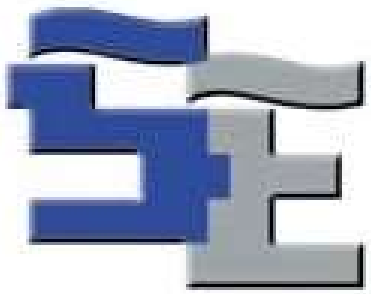


Software Engineering

Agile principles

Fundamentals of agile development

- **Agile software development** is a group of software development methodologies based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams.

Agile Manifesto

- Manifesto for Agile Software Development
- To define the approach now known as agile software development

- AGILE methodology is a practice that promotes **continuous iteration** of development and testing throughout the software development lifecycle of the project.
- The Agile Manifesto states that −
  - We are uncovering better ways of developing software by doing it and helping others do it.

The agile software development emphasizes on four core values.

- Individual and team interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Agile principles

1. Customer satisfaction by rapid delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Working software is the principal measure of progress
5. Sustainable development, able to maintain a constant pace
6. Close, daily co-operation between business people and developers

7. Face-to-face conversation is the best form of communication (co-location)
8. Projects are built around motivated individuals, who should be trusted
9. Continuous attention to technical excellence and good design
10. Simplicity
11. Self-organizing teams
12. Regular alteration to changing situations

Characteristics of Agility

- Agility in Agile Software Development focuses on the culture of the whole team with multi-discipline, cross-functional teams that are empowered and self organizing.
- It fosters shared responsibility and accountability.
- Facilitates effective communication and continuous collaboration.
- The whole-team approach avoids delays and wait times.
- Frequent and continuous deliveries ensure quick feedback that in in turn enable the team align to the requirements.
- Collaboration facilitates combining different perspectives timely in implementation, defect fixes and accommodating changes.
- Progress is constant, sustainable, and predictable emphasizing transparency.

Agile methods

- Agile Modeling
- Agile Unified Process (AUP)
- Dynamic Systems Development Method (DSDM)
- Essential Unified Process (EssUP)
- Extreme Programming (XP)
- Feature Driven Development (FDD)
- Open Unified Process (OpenUP)
- RAD
- Scrum
- Velocity tracking

SCRUM

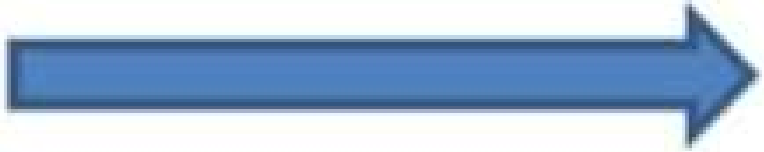
- **Scrum** is an agile framework for managing work with an emphasis on software development.
- Scrum believes in empowering the development team and advocates working in small teams (say- 7 to 9 members).
- It consists of three roles, and their responsibilities are explained as follows:

- **Scrum Master**
  - Master is responsible for setting up the team, sprint meeting and removes obstacles to progress
- **Product owner**
  - The Product Owner creates product backlog, prioritizes the backlog and is responsible for the delivery of the functionality at each iteration
- **Scrum Team**
  - Team manages its own work and organizes the work to complete the sprint or cycle

