Agile Software Development
CSE 8045

Unit – I

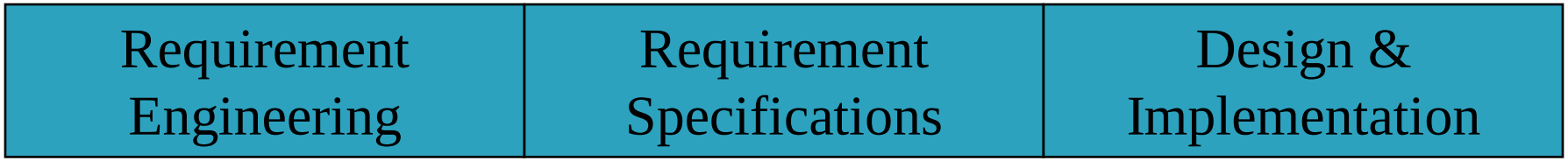
Fundamentals of Agile

&

Agile Scrum Framework

What are Traditional Methods ?

Traditional Methods Vs Agile Methods



Traditional Methods

Agile Methods

History of Agile Methods

- Particularly in 1990s, some developers reacted against traditional "heavyweight" software development processes.

- New methods were being developed and tested,
  - e.g. extreme programming, SCRUM, Feature-driven development
  - Generally termed "light" processes

- "Representatives" from several of these methods got together in Utah in 2001
  - Settled on term "Agile" as a way to describe these methods
  - Called themselves the "Agile Alliance"
  - Developed a "manifesto" and a statement of "principles"
  - Focuses on common themes in these alternative methodologies

Manifesto for Agile Software Development

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

Twelve Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity--the art of maximizing the amount of work not done--is essential.

11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Individuals and Interactions over Processes and Tools

- People make biggest impact on success
  - Process and environment help, but will not create success
- Strong individuals not enough without good team interaction.
  - Individuals may be stronger based on their ability to work on a team
- Tools can help, but bigger and better tools can hinder more than help
  - Simpler tools can be better

Working Software over Comprehensive Documentation

- Documentation important, but too much is worse than too little
  - Long time to produce, keep in sync with code
  - Keep documents short and salient
- Focus effort on producing code, not descriptions of it
  - Code should document itself
  - Knowledge of code kept within the team
- Produce no document unless its need is immediate and significant.

Customer Collaboration over Contract Negotiation

- Not reasonable to specify what's needed and then have no more contact until final product delivered
- Get regular customer feedback
- Use contracts to specify customer interaction rather than requirements, schedule, and cost

Responding to Change over Following a Plan

- Environment, requirements, and estimates of work required will change over course of large project.

- Planning out a whole project doesn't hold up
  - Changes in shape, not just in time

- Keep planning realistic
  - Know tasks for next couple of weeks
  - Rough idea of requirements to work on next few months
  - Vague sense of what needs to be done over year

Extreme Programming (XP)

- One of the most well-known agile methods
- Developed in 1990s
  - Kent Beck, 1996
  - Chrysler Comprehensive Compensation Project
  - Published book in 1999

Extreme Programming Practices

1. **On-Site Customer**
   - Customer is actively involved with development process
   - Customer gives “User Stories”
     - Short, informal “stories” describing features
     - Keep on “story cards”

2. **Planning Game**
   - Developers and customers together plan project
   - Developers give cost estimates to “stories” and a budget of how much they can accomplish
     - Can use abstract accounting mechanism
     - Later compare to actual cost, to improve estimates over time
   - Customer prioritizes stories to fit within budget

Extreme Programming Practices

3. **Metaphor**
   - Come up with metaphor that describes how the whole project will fit together
   - The picture in a jigsaw puzzle
   - Provides framework for discussing project in team
   - Tools and materials often provide good metaphors

4. **Small Releases**
   - Time between releases drastically reduced
     - A few weeks/months
   - Multiple iterations
   - Can have intermediate iterations between bigger “releases”

Extreme Programming Practices

5. **Testing**
   - Test-first programming
   - Unit testing frequently by developers
   - Acceptance tests defined by customers

6. **Simple Design**
   - Principles discussed in earlier lectures
   - Design should be quick, not elaborate
   - Pick the simplest thing that could possibly work
   - Resist adding stuff not ready yet

Extreme Programming Practices

7. **Refactoring**
   - Code gets worse with feature adds, bug fixes
   - Rewrite small sections of code *regularly*
   - Rerun all unit tests to know nothing broken
     - Means you should have designed comprehensive tests

8. **Pair Programming**
   - Discussed later

9. **Collective Ownership**
   - Anyone can edit anything
   - Errors are the fault of the whole team

