VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Module – 3

Agile Requirements

Dr. Rajesh M
School of Computer Science and Engineering
Vellore Institute of Technology [VIT]
Chennai, India.

Module	Topics	L Hrs
3	AGILE REQUIREMENTS : Meeting the requirements challenge iteratively- Requirements for Agile approach – Gathering & analysis –Behavior Driven Development (BDD) and Acceptance Test Driven Development (ATDD)- Designing storyboards and scrums in Agile approach.	6

Requirements for Agile approach

Requirements for Agile approach

- The ***main objective of Agile RE(Requirement Engineering)*** is **to** ***accommodate changing requirements even late*** in the development lifecycle.

The requirements engineering phase of the SDLC comprises of the following activities:

- **Requirements Elicitation** – Discover, understand, articulate and record customer and user needs.
- **Requirements Analysis** – Understand needs/constraints of the user and derive the set of requirements

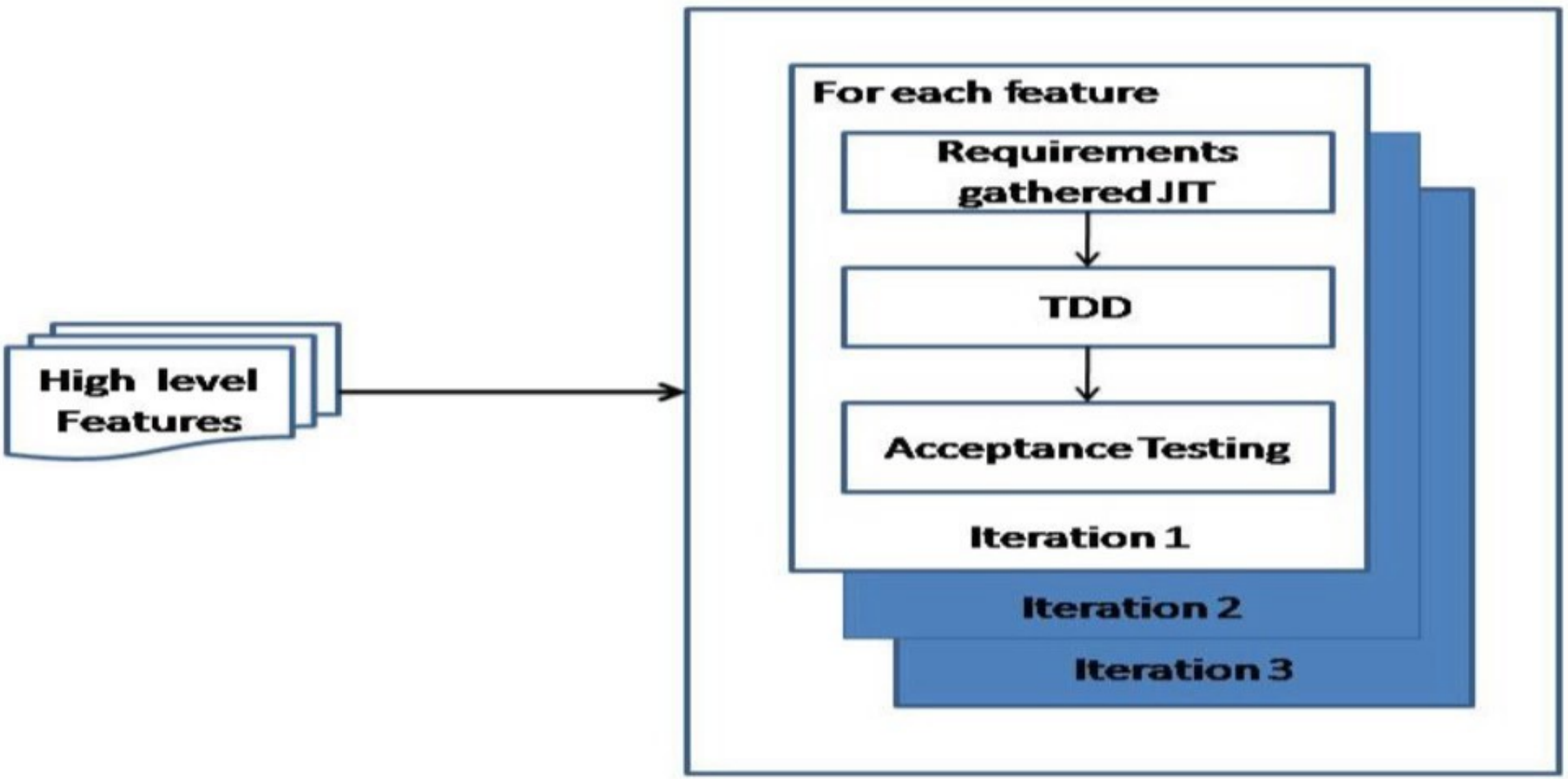
Agile RE

- **Requirements Verification/Validation** – Ensure that the requirements are adequate and conform to agreed standards

- **Requirements Specification** – Record the requirements in an unambiguous manner

- **Requirements Management** – Plan and Control the above activities

Agile RE


High level Features
For each feature
Requirements gathered JIT
TDD
Acceptance Testing
Iteration 1
Iteration 2
Iteration 3

Advantages and Disadvantages of Agile RE

Advantages:

- **Accommodates changing requirements even late** in the development cycle.
- **Mitigates issues faced** when gathering requirements upfront.

Disadvantages:

- **No real process/ specification of activities** in the RE phase
- Non-functional requirements are not well-defined.
- The **focus is on identifying all the requirements** and not on classifying the requirements into functional and non-functional requirements.
- Also, **non-functional requirements are recorded as user stories**.

Traditional RE Vs Agile RE

	Agile RE	Conventional RE
1.	Evolutionary requirements. No BRUF. Requirements are gathered just-in-time.	Requirements do not evolve over time. All the requirements are gathered up front before the design phase.
2.	Minimal requirements documentation. Requirements are stored on index cards.	Formal Software Requirements Specification (SRS) document is produced.
3.	Implicit verification and validation of requirements. V&V is carried out just-in-time.	Explicit verification and validation of requirements activities.
4.	Changes to requirements even late in the development are accommodated.	Changes in requirements during the later phases of the development life cycle are not accommodated.
5.	The time for project completion is decomposed into multiple release cycles and the features are estimated to fit within these release cycles.	Requirements are gathered and specified before estimating the time required to develop these requirements.
6.	On-site customer is present to answer questions from the development team	No on-site customers.
7.	Requirements are recorded on index cards in the domain language of the user.	Requirements are specified in a formal language

Requirements Gathering - issues

Requirements Gathering - issues

Some of the issues faced by organizations involved in up front requirements gathering and specification efforts are:

- **Requirements change over a period of time due to changes in customer and user needs**, technological advancement and schedule constraints.

- Developers and requirements engineers often **misinterpret or fail to capture customer and user needs** and intents and thereby, specify incorrect requirements.

Requirements Gathering - issues

- **Customers and users are not very clear about their needs** and often state the desired functionality in terms that can mislead or result in incorrect inferences by the requirements engineer.