Scrum Master

Scrum Meeting

Product Owner



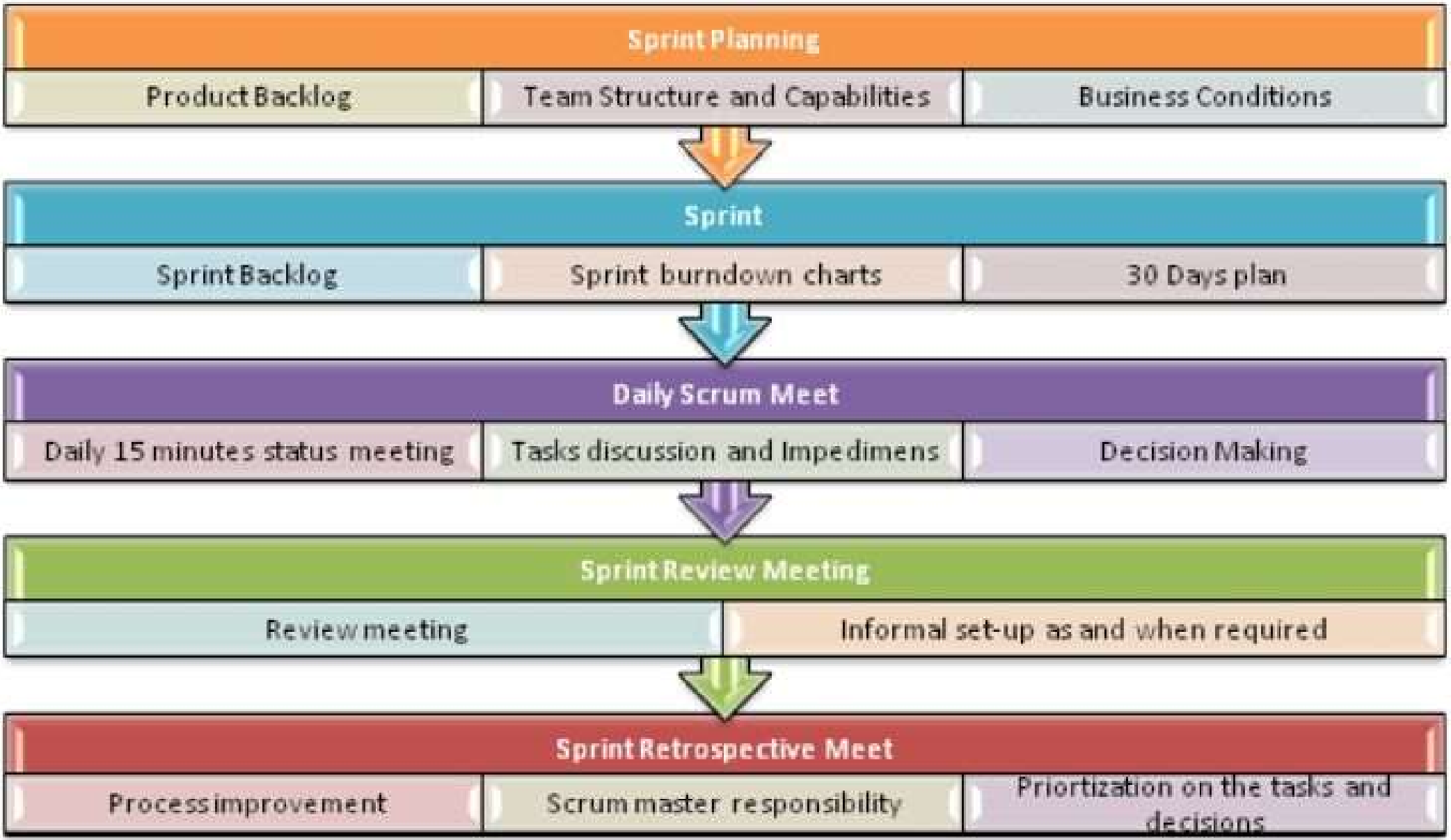
Prepare product backlog

Team organizes tasks

Product Backlog

- This is a repository where requirements are tracked with details on the no. of requirements to be completed for each release.
- It should be maintained and prioritized by Product Owner, and it should be distributed to the scrum team.
- Team can also request for a new requirement addition or modification or deletion

Scrum Practices



Process flow of Scrum Methodologies:

- Each iteration of a scrum is known as **Sprint**
- **Product backlog** is a list where all details are entered to get end product
- During each Sprint, top items of Product backlog are selected and turned into **Sprint backlog**
- Team works on the defined sprint backlog
- Team checks for the daily work
- At the end of the sprint, team delivers product functionality

Extreme Programming

- eXtreme Programming (XP) was conceived and developed to address the specific needs of software development by small teams in the face of vague and changing requirements.
- Extreme Programming is one of the Agile software development methodologies.
- It provides values and principles to guide the team behaviour.
- The team is expected to self-organize.