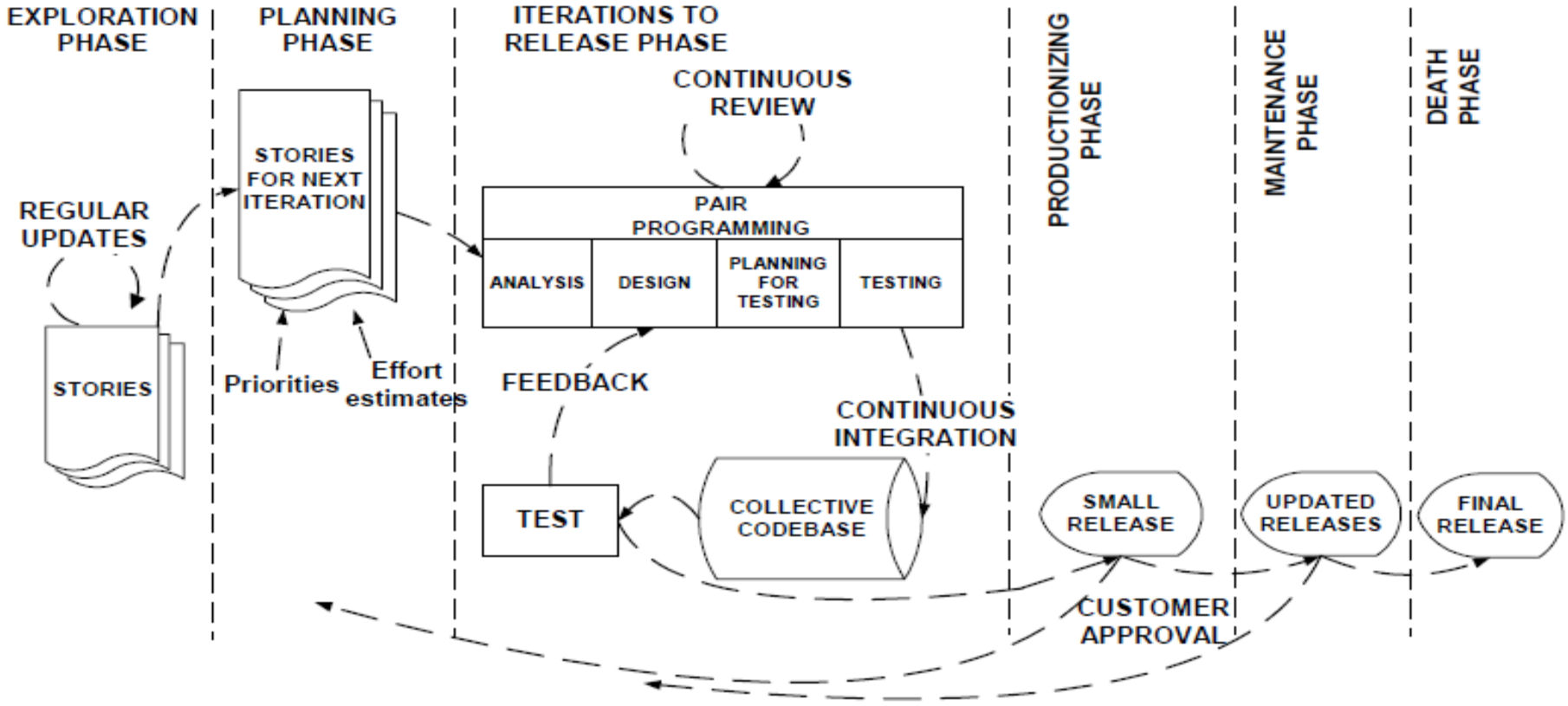
Extreme Programming Practices

10. **Continuous Integration**
    - Commit changes frequently (several times a day)
    - Verify against entire test suite!

11. **Coding Standards**
    - Enables effective teamwork

12. **Sustainable Pace**
    - No overtime
    - Only exceptions in final week
    - Good estimation skills for budgeting will help ensure reasonable times
    - Time less likely to be wasted in pairs, bullpen rooms
    - Plan time each day for administrative work (<1 hour), breaks

Extreme Programming Lifecycle



SCRUM

- Idea first appeared in a business journal in 1986 (applied to product development management).
- Used in software development and presented in 1995 paper.
- Term is based on rugby term
- Small cross-functional teams

SCRUM Practices

- **Product and release *backlog***
  - A list of the features to be implemented in the project (subdivided to next release), ordered by priority
  - Can adjust over time as needed, based on feedback
  - A product manager is responsible for maintaining

SCRUM Practices

- **Burn-down chart**
  - Make best estimate of time to complete what is currently in the backlog
  - Plot the time on a chart
  - By studying chart, understand how team functions
  - Ensure burndown to 0 at completion date
    - By adjusting what's in the backlog
    - By adjusting the completion date