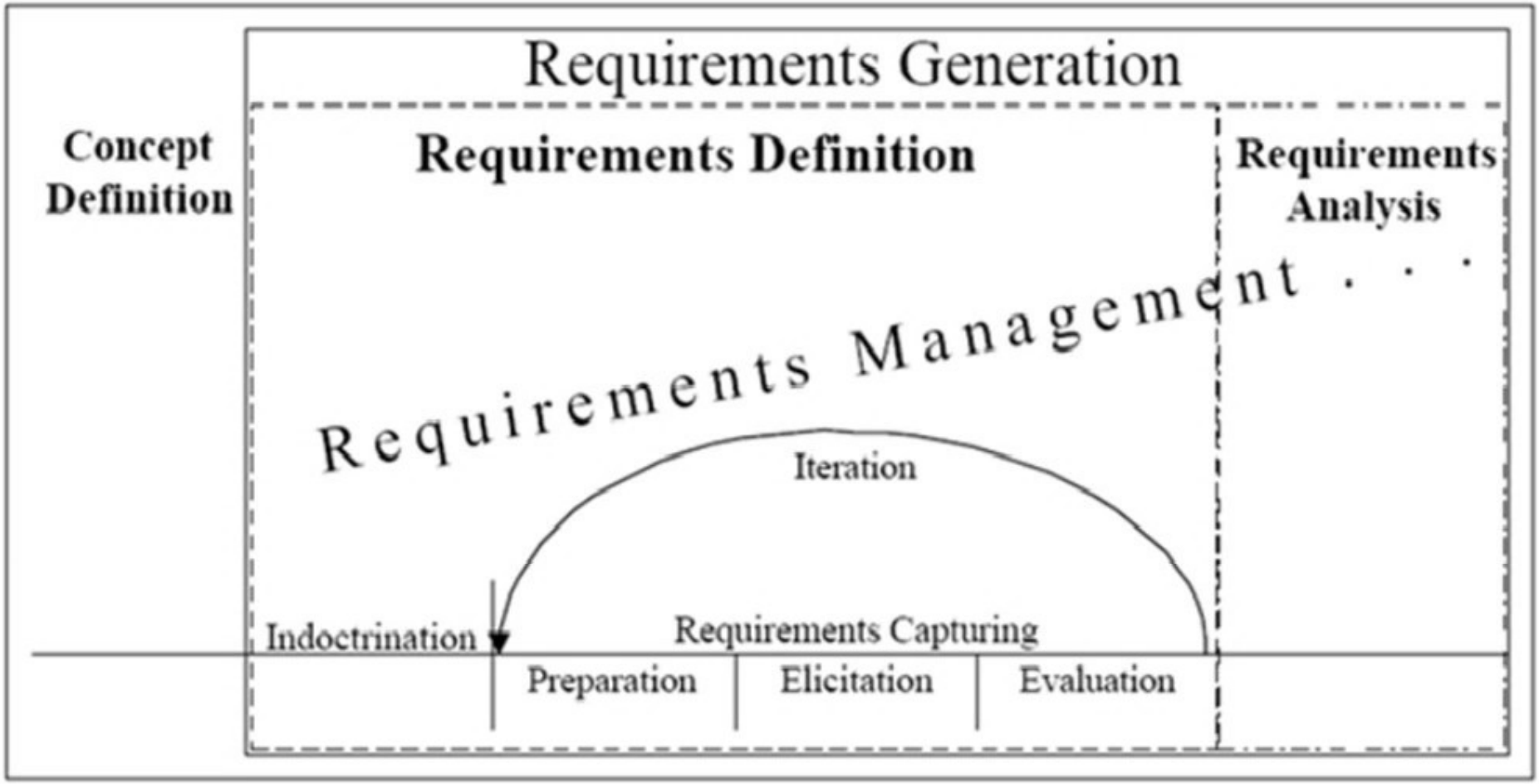
- **Rapidly changing business environments** can cause requirements to become obsolete before project completion.

Agile RE process activities

The major Agile RE process activities can be summarized as below:

1. ***Identify the high level features*** of the system. The features denote the functionalities expected by the customers.
2. ***Prioritize the features*** based on the customer needs and their business value.
3. For each feature to be developed, ***gather the details JIT from the customers and users***.
4. With the gathered details for the features, ***proceed to development and acceptance testing.***

Agile Requirement – Report



©2002, Markus K. Gröner

Agile Methods: BDD

Behavior Driven Development (BDD)

- **Behavior Driven Development (BDD)** is an agile software development practice – introduced by Dan North in 2006 – that **encourages collaboration between everyone involved in developing software**: developers, testers, and business representatives such as product owners or business analysts.

- BDD **aims to create a shared understanding of how an application should behave by discovering new features** based on concrete examples. Key examples are then formalized with natural language following a Given/When/Then structure.

Behavior Driven Development (BDD)

- Behavior Driven Development (BDD) is **similar to the Test Driven Development** (TDD), and the **focus is on testing the code** to ensure the expected behavior of the system.

- In BDD, language like English is used so that it makes sense to the users, testers and developers. It ensures –
  - **Continuous communication** among the users, testers and developers.
  - **Transparency** on **what is being developed and tested**.

why use BDD?

- Here are some of the top advantages that teams practicing BDD experience:

- **Reduced Rework / Shared Understanding:** Concrete examples of expected system behavior foster a shared understanding by being specific enough for developers and testers while still making sense to business participants. Think of a BDD scenario like a bug report that is written in natural-language and posted before the system has been implemented. It describes the steps to reproduce (Given/When) and expected outcome (Then) without the symptoms (unwanted behavior) that usually triggered the bug report.

why use BDD?

- **Faster Feedback:** Example scenarios describe focused, granular changes to the system under development. This enables teams to evolve the system in small steps while keeping it **potentially shippable**, allowing for shorter release cycles and faster feedback.

- **Effectiveness:** Since you extend the system in small increments, you can **validate business value earlier and avoid unnecessary features**. This prevents gold-plating and makes the overall implementation of the system more effective.

why use BDD?

- **Lower Cost:** Driving automated acceptance tests through test-first BDD scenarios is much cheaper than post-automating acceptance tests.
- Teams practicing ATDD (Acceptance Test Driven Development) use their shared understanding to develop the feature and the test automation, while teams separating development and test automation need to interpret and fine-tune scenarios multiple times. This causes extra effort and can lead to misaligned interpretations.
- Additionally, test automation based on scenarios that were used to drive the implementation have a better focus and more relevant assertions. This leads to higher test automation coverage and more relevant test automation scenarios, which reduces regression errors and efforts for manual checks.

why use BDD?

- **Single Source of Truth:** Specification through examples that is guarded with automated acceptance tests is an always-up-to-date description of the current system behavior ("Living Documentation"). This provides a single source of truth for the team, business stakeholders and important external parties like regulatory authorities or partners relying on the system specification.

- **User Satisfaction:** By focusing on the needs of the business, you get satisfied users – which translates to customer loyalty and better business outcomes. The higher degree of test automation frees more time for manual exploratory testing and yields less errors in production, especially when shipping new versions at a high cadence. More frequent, high quality releases enable rapid response to evolving user needs and dynamic market pressuress.

why use BDD?

