features is usually called *Must-be* or Basic Expectations.
- **Attractive**  
There are **unexpected features** which, when presented, **cause a positive reaction**. These are usually called *Attractive*, *Exciters* or *Delighters*.
- **Indifferent**  
Naturally, there are also features towards which we feel **indifferent**. Those which their presence (or absence) doesn't make a real difference in our reaction towards the product.

### **VALUE STREAM MAPPING: [Check PDF for more]**

- It **visualizes the whole process** from idea to customer release as a **series of connected tasks**. Both **value and non-value adding tasks should be defined and tracked from customer requirement to delivery**
- By looking at the holistic representation of your complete development cycle, and whether or not the individual steps are adding value, **you can create a visual model of your process and easily identify waste in the system**

### **WALKING SKELETON:**

- **Tiny implementation** of a system **that performs a small end-to-end function**
- It need not use the final architecture, but it **should link together the main architectural components**
- A **vertical slicethrough** all the **architectural layers**
- Minimal implementation that proves the architecture through some end-to-end functionality
- The architecture & functionality can then evolve in parallel
- Helps get **early feedback** if chosen architecture & technology is suitable
- **Build quality becomes consistent as the architecture is defined early**

### **BUSINESS VALUE BASED:**

- In this case, each **requirement carries a business value it could potentially generate to the company**. The business value would **be decided either** by the **Product Owner or the Product owner team**.
- The **requirement with highest business value is implemented** during earlier releases.

### **TECHNOLOGY RISK BASED:**

- In this method, **requirements are prioritized based on the risk associated in implementing it**. **The risk is typically based on the technology**.
- The requirement with highest technology risk is implemented during the earlier iterations.

### **100 POINT METHOD:**

- The 100-point method is a prioritization method that can be used to **prioritize items in a group environment**. **Each person** within the group **is given 100 points** which they can distribute as **votes across the available items**. The votes do not need to be distributed equally; rather a weighted distribution can be used to reflect the higher priority that some items warrant.
- Imagine that a group was trying to prioritize 5 items. **A person could decide that each item is of equal importance** and assigned 20 points to each. Or, they could **decide that item 1 is more important than 2** which is more important than 3, and so on, and therefore spread the **votes out in a weighted fashion** where item 1 gets 40 votes, item 2 gets 30 votes, item 3 gets 15 votes, etc, until all of the votes are allocated. Each person allocates their 100 points. **Then all of the votes are added to determine the final vote count for each item resulting in the prioritized list**.



## 29. ESTIMATION TECHNIQUES:

### PLANNING POKER:

The reason for using the Fibonacci sequence is to **reflect the inherent uncertainty in estimating larger items.**

**Planning Poker is an agile estimating and planning technique that is consensus based.** To start a poker planning session, the **product owner or customer reads an agile user story or describes a feature** to the estimators.

**Each estimator is holding a deck** of Planning Poker cards with values **like 0, 1, 2, 3, 5, 8, 13, 20, 40 and 100**, which is the **sequence we recommend**. The values represent the **number of story points, ideal days, or other units in which the team estimates.**

The **estimators discuss the feature**, asking questions of the product owner as needed. When the feature has been fully discussed, each **estimator privately selects one card to represent his or her estimate**. All cards are then **revealed at the same time**.

If all estimators selected the **same value, that becomes the estimate**. If not, the estimators **discuss** their estimates. The **high and low estimators should especially share their reasons**. After further discussion, each estimator **reselects an estimate card**, and all cards are again revealed at the same time.

The poker planning process **is repeated until consensus is achieved or until the estimators decide that agile estimating and planning of a particular item needs to be deferred until additional information can be acquired.**

### When should we engage in Planning Poker?

Most teams will hold a Planning Poker session **shortly after an initial product backlog is written**. This session (which may be spread over multiple days) is used **to create initial estimates** useful in scoping or sizing the project.

**Because product backlog items** (usually in the form of user stories) will **continue to be added** throughout the project, most teams will find it helpful to conduct subsequent **agile estimating and planning sessions once per iteration**. Usually this is done a few days before the end of the iteration and immediately following a daily standup, since the whole team is together at that time anyway.