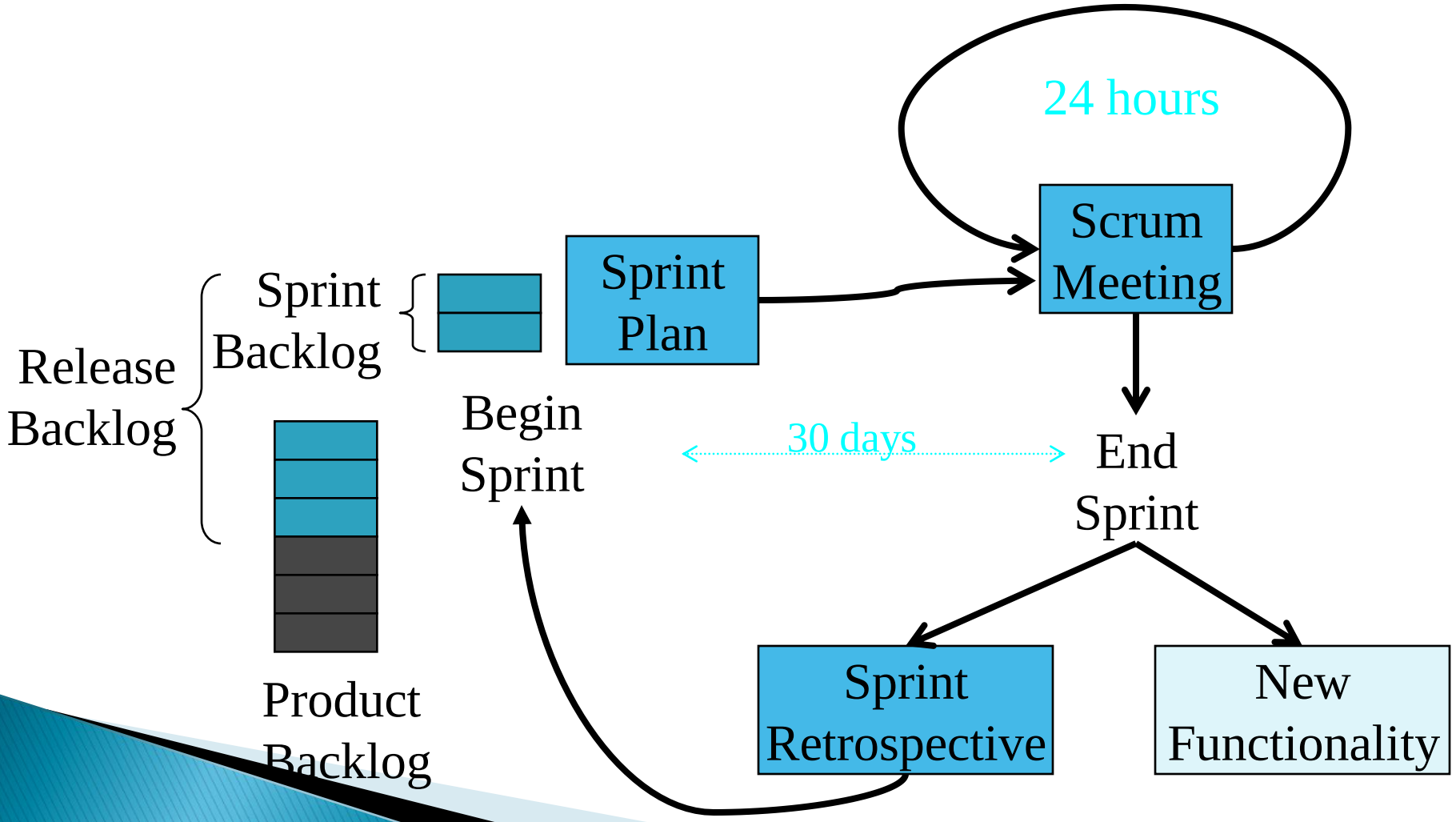
SCRUM Practices

- **Sprint**
  - The sprint is a ~1 month period after which some product is delivered
  - Features are assigned from the product backlog to a sprint backlog
    - Features divided into smaller tasks for sprint backlog
    - Feature list is fixed for sprint
  - Planning meeting
    - Tasks can be assigned to team members
    - Team members have individual estimates of time taken per item
  - During sprint, work through features, and keep a burn-down chart for the sprint
  - New functionality is produced by the end of the sprint
  - After sprint, a review meeting is held to evaluate the sprint

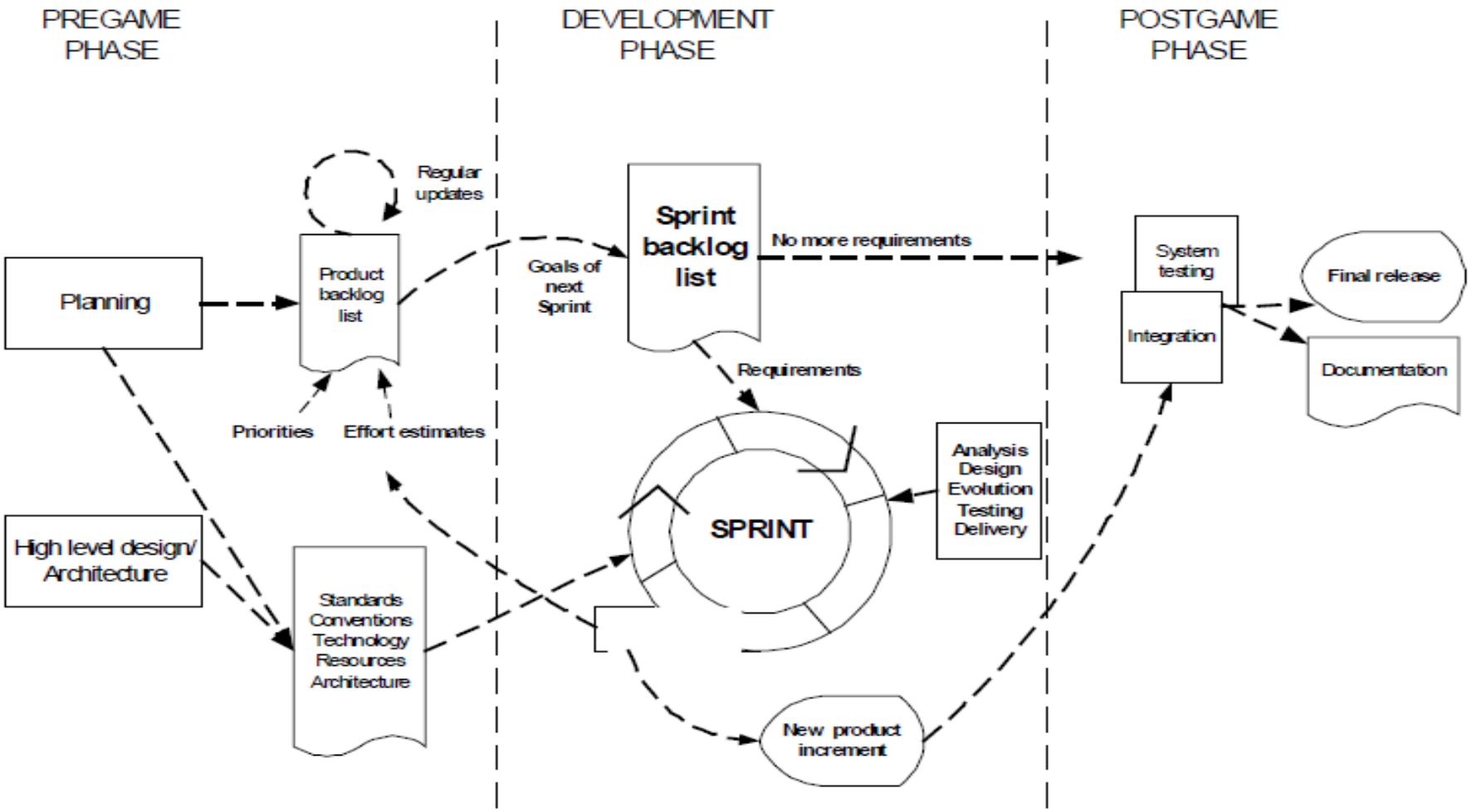
SCRUM Practices

- **Scrum meeting**
  - 15 minute *daily* meeting
  - All team members show up
  - Quickly mention what they did since last Scrum, any obstacles encountered, and what they will do next
  - Some team member volunteers or is appointed to be the *Scrum Master* - in charge of Scrum meeting, and responsible for seeing that issues raised get addressed
  - Customers, management encouraged to observe

SCRUM


24 hours
Scrum Meeting
Sprint Plan
Sprint Backlog
Release Backlog
Begin Sprint
30 days
End Sprint
Product Backlog
Sprint Retrospective
New Functionality

SCRUM Process



SCRUM Project Phases

Scrum processes address the specific activities and flow of a Scrum project. In total there are 19 processes which are grouped into following five phases:

1. **Initiate -** This phase includes the processes related to initiation of a project: Create Project Vision, Identify Scrum Master and Stakeholder(s), Form Scrum Team, Develop Epic(s), Create Prioritized Product Backlog, and Conduct Release Planning.