- **Code Quality:** Using Acceptance Test Driven Development has a positive impact on code quality: it promotes emergent design that ensures loosely-coupled, highly-cohesive architecture and avoids over-engineering and technical debt.

- This ensures that a system stays testable and maintainable– and that it can be quickly changed to support new requirements without sacrificing stability.

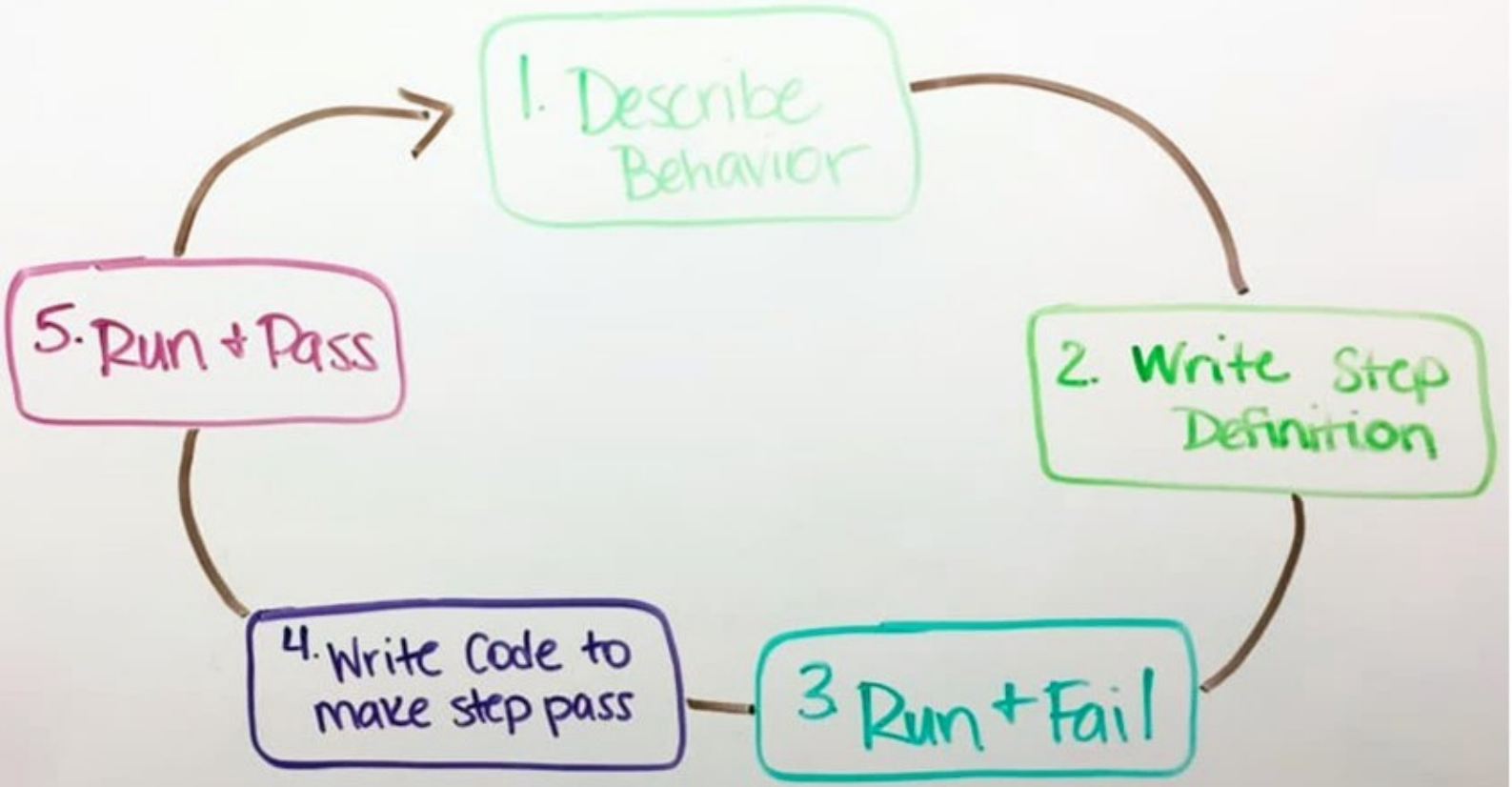
Difference between BDD and TDD?

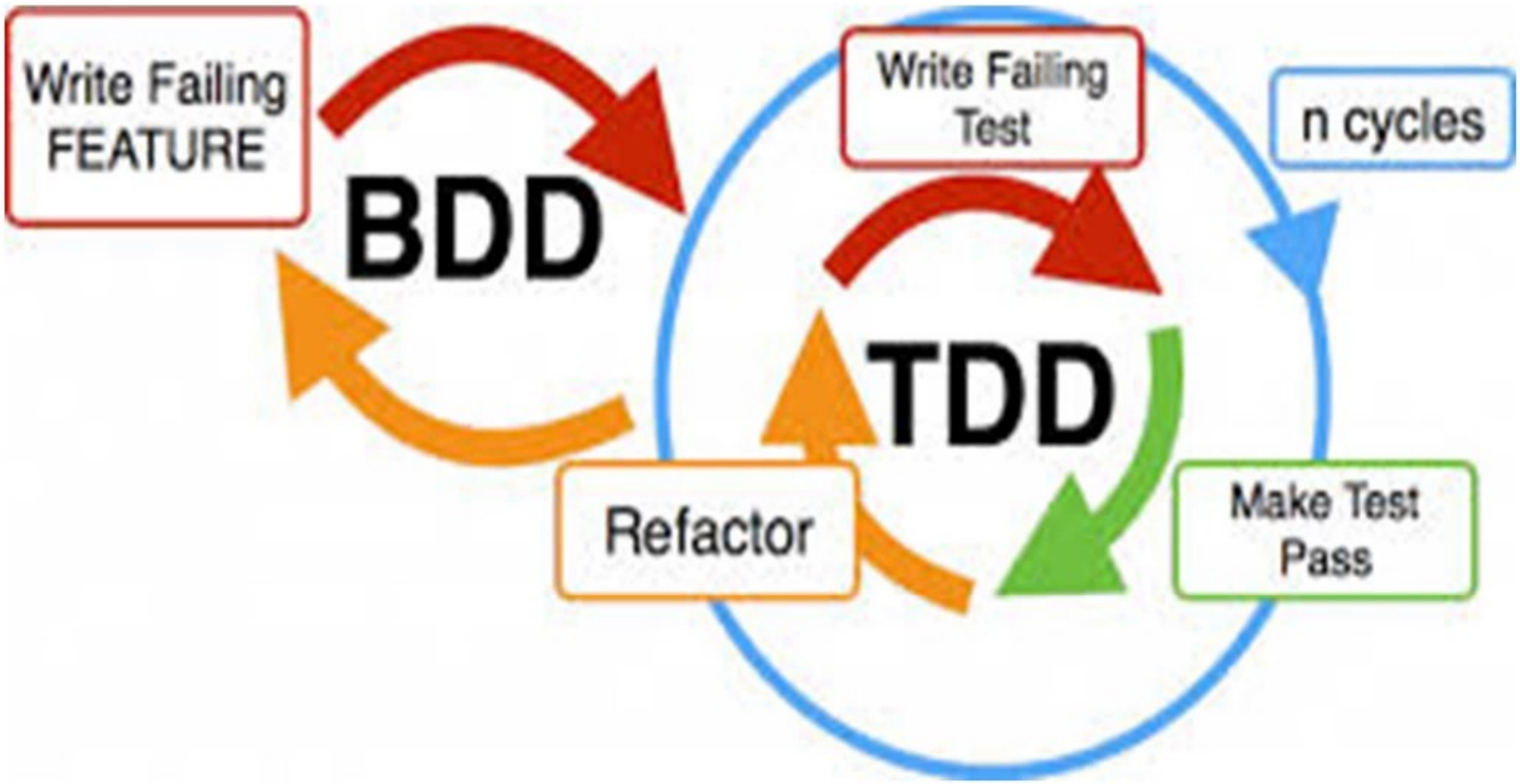
- In TDD (Test Driven Development), the test is written to check the implementation of functionality, but as the code evolves, tests can give false results.