### T-SHIRT SIZE:

Occasionally team **members align the story points with the hours of effort**, which **can create confusion**. To avoid this, it may be more effective to **switch to a nonnumeric estimation technique**.

Teams who don't enjoy playing with numbers that much use T-shirt size as another popular technique. So, when the teams tend to overanalyze the numbers, T-shirt sizes come in handy, as T-shirt sizes are easier to relate to.

With T-shirt sizing, the team is asked to **estimate whether they think a story is extra- small, small, medium, large, extra-large, or double extra-large**. **By removing the implied precision** of a numerical score, the **team is free to think in a more abstract way about the effort involved in a story**.

Some teams even adopt creative approaches, such as **using dog breeds**, to estimate stories. For example, "That story's clearly a Chihuahua, but the other one is a Great Dane." **Engaging the fun, creative side of the team while they're estimating technical stories** can be effective at getting them out of their **analytical thought processes** and into a **more flexible, relative mindset**.

There are **some practical issues to consider** when adopting T-shirt sizing for story estimation. For one, **non-numerical scales are generally less granular**. Although that can speed the voting process by **reducing the number of options**, it may also **reduce the accuracy of velocity estimates**.

In addition, the ability to compare stories with each other can be a **little bit more complicated**, because there is **no clear mathematical relationship between a medium and an extra-small**. **T-shirt size scales** also require **extra effort on the part of the person** coordinating the Agile process. The **T-shirt sizes need to be converted to numerical values** for the sake of tracking effort over time and charting an estimated velocity for the team. For this reason, while T-shirt sizes can be very effective for teams just starting out with Agile, eventually **it's a good idea to move the team toward a more rational numerical scale**

## **RELATIVE MASS VALUATION:**

**When adopting Agile as a new technique for a team, frequently there will be a large backlog of stories that need to be estimated all at once.**

**Relative mass valuation is a quick way to go through a large backlog of stories and estimate them all as they relate to each other.** To use this approach, **first write up a card for each story.** Then, set up a **large table** so that the stories can be moved around easily relative to each other.

Start **by picking any story, then get the team to estimate** whether they think that it is **relatively large, medium, or small.** If it's a **large story, place it at one end of the table.** If it's a **small story, it goes at the other end of the table.** A **medium story goes in the middle.** *Now select the next story, and ask the team to estimate if it's more or less effort than the story that you just put down.* **Position the story card** on the table relative to the previous card, and go to the next card.

By using this technique, it's possible to go through 100 or more backlog stories **and estimate their relative effort in as little as an hour.** Everyone on the team **will feel a sense of accomplishment when they see the scope of their work laid out in front of them,** estimated in order of effort.

The **next step is to assign point values** based on the **position of the stories** on the table. **Start** with the **easiest** story that is worth assigning points to, and call it a 1. Then, move up the list of cards, assigning a value of 1 to every story until you get to one that **seems at least twice as difficult as the first one.** That story gets a 2. You may need to remind the team not to get caught up in the fine details. The idea is to get **a rough point estimate, not a precise order.**

Ultimately, any story may be completed in any order based on **the business value and priority assigned** by the product owner, so all the team needs to estimate is how many points one story will take relative to another

## **BUCKET SYSTEM:**

The Bucket System is a way to estimate large numbers of items with a small- to medium-sized group of people, and to do it quickly. The Bucket System has the following qualities that makes it particularly suitable for use in Agile environments:

- It's fast! A couple of hundred items can be estimated in as little time as one hour.
- It's collaborative. Everyone in a group participates roughly equally.
- It provides relative results, not absolute estimates (points vs. hours).
- The results are not traceable to individuals and so it encourages group accountability.
- Works with teams to estimate effort or with stakeholders to estimate value.