SCRUM Project Phases

2. **Plan and Estimate -** This phase consists of processes related to planning and estimating tasks, which include Create User Stories, Approve, Estimate, and Commit User Stories, Create Tasks, Estimate Tasks, and Create Sprint Backlog.

SCRUM Project Phases

3. **Implement -** This phase is related to the execution of the tasks and activities to create a project's product. These activities include creating the various deliverables, conducting Daily Standup Meetings, and grooming (i.e., reviewing, fine-tuning, and regularly updating) the Product Backlog at regular intervals.

SCRUM Project Phases

4. **Review and Retrospect -** This phase is concerned with reviewing the deliverables and the work that has been done and determining ways to improve the practices and methods used to do project work.

SCRUM Project Phases

5. **Release -** This phase emphasizes on delivering the Accepted Deliverables to the customer and identifying, documenting, and internalizing the lessons learned during the project.

SCRUM Roles

Scrum Team comprise of –

1. Team Lead
2. Team Member
3. Product Owner
4. Stakeholder

Compulsory

5. Technical Expert
6. Independent Tester
7. Architecture Owner

Optional

1. **Team Lead –**

   This role, called “Scrum Master” in Scrum or team coach or project lead in other methods, is responsible for facilitating the team, obtaining resources for it, and protecting it from problems. This role encompasses the soft skills of project management but not the technical ones such as planning and scheduling, activities which are better left to the team as a whole (more on this later).

2. **Team Member –**

This role, sometimes referred to as developer or programmer, is responsible for the creation and delivery of a system. This includes modeling, programming, testing, and release activities, as well as others.

3. **Product Owner –**

The product owner, called on-site customer in XP and active stakeholder in AM, represents the stakeholders. This is the one person responsible on a team (or sub-team for large projects) who is responsible for the prioritized work item list (called a product backlog in Scrum), for making decisions in a timely manner, and for providing information in a timely manner.

4. **Stakeholders –**

A stakeholder is anyone who is a direct user, indirect user, manager of users, senior manager, operations staff member.

The "gold owner" who funds the project, support (help desk) staff member, auditors, your program/portfolio manager.

SCRUM Terminologies

- **Product Backlog** : The agile product backlog in Scrum is a prioritized features list, containing short descriptions of all functionality desired in the product. A typical Scrum backlog comprises the following different types of items:

1. Features
2. Bugs
3. Technical work
4. Knowledge acquisition

SCRUM Terminologies

- **Sprint Backlog :** The sprint backlog is a list of tasks identified by the Scrum team to be completed during the Scrum sprint.

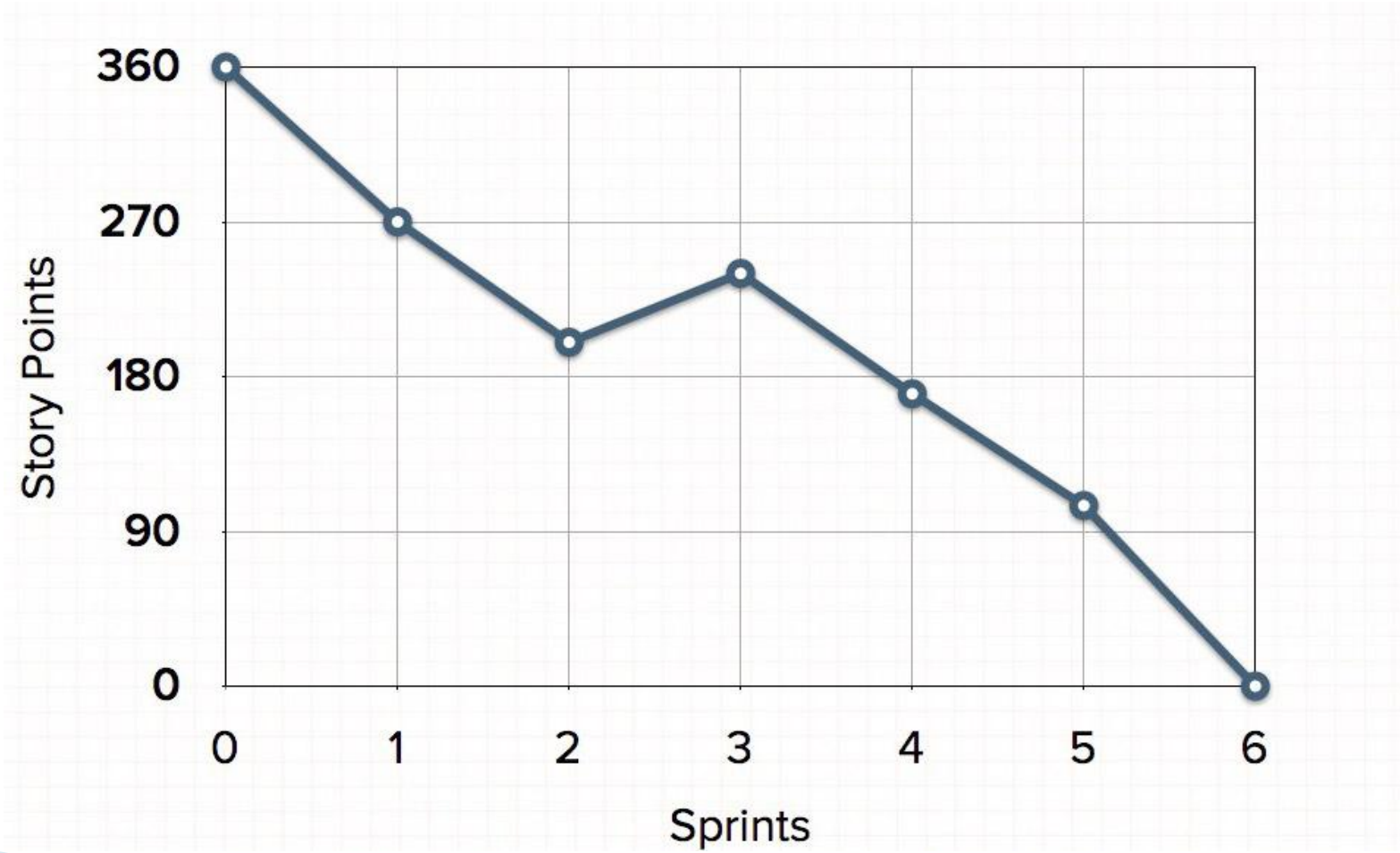
During the sprint planning meeting, the team selects some number of product backlog items, usually in the form of user stories, and identifies the tasks necessary to complete each user story.