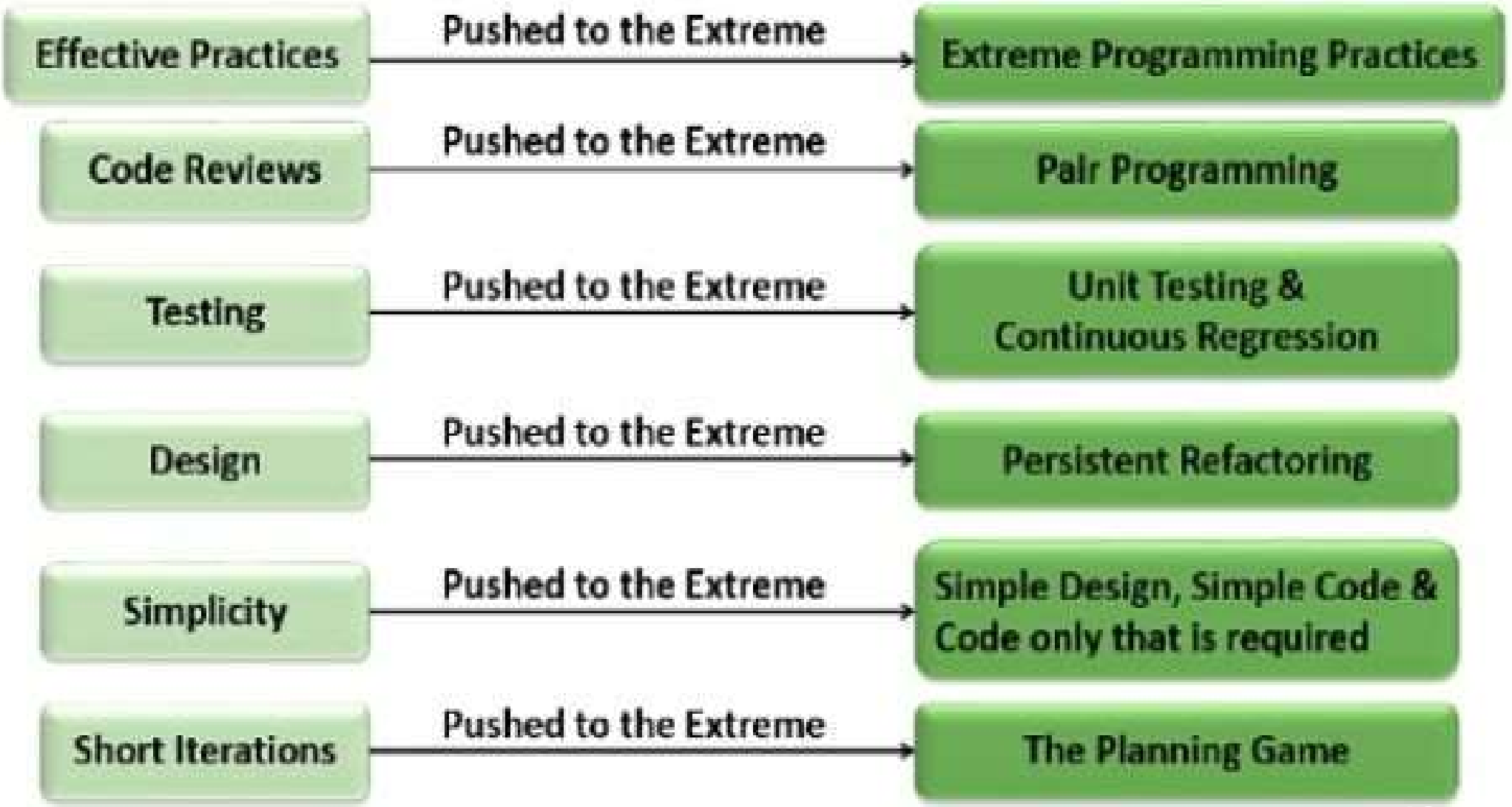
Extreme Programming involves –

- Emphasis on continuous feedback from the customer
- Short iterations
- Design and redesign
- Coding and testing frequently
- Eliminating defects early, thus reducing costs
- Keeping the customer involved throughout the development
- Delivering working product to the customer

Why is it called "Extreme?"

Extreme Programming takes the effective principles and practices to **extreme levels**.

- Code reviews are effective as the code is reviewed all the time.
- Testing is effective as there is continuous regression and testing.
- Design is effective as everybody needs to do refactoring daily.
- Integration testing is important as integrate and test several times a day.
- Short iterations are effective as the planning game for release planning and iteration planning.