The Bucket System of estimation works as follows:

1. Set up the physical environment, as per the diagram below. Ensure that all the items to be estimated are written on cards.
2. Choose an item at random from the collection and read it to the group and place it in the "8" bucket. This item is our first reference item.
3. Choose another item at random from the collection and read it to the group. The group discusses its relative position on the scale. Once consensus has been reached, put the item in the appropriate bucket.
4. Choose a third item at random and, after discussion and consensus is reached, place it in the appropriate bucket.
5. If the random items have clearly skewed the scale toward one end or the other, rescale the items (e.g., the first item is actually very small and should be in the "1" bucket).
6. Divide and conquer. Allocate all the remaining items equally to all the participants. Each participant places items on the scale without discussion with other participants. If a person has an item that they truly do not understand, then that item can be offered to someone else.
7. Sanity check! Everyone quietly reviews the items on the scale. If a participant finds an item that they believe is out of place, they are welcome to bring it to the attention of the group. The group then discusses it until consensus is reached, and it is placed in one of the buckets.
8. Write the bucket numbers on the cards so that the estimates are recorded

## DOT VOTING:

Dot voting is a technique that allows participants to vote their preferences **among a set of items by placing a colored dot on items that they believe are higher priority than other items**. Items with **more dots are higher priority than items with fewer dots**. This technique is frequently used during the **sprint retrospective activity**.

For example, perhaps there is **a list of ten items identified in a retrospective**, and the team wants to **narrow the list to a couple they will work on**. Everyone has three dots to vote on the retrospective items. **Some will put one dot each on three different items, others might choose to put all three dots on one option, or perhaps two dots on one option and one on another.**

The method is summarized as follows:

- **Post the user stories on the wall** by using yellow sticky notes or in some manner that enables each item to receive votes.

- **Give four to five dots to** each stakeholder.
- Ask the stakeholders to place their votes. Stakeholders should apply dots (using pens, markers, or, most commonly, stickers) under or beside written stories to show which ones they prefer.
- Order the product backlog from the most number of dots to the least.
- When you are done with this first pass, it is almost certain that the **stakeholders will not be completely happy with the outcome of the vote**. If that is the case, you should review the voting and **optimize it**

Here's what you can do:

- Arrange the votes into three groups to represent high, medium, and low priorities.
- Discuss stories in each group.
- Move items around to create a high-priority list.
- Make a new vote with items in the high-priority list

### **AFFINITY MAPPING:**

Items are **grouped by similarity – where similarity is some dimension that needs to be estimated**. This is usually a very physical activity and requires a relatively small number of items (20 to 50 is a pretty good range). The groupings are then associated with numerical estimates if desired

How big the story should be?

1/6 – 1/10 OF the whole sprint (Epics to user stories while breaking down )

### **30. RETROSPECTIVE TECHNIQUES:**

#### **Silent Writing:**

- In this technique, **each member** in the meeting is given a **stack of post-it notes** in which they **write one observation per Post-It note about the previous Sprint**. Discussion is held off until everyone is finished writing.
- The Scrum Master then facilitates dot-voting to **identify top issues** from the paper groupings. **This method identifies top issues** while giving everyone in the room a chance to be heard in a collaborative **environment** – and it's very helpful for **engaging introverts who may not want to speak up until they are comfortable**. It also allows the team to get on the same page (literally). Once the team identifies

**an issue to discuss** (a challenge, for example) they **can work together to** figure out a solution.

### Happiness Histogram:

- **Prepare a visual with a horizontal scale from 1 (Unhappy) to 5 (Happy).** Using **sticky notes or virtual techniques**, ask each person to signify where they would place themselves on the happiness scale. After everyone has identified where they are, have each person comment.
- Depending on **how you want to facilitate** this, you can **either have the team do a deep dive right then** and there or postpone a more **nuanced conversation** about what people observe until later in the retrospective.
- The reason this is referred to as a histogram is that if one person has **the same happiness score as another**, they place their sticky **above the** one with the same score

### ORID Technique:

ORID stands for Objective, Reflective, Interpretive and Decisional.

Interpreted as a consecutive order of stages, it reflects the natural order in which humans think through an issue

Engage the team in a series of questions that help slow down decisions and gather insights before moving to recommendations:

- **O:** Ask objective questions first. "What was happening? What did you notice?" Have team members work in small groups to gather these items.
- **R:** Ask reflective questions next. "What reaction did you have to that? What was challenging or helpful?"
- **I:** Ask interpretive questions. "What does that say about how we work? What might be some recommendations for our work?"