SCRUM Terminologies

- **Release Burndown Chart :** On a Scrum project, the team tracks its progress against a release plan on a release burndown chart. The release burndown chart is updated at the end of each sprint by the Scrum Master.

The horizontal axis of the sprint burndown chart shows the sprints; the vertical axis shows the amount of work remaining at the start of each sprint. Work remaining can be shown in whatever unit the team prefers -- story points, ideal days, team days and so on.

Example of Burndown Chart



Feature Driven Development

Features are client valued functionalities or business requirements from the system. FDD focuses on delivering tangible working software repeatedly in timely manner. FDD emphasizes on listening, planning, designing and building of features. It comprise of five steps :

1. Develop overall model
2. Build feature list
3. Plan by feature
4. Design by feature
5. Build by feature

1. **Develop over all model –**

- Starts with a high level walkthrough of the scope of the system and its context.

- Detailed domain model are created for each modeling area by small groups and presented for peer review

- One of the proposed models or a combination of them is selected as a final model.

2. **Build Feature List –**

- List of features is identified by functionalities based on initial modeling.
- Features are then decomposed in subject areas i.e. business activities
- Features as individual should not take ore than 2 weeks to be implemented, else need to be broken

3**. Plan by Feature –**

- Produce a development plan.
- Assign ownership of the features as classes to programmers.
- Perform resource scheduling.

4. **Design by Feature –**

- A design package is produced for each feature.
- Chief programmer along with corresponding class owner selects and design for a set of features.
- Chief Programmers refines overall model.
- A final design inspection is held.

5. **Build by Feature –**

- After successful design feature as planned is developed.
- Class owner develops code for their classes.
- After a unit test and successful code inspection completed feature is promoted to the main build.

FDD Tools

- CASE Spec
- Tech Excel Dev Suite
- FDD Tool Kit
- FDD Viewer

Lean Software Development

It comprise of seven activities :

1. Eliminate Waste
2. Amplify Learning
3. Decide as late as possible
4. Deliver as fast as possible
5. Empower the team
6. Build Integrity In
7. See the whole system

1. **Eliminate Waste –**

   All the wastes that delay in development such as unnecessary code, functionalities, unclear requirements, avoidable process repetitions etc should be identified along with their sources and should be eliminated.

2. **Amplify Learning –**

- Software development is a continuous learning process.
- It should be speed up by using short iteration cycle coupled with refactoring and integration testing.
- Instead of more documentation different ideas are abstracted by presenting screen shots to the end user and getting their inputs.

3. **Decide as late as possible –**

Focuses on delaying decisions as much as possible until they can be made confirm on the basis of facts. But not uncertain assumptions or predictions should be entertained.

4. **Deliver as fast as possible –**

- As soon as the end product is delivered, the feedbacks can be obtained and can be incorporated into the next iteration.
- It focuses on rapid delivery of a quality product just after delayed decision.

5. **Empower the Team –**

- Developers should be given complete access to the customer with proper support in difficult situations by their managers.
- The decision making of the team members must be respected to deliver a quality product.

6. **Build Integrity In –**

- Customer needs to have an overall experience of the system called as Perceived Integrity means how it is delivered and deployed.
- Focuses on Conceptual Integrity that means how sub modules of the system work well on integration, of their balancing, flexibility, maintainability and efficiency.
- At last integrity should be verified with thorough testing.

7. **See the Whole System–**

- Software is not a sum of their parts but a product of their interaction.
- The focus on overall development should be noticed while developing unit parts, as the defect incorporated in them may cause harm to the whole system.

Other Agile Methods

- Crystal
- Feature-driven development (FDD)
- Adaptive software development (ASD)
- Dynamic System Development Method (DSDM)

Agile Estimation

There are three main concepts you need to understand to do agile estimation:

- **Estimation of Size** gives a high-level estimate for the work item, typically measured using a neutral unit such as points
- **Velocity** tells us how many points this project team can deliver within an iteration
- **Estimation of Effort** translates the size (measured in points) to a detailed estimate of effort typically using the units of Actual Days or Actual Hours. The estimation of effort indicates how long it will take the team member(s) to complete the assigned work item(s).

Benefits of Agile Methods

- For product development where a software company is developing a small or medium sized product.
- For custom system development within an organization where customer involvement is committed in the development process.
- For the projects where not lot many external rules and regulations are to be imposed.

Drawbacks of Agile Methods

- Less focus on design and plan.
- No proper documentation.
- Limited team
- No rigorous requirement elicitation and modeling
- Haphazard implementation process (No sequence)

Challenges of Agile

?