- BDD (Behavior Driven Development) is also a test-first approach, but differs by testing the actual behavior of the system from the end users perspective

Testing Methods

Behavior Driven Development (BDD)




Write Failing FEATURE
BDD
Write Failing Test
n cycles
TDD
Refactor
Make Test Pass

Agile Methods: ATDD

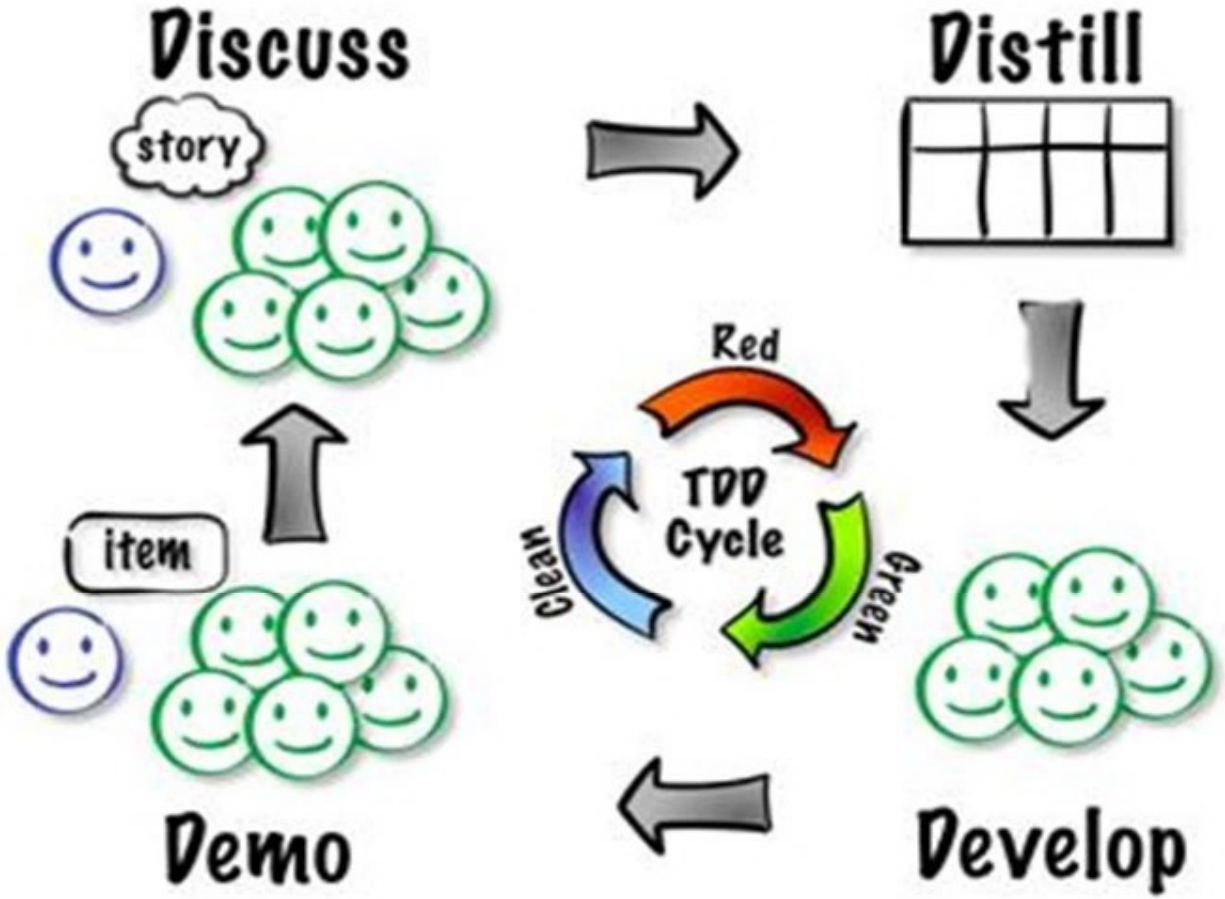
Acceptance Test Driven Development

- In the **Acceptance Test Driven Development (ATDD)** method, the code is developed based on the test-first approach directed by Acceptance Test Cases.

- The focus is on the acceptance criteria and the **Acceptance Test Cases written by the testers during User Story Creation** in collaboration with the customer, end users and relevant stakeholders.