- D: Ask decisional questions. "Given what we have recorded here, what new agreements or practices might we invite into our next iterations?"

### **The Wheel (also known as the Starfish):**

1. Draw a large circle on a whiteboard and divide it into five equal segments
2. Label each segment 'Start', 'Stop', 'Keep Doing', 'More Of', 'Less Of'
3. For each segment pose the following questions to the team:
  1. What can we start doing that will speed the team's progress?
  2. What can we stop doing that hinders the team's progress?
  3. What can we keep doing to do that is currently helping the team's progress?
  4. What is currently aiding the team's progress and we can do more of?
  5. What is currently impeding the team's progress and we can do less of?
4. Encourage the team to place stickies with ideas in each segment until everyone has posted all of their ideas
5. Erase the wheel and have the team group similar ideas together. Note that the same idea may have been expressed in opposite segments but these should still be grouped together
6. Discuss each grouping as a team including any corrective actions

### **The Sail Boat (Or speed boat. Or any kind of boat):**

1. Draw a boat on a whiteboard. Include the following details:
  1. Sails or engines – these represent the things that are pushing the team forward towards their goals
  2. Anchors – these represent the things that are impeding the team from reaching their goals
2. Explain the metaphors to the team and encourage them to place stickies with their ideas for each of them on appropriate area of the drawing
3. Wait until everyone has posted all of their ideas
4. Have the team group similar ideas together
5. Discuss each grouping as a team including any corrective actions going forward

### **Mad Sad Glad:**

1. Divide the board into three areas labelled:
  1. Mad – frustrations, things that have annoyed the team and/or have wasted a lot of time
  2. Sad – disappointments, things that have not worked out as well as was hoped
  3. Glad – pleasures, things that have made the team happy
2. Explain the meanings of the headings to the team and encourage them to place stickies with their ideas for each of them under each heading

3. Wait until everyone has posted all of their ideas
4. Have the team group similar ideas together
5. Discuss each grouping as a team identifying any corrective actions

### **Guess who:**

- It's a retrospective from a different perspective. It is an 'empathy hack' retro designed for team members to see others' points of view by selecting a colleague's name from a hat. Team members are encouraged to play the role of their colleagues and view the world from a different perspective.
- Guess who helps to increase the power of talking about each other

### **Futurespectives:**

Scrum proposes that the retrospective is done at the end of the sprint (iteration). But you can also use retrospectives at the beginning of an iteration or when a new team is assembled. Such a retrospective is called a futurespective: A retrospective where you start from the goal and explore ways how to get there

In a futurespective teams places themselves in the future by imagining that their goal has been reached. They start such retrospectives by discussing the team goals to assure that team members build a common understanding. The goals are formulated and written down so that they are visible for everybody.

Optionally teams can write down which benefits they got from attaining their goals. If teams like to party they can even do a small celebration for having reached the goals, which can help to make teams aware of the importance of reaching their goal.

Next teams discuss their imaginary past and explore how they have gotten to their goals. There are two things that that team's questions themselves:

- What are the things that have helped us to get here?
- Which things made it hard for us to reach our goal?

Optionally teams can also discuss:

- What did we learn as team along the way towards reaching our goal?



Teams can use sticky notes, flip-overs and/or white boards to capture the results from the discussions. When you do the exercise with remote teams you can use a Google Doc or tools like Lino or Group map.

Now the teams go back to the present. The results from exploring the past are used to agree how to work together in teams to reach the goal. This can for example be done by:

- Defining a Definition of Done
- Making a list of tools that will be used, processes/practices that teams will use, etc
- Defining actions that are needed to work together effectively

### **The Perfection Game:**

The Perfection Game can be used to get feedback on a product or service that has been provided. It is also a retrospective exercise usable to discover strengths and define effective improvement actions. The perfection game gives power to the teams and helps them to self organize and become more agile.

To get feedback with a perfection game you ask people to provide answers to the following questions:

- I rate the product/service ... on a scale from 1-10
- What I liked about it ...
- To make it perfect ...