Code Refactoring

- Code refactoring is the process of restructuring existing computer code without changing its external behavior.
- While refactoring new code is not developed rather improvement of the existing code is focussed.
- Refactoring improves non functional attributes of the software.
- Advantages include improved code readability and reduced complexity to improve source code maintainability, and create a more expressive internal architecture or object model to improve extensibility.

What are Functional & Non Functional Attributes of software

?

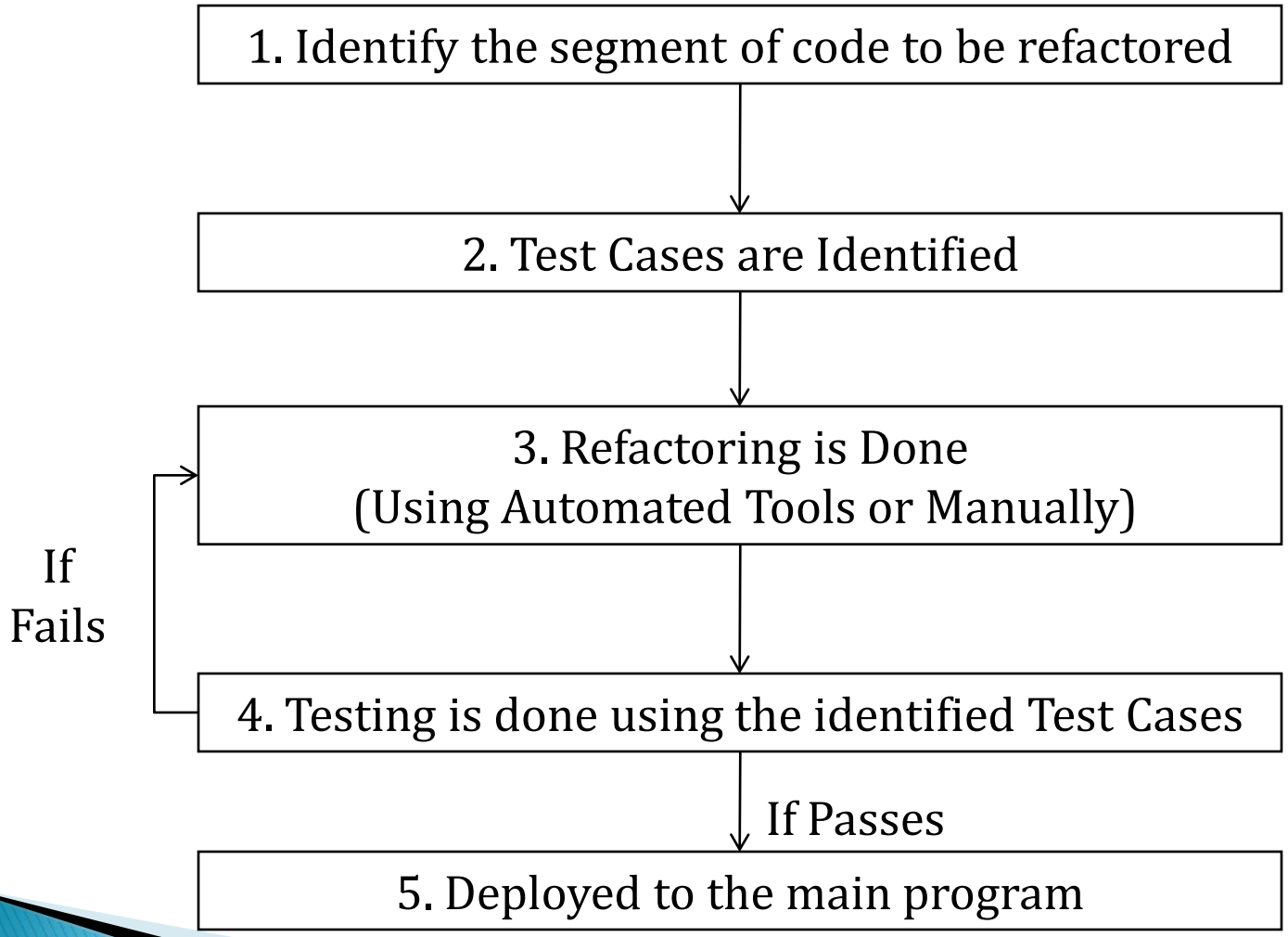
Typically, refactoring applies a series of standardized basic micro-refactorings, each of which is (usually) a tiny change in a computer program's source code that either preserves the behavior of the software, or at least does not modify its conformance to functional requirements.

Why to Refactor a code ?

There may be some stereo- typical situations where program code should be improved. Such situations are known as Code Smells. Some of them are –

1. Duplicate Code
2. Long Methods
3. Switch Case Statements
4. Data Clumping
5. Speculative Generality

Process of Code Refactoring



Benefits of Code Refactoring

1. **Maintainability -** It is easier to fix bugs because the source code is easy to read and the intent of its author is easy to grasp.

2. **Extensibility -** It is easier to extend the capabilities of the application if it uses recognizable design patterns, and it provides some flexibility where none before may have existed

Continuous Integration

- Continuous integration (CI) is the practice, in software engineering, of merging all developer working copies with a shared mainline several times a day.

- It was first named and proposed by Grady Booch in his method, who did not advocate integrating several times a day.

- It was adopted as part of extreme programming (XP).

- CI typically use a build server to implement continuous processes of applying quality control in general — small pieces of effort, applied frequently.

- In addition to running the unit and integration tests, such processes facilitate manual QA processes.

- This continuous application of quality control aims to improve the quality of software, and to reduce the time taken to deliver it.

Principles of Continuous Integration

1. **Maintain a code repository –**

   This practice advocates the use of a revision control system for the project's source code. All artifacts required to build the project should be placed in the repository.

2. **Automate the build –**

   A single command should have the capability of building the system. Many build-tools, such as make, Debian DEB, Red Hat RPM or Windows MSI files

3. **Make the build self-testing -**

   Once the code is built, all tests should run to confirm that it behaves as the developers expect it to behave.

4. **Everyone commits to the baseline every day -**

   By committing regularly, every committer can reduce the number of conflicting changes.