ATDD Cycle

Acceptance Test Driven Development (ATDD) Cycle



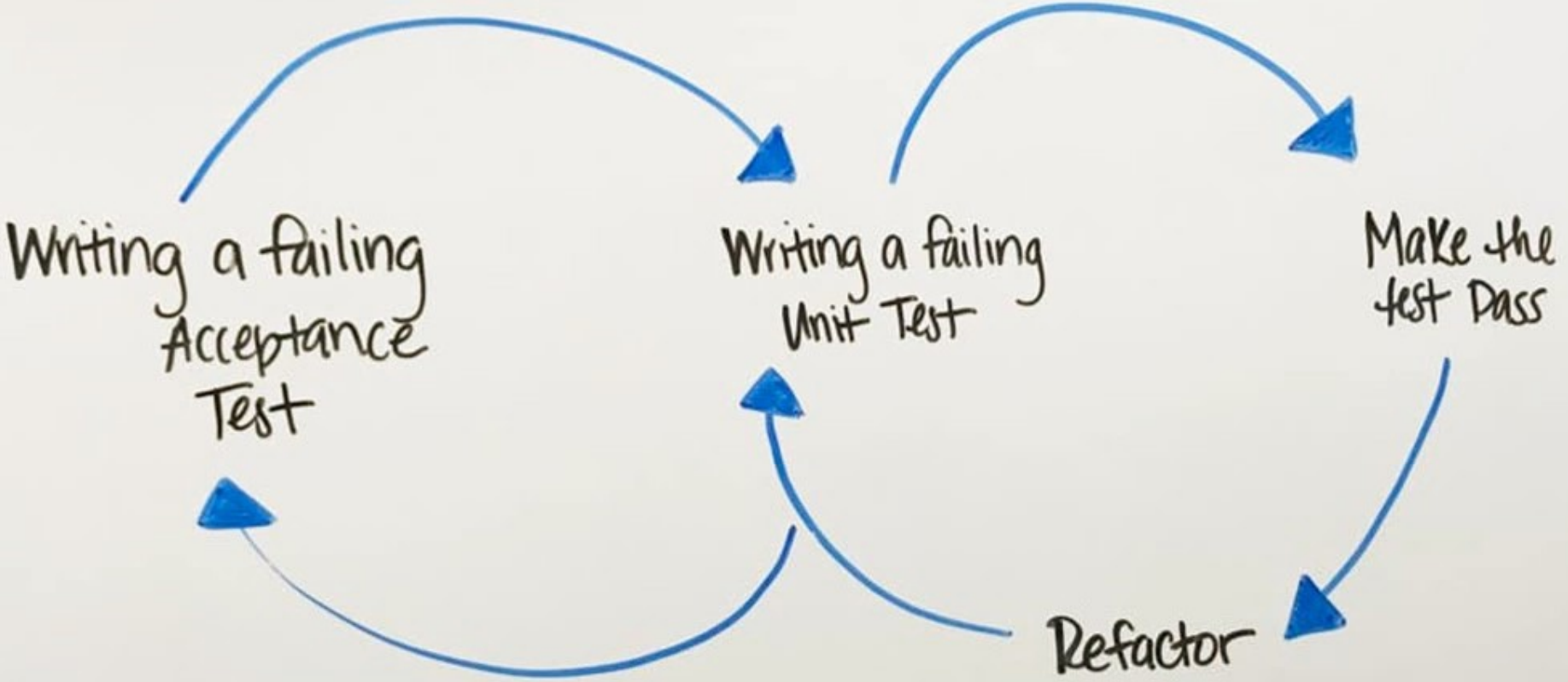
ATDD cycle model developed by James Shore with changes suggested by Grigori Melnick, Brian Marick, and Elisabeth Hendrickson.

ATDD Process

- **Step 1 – Write Acceptance Test Cases** along with user stories in collaboration with the customer and users.
- **Step 2 – Define** the **associated acceptance criteria.**
- **Step 3 – Develop code** based on the acceptance tests and acceptance criteria.
- **Step 4 – Run the acceptance tests** to ensure that the code is running as expected.
- **Step 5 – Automate** the **acceptance tests**. Repeat **Step 3 – Step 5** until all the user stories in the iteration are implemented.
- **Step 6 – Automate the regression tests**.
- **Step 7 – Run the automated Regression Tests** to *ensure Continuous Regression.*

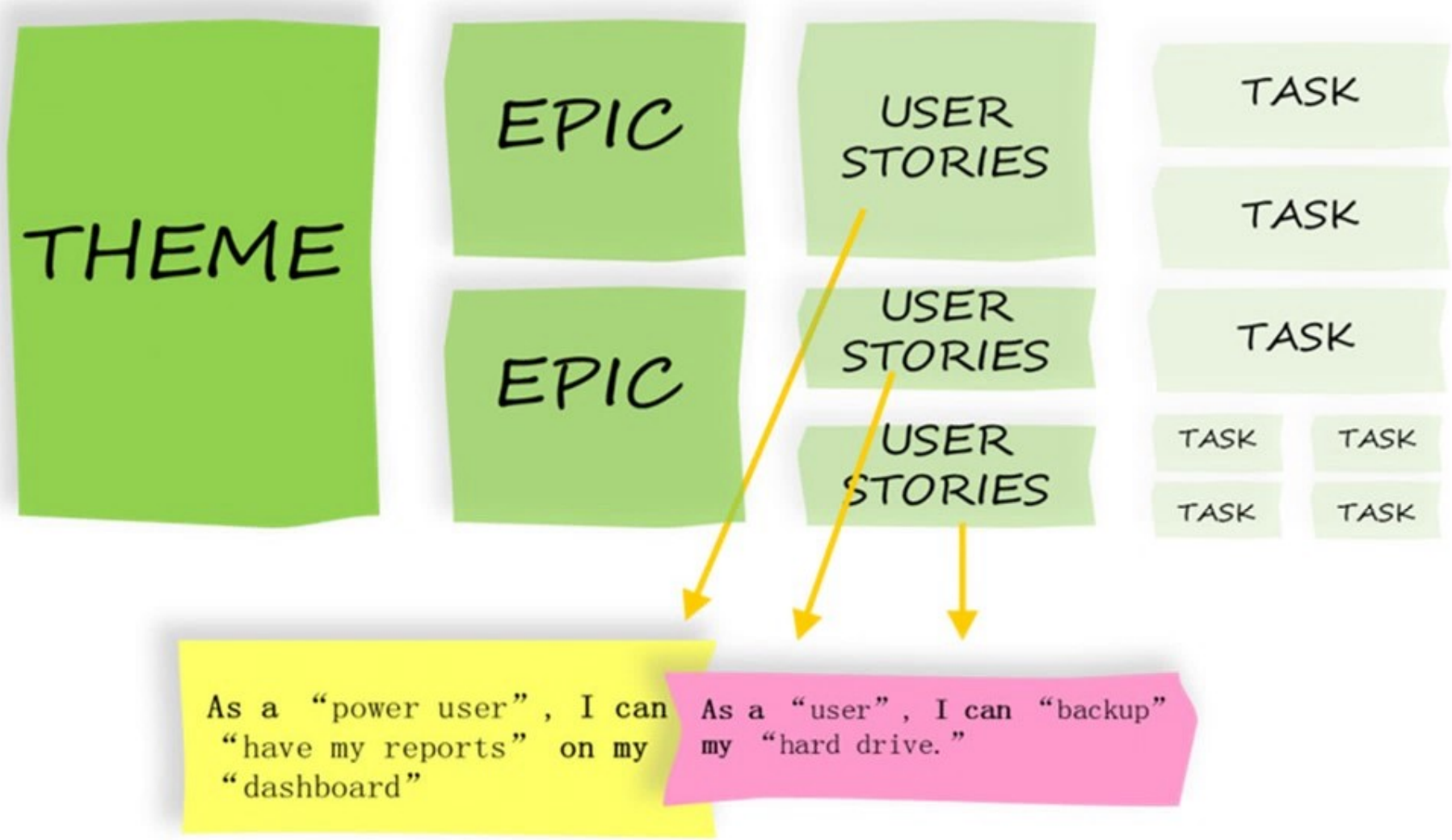
Testing Methods

Acceptance Test Driven Development (ATDD)