People have to rate the value that they received on a scale from 1 to 10, based on how much value they think they could add themselves by improving the product or service. For example, when there is nothing that they think they can improve, they should rate with a 10. If they think that they could make it twice as valuable, they should give it a 5

### **Speed Dating:**

Agile speed dating is a good way to discuss delicate subjects. After the first sprints the easy problems are solved and the more difficult ones remain. To solve these difficult subjects agile speed dating is a recommended method.

Setting up the venue – 5 minutes

Before the group enters the space, place sets of chairs facing one another throughout the room. Explain the rules to your team and have them divide into two groups. Group A will remain stationary. Group B will rotate clockwise upon hearing the buzzer or bell. Each individual within both groups should be given a card that contains their name.

#### Speed Dating – 5 minutes per pair

Assuming you have 10 teammates, there will be 5 rotations in total. Each encounter should be time boxed to 5 minutes. This gives each teammate the chance to voice their thoughts around the iteration and have open dialogue around opinions they may share or even differences of opinion. As the host, it is your responsibility to sound a buzzer at each 5-minute interval so that Group B can rotate to the next pairing and begin their discussion promptly.

#### Speed Dating Retrospective – 5 minutes

By now, the rotation has completed and each member of Group A has had a conversation with each member of Group B. Instruct each teammate to write the name of someone in the opposite group whom they felt either the most in sync with in regard to their iteration thoughts, or someone that shared something of great value. Once you collect the cards, you will take a few moments to hopefully find a few that match.

#### Speed Dating Results – 15 minutes

Invite the matches up to the front of the room and congratulate them for recognizing and appreciating one another's opinions. Have each one of them take a moment to explain something that stuck out around their conversation.

#### Speed Dating "Duds" / Wrap up – 15 minutes

While not everyone may have established a "match", we know there was still a lot of great dialogue. This is an opportunity for people to talk about meaningful conversation points that they shared during the last hour. Encourage your teammates to share points that were discussed by others vs. those that they initiated themselves.

#### **Safety Check:**

- **At the start of an agile retrospective you can do a safety check by asking people to write down how safe they feel in the retrospective.** If the score indicates that **people feel unsafe**, then that **will have serious impact on the**

**retrospective.** Here are some suggestions how you can deal with this when facilitating retrospectives.

- In a safety check people can use a score from 1 to 5. The scoring is done anonymously. A score of **5 means that they feel that they will be fully open and honest in the meeting** and are willing to talk about anything, **where a 1 means that they don't feel safe at all and don't want to speak up**

### 31. DEFINITION OF DONE AND READY

Definition of "Done" is the **criteria of satisfaction** at the product level **to avoid technical debt**

- Item should be **properly tested**.
- Item should be **refactored**.
- Item should **be potentially shippable**.

**The Definition of Ready** is a **set of rules or criteria** that **the team adopts as a guide** for **when a story can legitimately be moved from the backlog into a Sprint** -- either **during Sprint Planning** or during a Sprint **where all committed stories have been completed**. I've honestly rarely seen this adopted in practice, but could see the following criteria applying: should be immediately actionable

- Story has been **reviewed** and estimated by the team;
- Story is **complete in** format - User X needs to Y so they can Z;
- **Acceptance criteria** are clear and agreed upon;
- **Product Owner** has approved the story

### 32.ACCEPTANCE CRITERIA

#### ACCEPTANCE CRITERIA:

- User Stories are specific requirements outlined by various stakeholders as they pertain to the proposed product or service. Each User Story will have associated User Story Acceptance Criteria (also referred to as "Acceptance Criteria"), which are the objective components by which a User Story's functionality is judged.
- Acceptance Criteria are developed by the Product Owner according to his or her expert understanding of the customer's requirements. The Product Owner then communicates the User Stories in the Prioritized Product Backlog to the Scrum Team members and their agreement is sought. Acceptance Criteria should **explicitly outline the conditions that User Stories must satisfy**. Clearly defined Acceptance Criteria are crucial for timely and effective delivery of the functionality defined in the User Stories, which ultimately determines the success of the project

### 33. SPRINT ZERO

#### **SPRINT ZERO: (RESEARCH MORE):**

##### **SPRINT "0":**

Sprint zero is usually claimed as necessary because there are **things that need to be done before a Scrum project can start**. For example, **a team needs to be assembled**. That may involve **hiring or moving people onto the project**. Sometimes there is **hardware to acquire or at least set up**. Many projects argue for the need to write an **initial product backlog** (even if just at a high level) during a sprint zero.