5. **Every commit (to baseline) should be built -**

   The system should build commits to the current working version to verify that they integrate correctly.

6. Keep the build fast -

The build needs to complete rapidly, so that if there is a problem with integration, it is quickly identified.

7. Test in a clone of the production environment -

Having a test environment can lead to failures in tested systems when they deploy in the production environment, because the production environment may differ from the test environment in a significant way.

8. **Make it easy to get the latest deliverables -**

Making builds readily available to stakeholders and testers can reduce the amount of rework necessary when rebuilding a feature that doesn't meet requirements.

9. Everyone can see the results of the latest build -

It should be easy to find out whether the build breaks and, if so, who made the relevant change.

10. Automate deployment -

Most CI systems allow the running of scripts after a build finishes. o, who made the relevant change.

Benefits of Continuous Integration

1. Integration bugs are detected early and are easy to track down due to small change sets.

2. Avoids last-minute chaos at release dates.

3. Constant availability of a “current” build for testing, demo, or release purposes

4. Frequent code check-in pushes developers to create modular, less complex code

Pair Programming

- Pair programming is an agile software development technique in which two programmers work together at one workstation.

One, the driver, writes code while the other,

the observer or navigator, reviews each line of code as it is typed in.

The two programmers switch roles frequently.

While reviewing, the observer also considers the "strategic" direction of the work, coming up with ideas for improvements and likely future problems to address.

This frees the driver to focus all of his or her attention on the "tactical" aspects of completing the current task, using the observer as a safety net and guide.

Benefits of Pair Programming

1. Pair programming increases the man-hours required to deliver code compared to programmers working individually.

2. A system with two programmers possesses greater potential for the generation of more diverse solutions to problems for three reasons:
   - ✓ *the programmers bring different prior experiences to the task;*
   - ✓ *they may access information relevant to the task in different ways;*
   - ✓ *they stand in different relationships to the problem by virtue of their functional roles.*

3. Knowledge is constantly shared between pair programmers.

4. Pair programming allows team members to share problems and solutions quickly making them less likely to have hidden agendas from each other.

Agile Design Principles & Practices

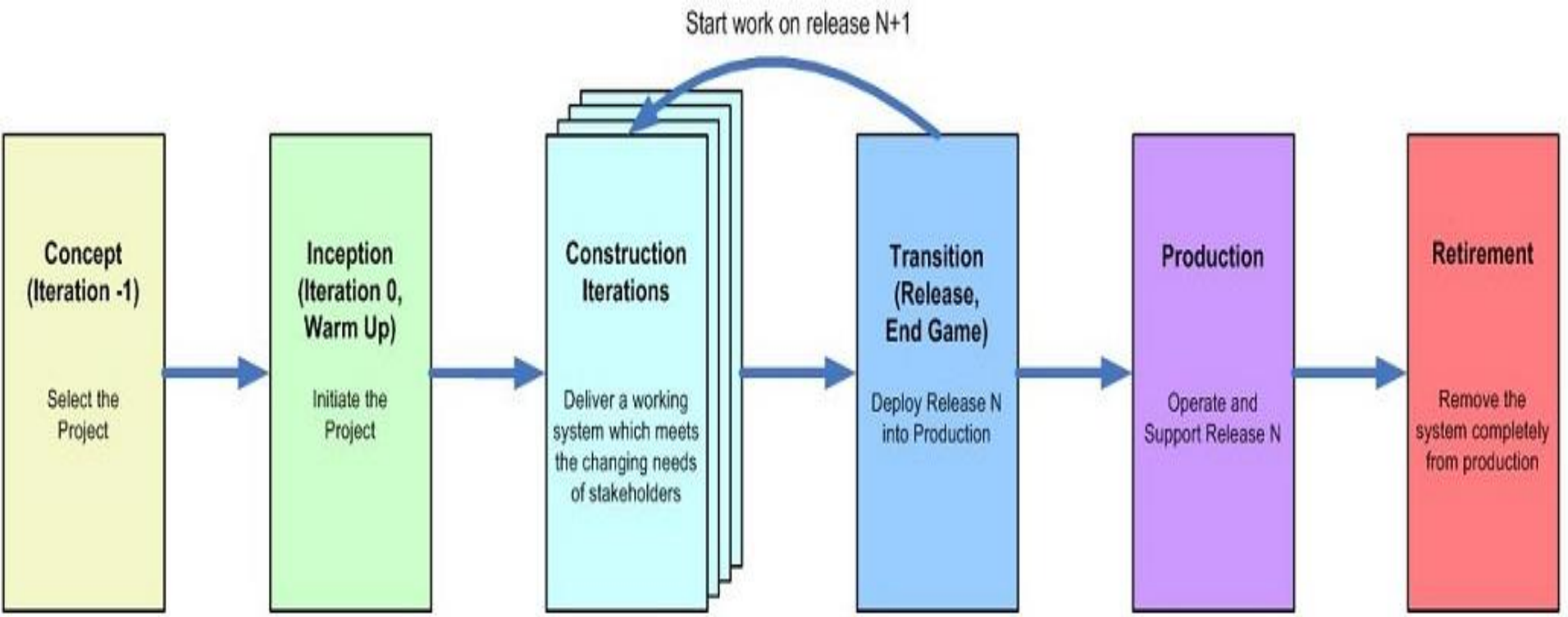
There is a range of agile design practices from High Level Architectural to Low Level Programming Practices. They are needed for an effective agile design and modeling.

1. Architectural Envisioning : Light weight modeling at the beginning of a project to identify and think through critical architectural issues.

2. Iteration Modeling : Light weight modeling for a few minutes. At the beginning of iteration or sprint to help identify team's strategy for that iteration.

3. Models Storming : Light weight modeling for a few minutes on just in time basis to think through the aspects of the suggested solutions.
4. Test First Design (TFD) : Write a single test before writing enough production code to fulfill that test.
5. Refactoring : Make a small change to a part of your solution which improves the quality without changing the semantics of part.
6. Continuous Integration : Automatically compile test and validate the components of your solution whenever one of those components change.