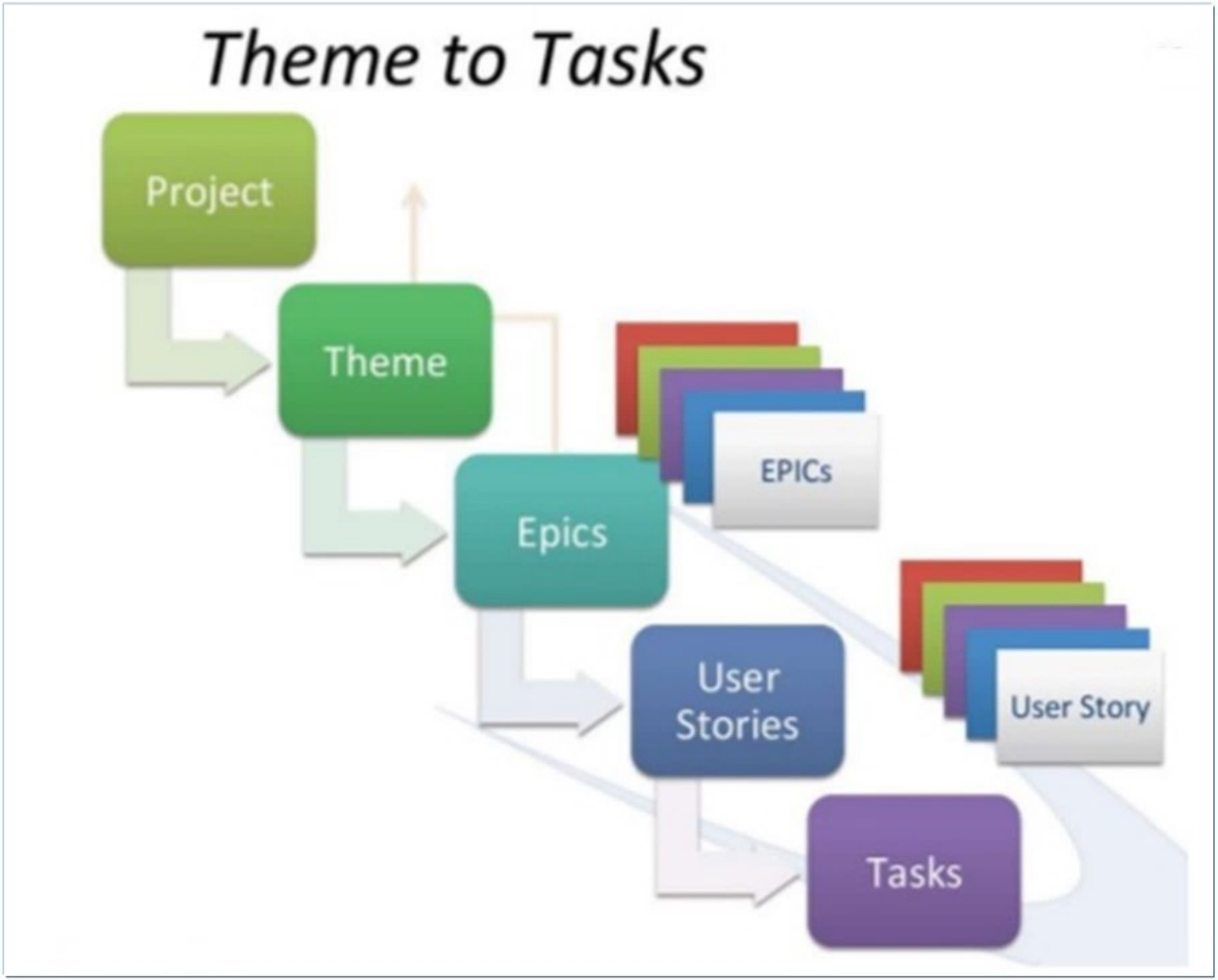


Designing story-boards in Agile approach

USER STORIES



Theme to Tasks



User Stories in Agile approach

- **Theme/ Initiatives / user Feature:**

- A theme provides a convenient way to indicate that a set of related epics have something in common, such as being in the same functional area.

- By assigning a financial value to Themes, managers can ensure the highest value is being delivered and that the project/program is aligned with its objectives and the strategic direction of the organization.

User Stories in Agile approach

- **Theme/ Initiatives / user Feature:**

- In many organizations the founders and management team will encourage the pursuit of some aspirational destination. These are the (sometimes super corny) goals announced each year or quarter, and themes are how you keep track of them.

- Initiatives are collections of epics
- Themes are labels that track high-level organizational goals

- Initiatives have a structural design. They house epics, and the completion of those epics will lead to the completion of the initiative. Themes are an organizational tool that allows you to label backlog items, epics, and initiatives to understand what work contributes to what organizational goals.

User Stories in Agile approach

- **Epic :**

- An Epic is useful as placeholders for large requirements.
- It probably won't fit into a sprint and should be broken down into stories.
- Epics are usually defined during the initial product roadmap and decomposed into stories in the product backlog as more is learned and is usually written in a User Story format.
- The decomposed stories in an epic have a common objective and a specific outcome or high-level user need or part of the journey or process someone takes in using the service.

User Stories in Agile approach

- **User Stories :**

- User stories are the smallest units of user functionality in agile which can be delivered in one agile sprint.
- They are typically estimated using story pointed and defined using INVEST criteria.
- User stories should deliver a vertical slice of functionality to the customer that is valuable and complete by the end of an iteration.
- A user story must deliver particular value to the user and must be describable in simple language that outlines the desired outcome.

User Stories in Agile approach

- **Examples of an agile story:**

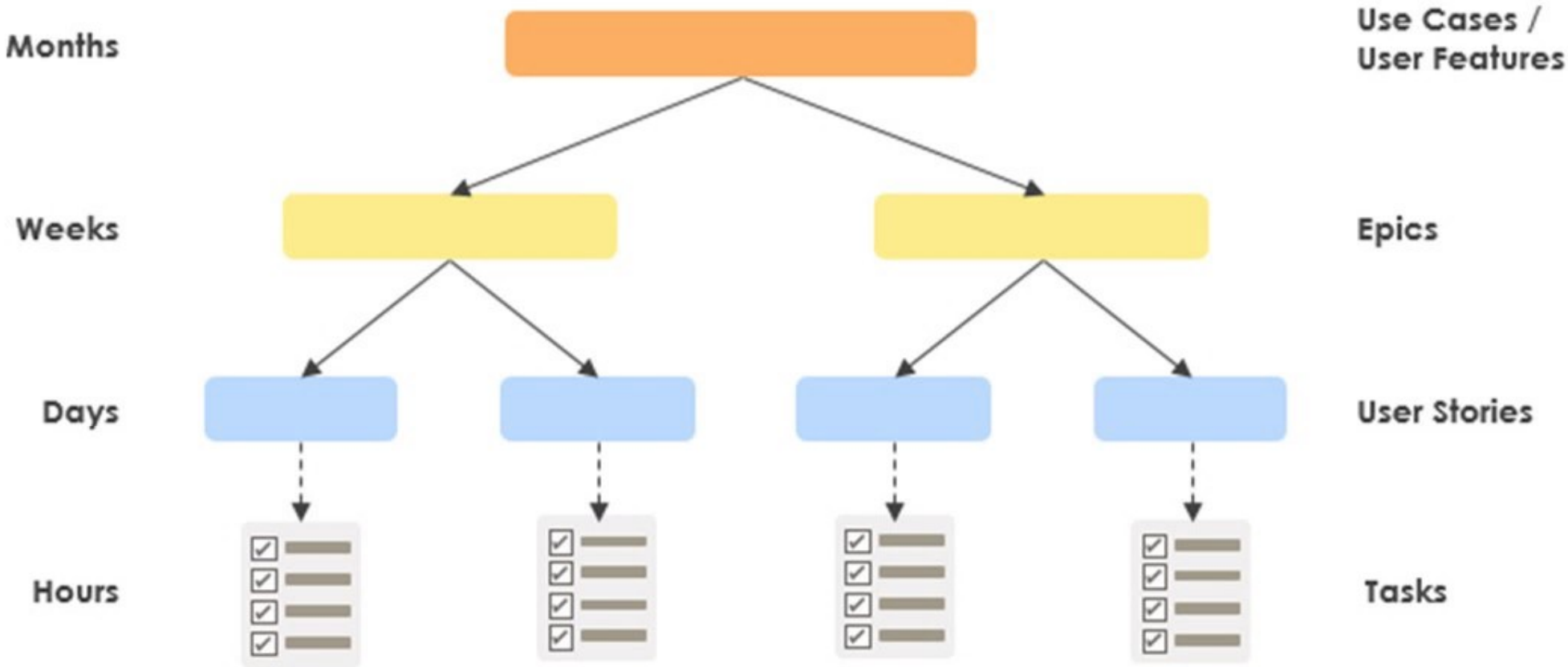
- iPhone users need access to a vertical view of the live feed when using the mobile app.
- Desktop users need a "view fullscreen" button in the lower right hand corner of the video player.
- Android users need to be linked to apple store.