- Idea is to prepare before the first sprint
- **It is not a sprint as there is no value delivered or a Potentially Shippable Product**
- It is **not always required**
- Scrum teams can **straight up dive in and start working**
  
- 2-3 days is a good length
  - Team engages
  - Start finding users
  - Customers & developers come together to form a team & chat
- **Good way to find more users**
- **Clarify goals** in Sprint Zero. Free-flowing format & brainstorming
- Discuss Product Backlog & start populating it with items

### 34. VELOCITY AND TYPES

#### **VELOCITY:**

- Velocity is a measure of the **amount of work a Team can tackle during a single Sprint and is the key metric in Scrum**. Velocity is calculated at the end of the Sprint by **totaling the Points for all fully completed User Stories**
- To calculate velocity of your agile team, simply **add up the estimates of the features, user stories, requirements or backlog items successfully delivered in an iteration**
- Velocity measures **how much functionality a team delivers in a sprint**
- Velocity measures a team's ability to **turns ideas into new functionality** in a sprint
- Velocity is measured in the same units as feature estimates, whether this is story points, days, ideal days, or hours that the Scrum team delivers – all of which are considered acceptable

## TYPES OF VELOCITY:

- **Optimal/Ideal Velocity:**
  - Maximum velocity the team has **achieved in the past and they would want to keep achieving**
- **Initial Velocity:**
  - Usually for **the first/conditional sprint if the team is mature**
  - It is **predicted on previous factors** or on capacity
- **Planned Velocity:**
  - **Based on Focus Factor**
  - **Planned for 1/3 of the effort**
  - IRL, ScrumMaster discusses with Product Owner which features they want by the end of the Sprint (Commitment driven)
- **Average Velocity:**
  - Velocity calculated based on the last few sprints

- What is Team's velocity
- Number of **story points delivered/demo in a Sprint is called velocity**. For example, if team planned 30 story point (Business value) worth of user stories in a sprint and able to deliver as planned then team's velocity is 30.
- What is Team's capacity?
- Total number of **available hours for a sprint is called Team's Capacity**. Available hours calculated **based on resources planned holiday and company** holiday if any.

## 35. CAPACITY

- What is Team's capacity?
- Total number of **available hours for a sprint is called Team's Capacity**. Available hours calculated **based on resources planned holiday and company** holiday if any.

## 36. FOCUS FACTOR:

- Focus factor is a simple mathematical formula for **forecasting the number of deliverables possible in an iteration**. In an Agile environment, this number can be arrived at by considering the capacity and velocity of a development team. The prerequisites for using focus factor are:
  - **Velocity calculated as an average of completed story points over the past three to five iterations**
  - Requirements (such as user stories) **estimated using story points**
  - The **development team's capacity (excluding nonworking days and time spent in meetings)**
- A team's focus factor is calculated as: **Focus Factor = Velocity/ Capacity**
- Example: If my team has **7 members who are productive for 6 days each**, and **as a team they have a velocity of 31**, then the focus factor is calculated as: **Focus factor = 31 / (7 \* 6) = 0.74**
- This **focus factor can now be used to forecast the deliverables for the future iterations**. So if only **five members are available during an iteration**, the **achievable story points** are calculated as: **Forecast = Focus factor \* Capacity = 0.74 \* 5 \* 6 = 22 [story points]**
- Maintaining the average velocity and focus factor of the past three to five iterations **provides a good forecast of the immediate next iteration**. This also helps us analyze and adjust the team's current performance and capability

### 37. Types of impediments

3 types of impediments are:

#### Scrum Master Owned Impediments:

“The team has been trying to have a meeting with the tech lead from another team for 4 days now. He is hard to catch up with and his calendar is nearly full.” **As a scrum master, I would coordinate/facilitate a face to face meeting so that the team is not consumed with administruvia <tiresome but needed details>**

#### Team Owned Impediments:

“**We need to test this feature but we don't have enough testers.**” As a Scrum master, I would guide the team to solving this themselves by “swarming” so that everyone becomes a tester so that we could get items to done

**A behavior** whereby team members with **available capacity and appropriate skills collectively work (swarm) on an item to finish what has already been started** before moving ahead to begin work on new items.