Agile Project Management



- Identify potential projects
- Prioritize potential projects
- Develop initial vision
- Consider project feasibility

- Active stakeholder participation
- Obtain funding and support
- Start building the team
- Initial requirements envisioning
- Initial architecture envisioning
- Setup environment

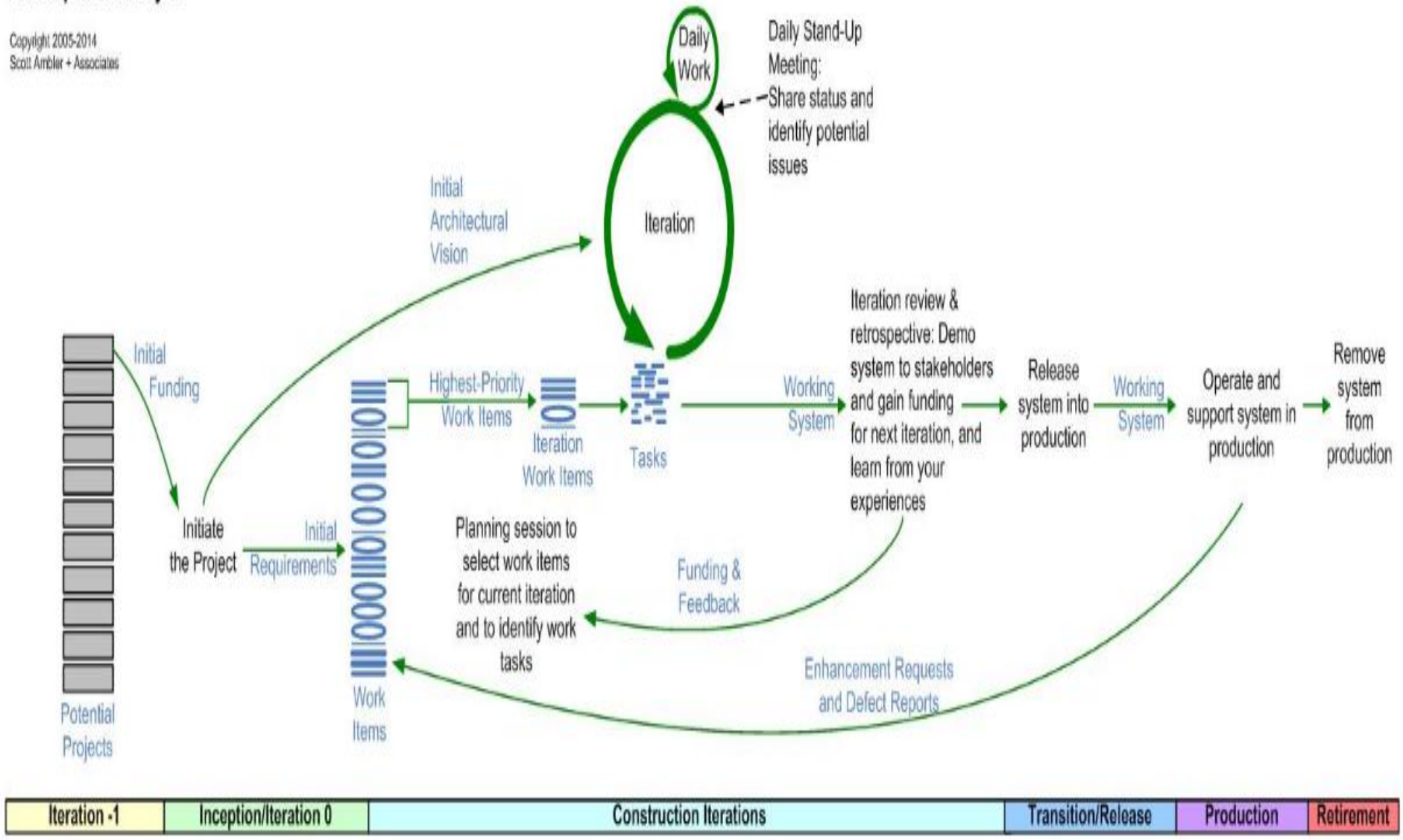
- Active stakeholder participation
- Collaborative development
- Model storming
- Test driven design (TDD)
- Confirmatory testing
- Evolve documentation
- Internally deploy software

- Active stakeholder participation
- Final system testing
- Final acceptance testing
- Finalize documentation
- Pilot test the release
- Train end users
- Train production staff
- Deploy system into production

- Operate the system
- Support the system
- Identify defects and enhancements

- Remove the final version of the system-Data conversion
- Migrate users
- Update enterprise models

Agile System Development Lifecycle



Agile Project Management Tools

- JIRA
- KANBAN
- TAIGA

Agile Testing

- **Agile testing** is a software testing practice that follows the principles of agile software development.
- Agile testing involves all members of a cross-functional agile team, with special expertise contributed by testers, to ensure delivering the business value desired by the customer at frequent intervals, working at a sustainable pace.
- Agile development recognizes that testing is not a separate phase, but an integral part of software development, along with coding. Agile teams use a "whole-team" approach to "baking quality in" to the software product.

1	Concurrent / Regular Testing	During the development of code by observer
2	Unit Testing	After the completion of each sprint by pair programmers
3	Integration Testing	During Continuous Integration at Central Repository
4	System Testing	After the completion of all sprints / End Game
5	Acceptance Testing	While delivering the project to the customer

- Testers on agile teams lend their expertise in eliciting examples of desired behavior from customers, collaborating with the development team to turn those into executable specifications that guide coding.

"**Testing and coding are done incrementally and interactively**"

- In contrast with other methodologies, Agile testing focuses on repairing faults immediately, rather than waiting for the end of the project. By doing so, costs are expected to go down

Agile Testing Tools

- JIRA
- BUG DIGGER
- FOGBUGZ
- PIVOTAL TRACKER
- SNAGIT