User Stories in Agile approach

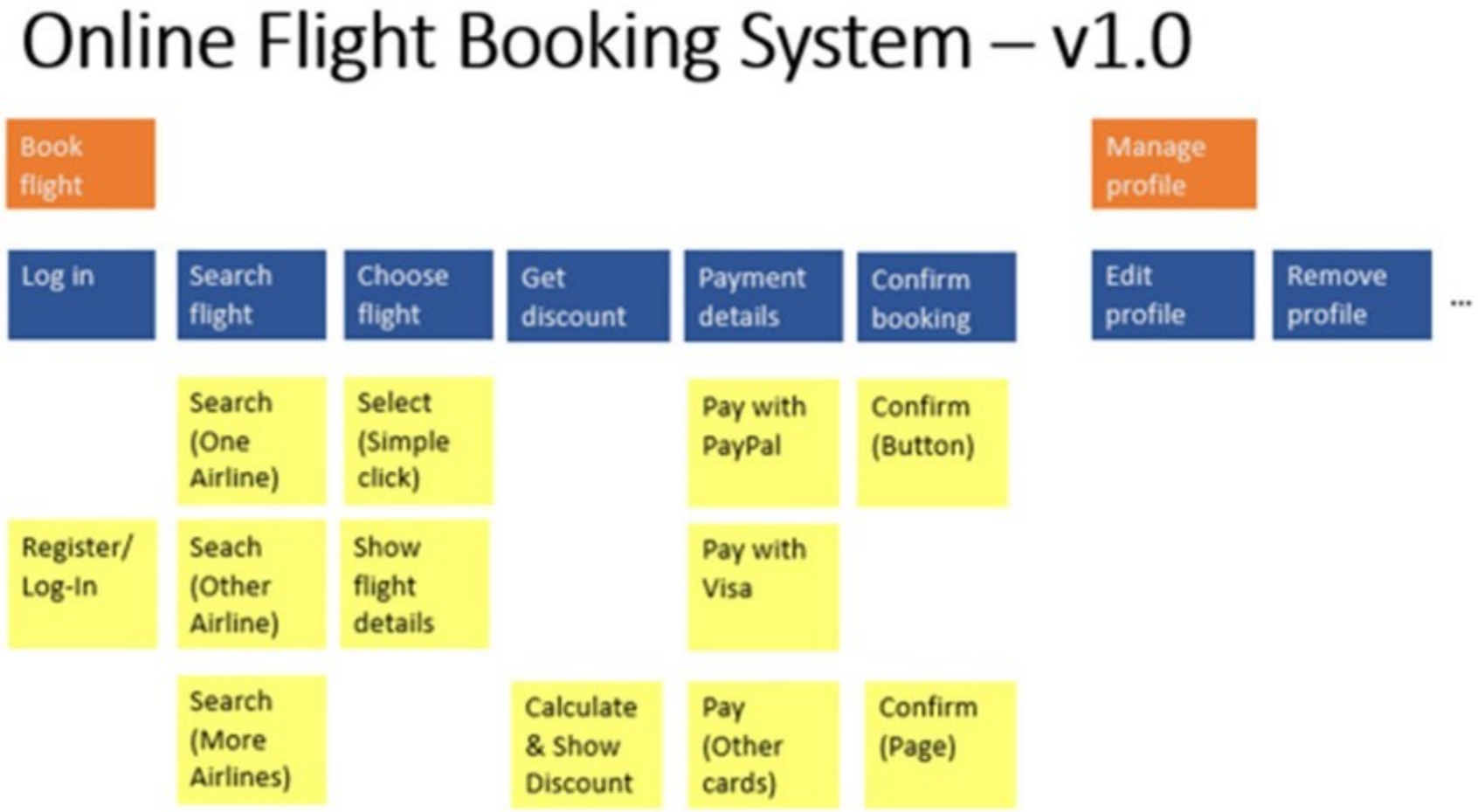
- **Tasks :**

- Tasks are decomposed parts of a story that get into HOW the story will be completed. Tasks can be hour estimated if desired.
- Tasks are usually defined by the people doing the work (developers, QA, etc), whereas stories and epics are generally created by the customer or the product owner on behalf of the customer.
- Thus, the tasks no longer need to be understandable by business users and so can be highly technical.
- The process of breaking a story down into tasks also helps the development team better understand what needs to be done.

User Stories in Agile approach



User Stories in Agile approach



Designing story-boards in Agile approach

- The features chosen for development during the current release cycle serve as the input to the Story Development Phase.
- Each feature is decomposed into stories. *"Stories are descriptions of user- or customer-valued functionality".*
- Stories are defined at a lower level of abstraction when compared to the features.

Designing story-boards in Agile approach

- *Each identified feature* for a current release is *decomposed into stories*.
- If multiple teams are involved, then each team can work on identifying stories for a particular feature.
- *Each story is a sentence or two or a paragraph* at most. Stories are recorded on index cards.

Designing story-boards in Agile approach

- The Story Development Phase and its activities are shown in the following Figure.
- Each feature to be implemented during the current release cycle is decomposed into stories.
- *The development team elicits stories* from the customers and users during pre-determined meetings.
- The elicited stories are *validated by the customers*.
- The developers estimate the time required for each story to be completed.

Designing story-boards in Agile approach

Examples for stories:

Consider the online payment feature mentioned in table 1. The following are two stories which can be created for this feature:

1. **A user can pay by credit card**
2. **A user can use an e-check to make a payment**

For the search catalog feature, the following story can be created:

1. **A user can search for a product on various fields**

Designing story-boards in Agile approach

- The *validated stories are then prioritized* by the developers.
- The *prioritization is based on dependencies* among the stories.
- These prioritized stories are *stored in a prioritized story stack.*
- This prioritized story stack can take the form of a *bunch of index cards or electronic cards* with stories and their priorities recorded on them.
- The stories are elicited over a number of iterations.