#### Organizational Impediments:

An example of an organizational impediment would be something like, “I have a person on my team who is constantly being asked by other managers to take on other duties. As a result, **he is never able to properly commit to and focus on the work for this project**” So I would work with managers to remove this impediment either by:

**Having the person replaced with** someone who had fewer distractions

**Having the other managers take those distracting** items off his list of things to do.

## 38 spikes and its types

### SPIKE:

- A **time boxed period to create a simple prototype (Technical investigation)**
- Spikes can either be planned to **take place in between Sprints** or, for larger teams, **a spike might be accepted as one of many Sprint delivery objectives**
- Spikes are *often introduced before the delivery of large or complex Product Backlog Items* in order to secure budget, expand knowledge, or produce a proof of concept
- The **duration** and objective(s) of a spike is **agreed between Product Owner and Development Team before the start**
- Unlike Sprint commitments, **spikes may or may not deliver tangible, shippable, valuable functionality**
- For example, the objective of a **spike might be to successfully reach a decision on a course of action**. The **spike is over when the time is up, not necessarily when the objective has been delivered**
- 

How to integrate a Spike with Scrum?

- Spike is a timebox for the Dev Team **to get enough knowledge for estimating a story**. The value for the product owner is, that he can then do his release planning, which would otherwise not be possible. **During a Spike, the Dev Team can create a functional prototype, even neglecting the DoD - but it has to throw it away afterwards. It's just for learning, not for creating product value**

Types of Spike :

1. **Functional Spike:** are used **when there is uncertainty how a user might interact with** the system. They are evaluated through prototyping, by using wireframes, mockups etc.
2. **Technical Spike:** *are used to research various technical approaches in the solution domain*. For e.g. to evaluate the potential performance or load impact of a user story.

39. **Technical Debt** includes those internal things that you choose not to do now, but which will impede future development if left undone. This includes deferred refactoring

- **Technical Debt** is the difference between what was promised and what was actually delivered
- Technical debt also comes in the form of **poorly constructed, inflexible software**. This may come about when functions or interfaces are hard-coded, and as such, are difficult to change
- QPMFUS

- Identified defects are not Technical Debt. They are *Quality Debt*.
- Lack of process or poor process is not Technical Debt. It is *Process Debt*. An example is Configuration Management Debt.
- Wrong or delayed features are not Technical Debt. They are *Feature Debt*.
- Inconsistent or poor user experience is not Technical Debt. It is *User Experience Debt*.
- Lack of skills is not Technical Debt. It is *Skill Debt*.
- Technical debt refers **top accumulated costs and long term consequences of using poor coding techniques**

so much work goes into keeping a production system running (i.e., "servicing the debt") that there is little time left over to add new capabilities to the system—in a legacy system

Technical debt **increases the cost of code quality and the impacts of architectural issues. For IT to help drive business innovation, managing technical debt is a necessity** as it may lead to **unmanageable debt leading to Legacy systems can constrain growth** because they may not scale;

New systems can incur technical debt even before they launch. Organizations should purposely reverse their debt to better support innovation and growth—and revamp their IT delivery models to minimize new debt creation.

#### 40. MINIMUM VIABLE PRODUCT (MVP):

A minimum viable product (MVP) **is a development technique in which a new product or website is developed with sufficient features to satisfy early adopters**. The final, complete set of features is only **designed and developed after considering feedback from the product's initial users**.

A minimum viable product (MVP) is the most **pared down version of a product** that can still be released. An MVP has three key characteristics:

- It has **enough value** that people are willing to use it or buy it initially
- It **demonstrates enough future benefit** to retain early adopters
- It **provides a feedback loop** to guide future development

#### 41. ESTIMATION CHARTS:

**Relative estimation is** one of the several distinct flavors of [estimation](#) used in Agile teams, and consists **of estimating tasks or user stories, not separately and in absolute units of time**, but by comparison or by grouping of **items of equivalent difficulty**.



relative estimation, consistent with estimation in [units other than time](#), avoids some of the pitfalls associated with estimating in general: seeking unwarranted precision, confusing estimates for commitments

## **complexity and size**

## **absolute estimate**

As soon as you ask any software developer about the **effort it takes to implement a particular requirement**, the answer would be either in hours or days. This is called **absolute estimate**

## **Ideal time**

## **42. SCRUM FOR DATAWAREHOUSING PROJECTS:**

Agile approach to data warehousing **solves many of the thorny problems** typically associated with **data warehouse development**—most notably **high costs, low user adoption, ever-changing business requirements and the inability to rapidly adapt as business conditions change**. The Agile approach can be used to develop any analytical database

- Highly desirable to be done in an evolutionary manner
- Need to be flexible
- Collaboration is crucial
  
- Design Data Models using simple language like "Who does What"?
- Ask business users for specific examples so you know how to build the data model
- Use test data from the users, and conduct ETL on the data
- High Level Modeling performed early in the lifecycle
- Light weight modeling throughout the lifecycle
- Comprehensive regression test suites are developed
- Solution is developed in thin vertical slices
- Requirements changes are welcome
  
- User Stories needs to drive Scrum development for DW, not data
- Get to know what data and how do users use the data

- How are we going to use the information?
- Scrum DW teams must deliver new data or reports each iteration
- Agile DW teams strive to fix the data at the source itself via data refactoring
- Agile DW is Pragmatic
  - Sometimes, you need to accept data inconsistencies
  - You cannot get all data you need for a report

### **Agile Database Technique Stack (Important):**

- Vertical Slicing
  - Slicing Strategies:
    - One new data element from a single data source
    - One new data element from several sources
    - Change to existing report
    - New report
    - New reporting view
    - New Data Mart Table
- Clean Architecture and Design
  - Cleaner the design of the database, easier it is to evolve
- Agile Data Modeling
  - Gather details, make decisions in an iterative way
- Database Refactoring
  - Simple change to database schema that improves its design while retaining both its behavioral and informational semantics
- Database Regression Testing
  - Test Driven Database Development
  - Behavior Driven Database Development
- Continuous Database Integration
- Configuration Management

### **REFACTORING:**

Refactoring consists of improving the internal structure of an existing program's source code, while preserving its external behavior **on strengthening security, the flow of inputs and outputs** through the application, and improving **design and usability**

## Scrum Release Planning

A very high-level plan for multiple Sprints (e.g. three to twelve iteration) is created during the Release planning. It is a guideline that reflects expectations about which features will be implemented and when they are completed. It also serves as a base to monitor progress within the project. Releases can be intermediate deliveries done during the project or the final delivery at the end.

To create a Release Plan the following things have to be available:

- A prioritized and estimated Scrum Product Backlog
- The (estimated) velocity of the Scrum Team
- Conditions of satisfaction (goals for the schedule, scope, resources)

Quiz



Scrum demands increasing rigor in the definition of "done."  
Skilled use of TDD (Test Driven Development), refactoring, and continuous integration help keep the software product in a constantly known state. This means regression tests are automated and building/testing occurs much more frequently than once per day.


*"Daily builds are for wimps." - Kent Beck*

Continue

Scrum Training Series | Introduction to Scrum © 2011 - 2012 CollabNet. All rights reserved. v32 CollabNet

Tweet Like Share 1.3K people like this. Sign Up to see what your friends like. +710 Recommend this on Google

Module 1: Introduction to Scrum



**Objective Questions** *(What happened?)*  
**Reflective Questions** *(How do we feel about it?)*  
**Interpretive Questions** *(What does it mean?)*  
**Decision Questions** *(What are we going to do about it?)*

a.k.a. *The Focused Conversation Model* by Brian Stanfield.\*

\* Brian Stanfield, *The Art of Focused Conversation* (1999)

Scrum Training Series | Sprint Retrospective Meeting © 2011 - 2012 CollabNet. All rights reserved. v49 CollabNet

Tweet Like Share 132 people like this. Sign Up to see what your friends like. +104 Recommend this on Google

Module 6: Sprint Retrospective Meeting