Story Development Phase - Objectives

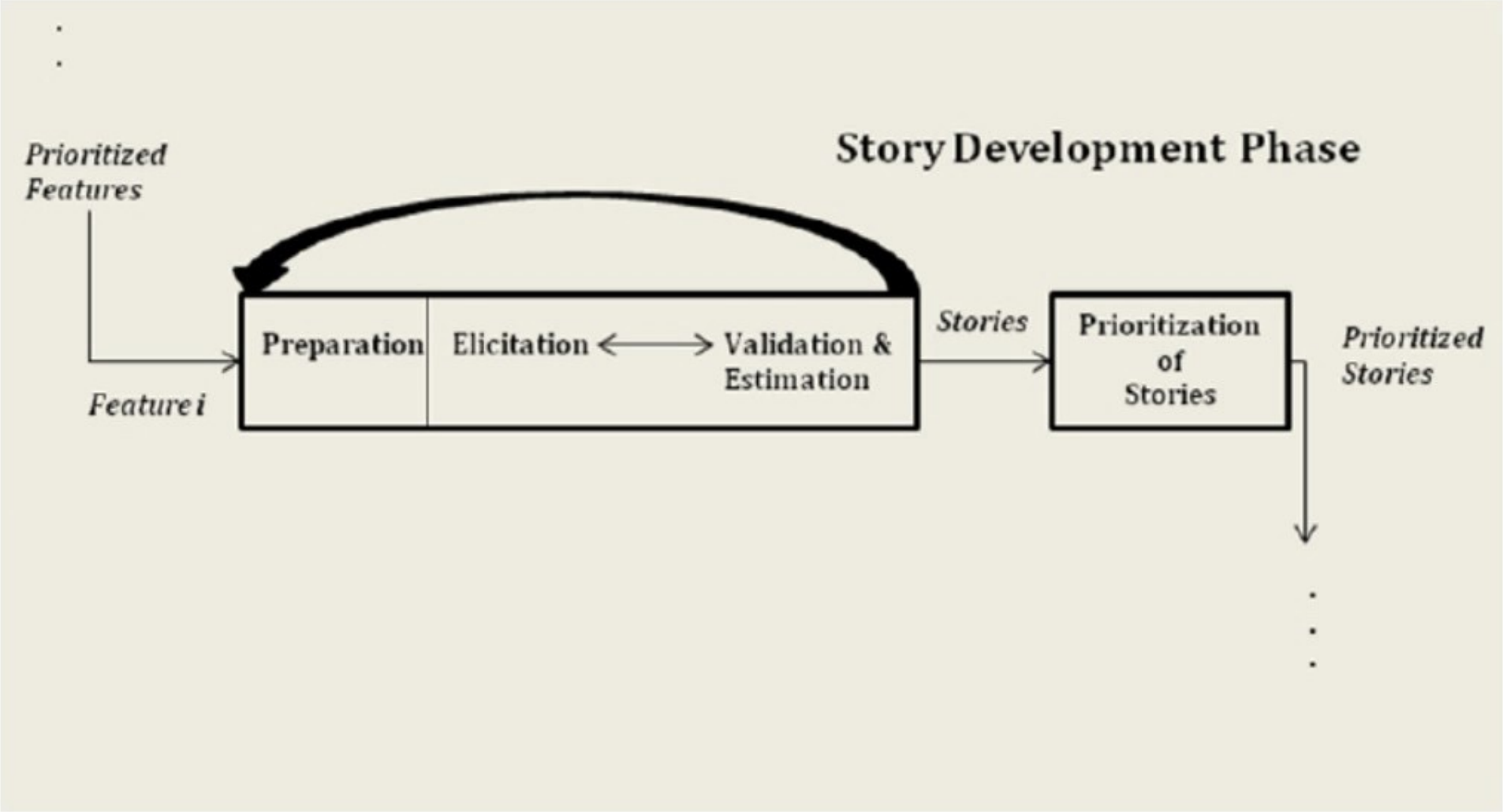
Story Development Phase - Objectives

The objectives of the Story Development Phase are:

- **Create user stories** – The *stakeholders meet to create stories for each feature* to be implemented during the current release cycle.
- **The stories are elicited** over a number of iterations.
- **Validate the stories and estimate the time** for completing a story – The *stakeholders validate the identified stories* over the many iterations and the developers estimate the time required to complete the stories.
- **Prioritize the stories** – The *developers prioritize the stories* and *store them in a prioritized story stack*.

Story Development Phases

Story Development Phases



Story Development Phases

The various activities of this phase are:

1. Preparation – This objective of this activity is to *arrange for a meeting among the stakeholders is to create stories* for a subset of prioritized features.

2. Elicitation – The *chosen feature is decomposed into stories*. This process takes into account the agile practice of no BRUF and the details are gathered on a just-in-time basis.

- Stories are created for all the user classes identified during the education phase.
- Common techniques for eliciting stories include interviews, observing users interact with *software, questionnaires and story-writing workshops*.
- A story writing workshop is a brainstorming session involving all the stakeholders dedicated to creating stories.
- Customers write the acceptance criteria for each story.
- Agile methods focus on delivering functionality of maximum business value to the customers.
- Customer and user preferences take precedence. Hence, the customers write the acceptance criteria and later validate the developed code against these criteria.

Story Development Phases

3. Validation and Estimation – The stories created are validated and the developers estimate the time required for completing each story.

- Validation involves discussing the stories with the customers, creating screen designs and paper prototypes.
- If there are changes to the identified stories or if need for new stories is recognized, another iteration of the story development phase is carried out.
- As mentioned earlier, the product development cycle spans multiple releases and each release is divided into iterations. The time for each story is usually estimated in story points which are relative estimates of complexity, effort or duration of a story.
- Each agile team has its own definition for story points. Story points can be defined in terms of number of hours, days or weeks of work.

Story Development Phases

4. Prioritization – The developers determine the dependencies if any among the stories and prioritize the stories accordingly.

- They also consider the customer and user preferences
- Prioritization techniques discussed in the previous section can be used.
- Developers and customers may have different sequences in which they would like to implement the stories.
- If there is a conflict, the customer is given preference.
- The customer is made aware of issues that may arise when their choices are given precedence.
- The output of this phase is a prioritized story stack. The stories are identified over a series of meetings.
- However, after a story or an initial set of stories are identified, tasks can be created and developed.

References

- K.S. Rubin, Essential Scrum: A Practical Guide to the Most Popular Agile Process, Addison-Wesley, 2012.
- M. Cohn, Succeeding with Agile: Software Development Using Scrum, Addison-Wesley, 2009
- S.W. Ambler, M. Lines, Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise, IBM Press, 2012.
- Chetankumar Patel, Muthu Ramachandran, Story Card Maturity Model (SMM): A Process Improvement Framework for Agile Requirements Engineering Practices, Journal of Software,Academy Publishers, Vol 4, No 5 (2009), 422-435, Jul 2009.
- Kevin C. Desouza, Agile information systems: conceptualization, construction, and management, Butterworth-Heinemann, 2007
- K. Beck, C. Andres, Extreme Programming Explained: Embrace Change, 2nd Edition, Addison-Wesley, 2004.

Thank You