Effective Practices
Pushed to the Extreme
Extreme Programming Practices
Code Reviews
Pushed to the Extreme
Pair Programming
Testing
Pushed to the Extreme
Unit Testing &
Continuous Regression
Design
Pushed to the Extreme
Persistent Refactoring
Simplicity
Pushed to the Extreme
Simple Design, Simple Code &
Code only that is required
Short Iterations
Pushed to the Extreme
The Planning Game

XP Roles

- **Customer**: He decides what the project needs to do. And provides the user stories.
- **Programmer**: Defines the architecture & writes the code. Implements the user stories & unit testing
- **Tracker**: Monitor the team member's progress
- **Coach**: Helps the team work effectively, self organise, and use good XP practices
- **Tester**: Runs functional tests, acceptance test
- **Administrator**: Set up and maintains the team members computers, network, and development tools

XP VALUES & PRINCIPLES

Communication

- between team members and also with the users

Simplicity

- The methodology favours simple designs

Feedback

- feedback can work in different ways- testing, reviews, etc

Courage

- Solve the problem for today not for tomorrow

Respect

- communication between different stakeholders as well

Most common XP practices

- Have a customer on site
- Play the planning game
- Use stand up meetings
- Make frequent small releases
- Use intuitive metaphors
- Keep designs simple
- Defer optimization
- Refactor when necessary
- Give everyone ownership of the code
- Use coding standards
- Promote generalization
- Use pair programming
- Test constantly
- Integrate continuously
- Work sustainably
- Use test-driven and test-first development.

Advantages of Extreme Programming

- To save costs and time required for project realization. Extreme Programming teams save lots of money because they don’t use too much documentation. They usually solve problems through discussions inside of the team.
- Simplicity
- The whole process in XP is visible and accountable.
- Constant feedback
- XP assists to create software faster thanks to the regular testing at the development stage.
- Employee satisfaction and retention.

Drawbacks of Extreme Programming

- Not effective in larger groups
- Pair programming does not work well in many cases
- The dependence on the customer
- Extreme Programming's focus on simplicity may make it very difficult to add to the current project
- Extreme Programming team is entirely dependent on the emotional maturity of all team members, a factor that is not always dependable.

FEATURE- DRIVEN DEVELOPMENT (FDD)

- Feature-Driven Development (FDD) is another iterative and incremental development model.
- FDD was created to work with **large teams**
- This method is focused around "**designing & building**" features.

FDD Roles – primary roles

- **Project manager -** This is the project's administrative leader. The project manager tracks the project's progress, budget, and other resources.
- **Chief architect -** This person is responsible for the project's overall programmatic design. The chief architect doesn't write the design but instead helps the team come up with a design cooperatively.
- **Development manager -** This person manages day-to-day development activities. The development manager resolves conflicts, makes sure the development teams have the resources they need, etc

- **Chief programmers -** These are experienced developers who are familiar with all the functions of development (design, analysis, coding, and so on). They lead teams of programmers who work on a set of assigned programming tasks.
- **Class owner -** In many agile models, the code is jointly owned by the entire development team, so any member can modify anyone else's code. In contrast, FDD assigns ownership of each class to a specific developer.
- **Domain expert -** These are customers, users, executive champions, and others who know about the project domain and how the finished application should work. They are the source of information for the developers. They are sometimes called Subject Matter Experts (**SME** s).

FDD Secondary roles (mostly in larger projects)

- **Build engineer-** Sets up and controls the build process.
- **Deployer -** Handles deployment
- **Domain manager -** Leads the domain experts and to resolve domain issues
- **Language guru or language lawyer -** an expert in the programming language
- **Release manager -** To track the project's progress
- **System administrator -**Maintains computers and network
- **Technical writer -** Writes online & printed documentation
- **Internaler** -Verifies the application meets the requirements
- **Toolsmith** - Creates tools

FDD Phases

- FDD projects move through five phases. The first 3occur at the start of the project and the last 2 phases are repeated iteratively until the application is complete.
- The **5 FDD phases** are
    1) Develop a Model
    2) Build a Feature List
    3) Plan by Feature
    4) Design by Feature
    5) Build by Feature

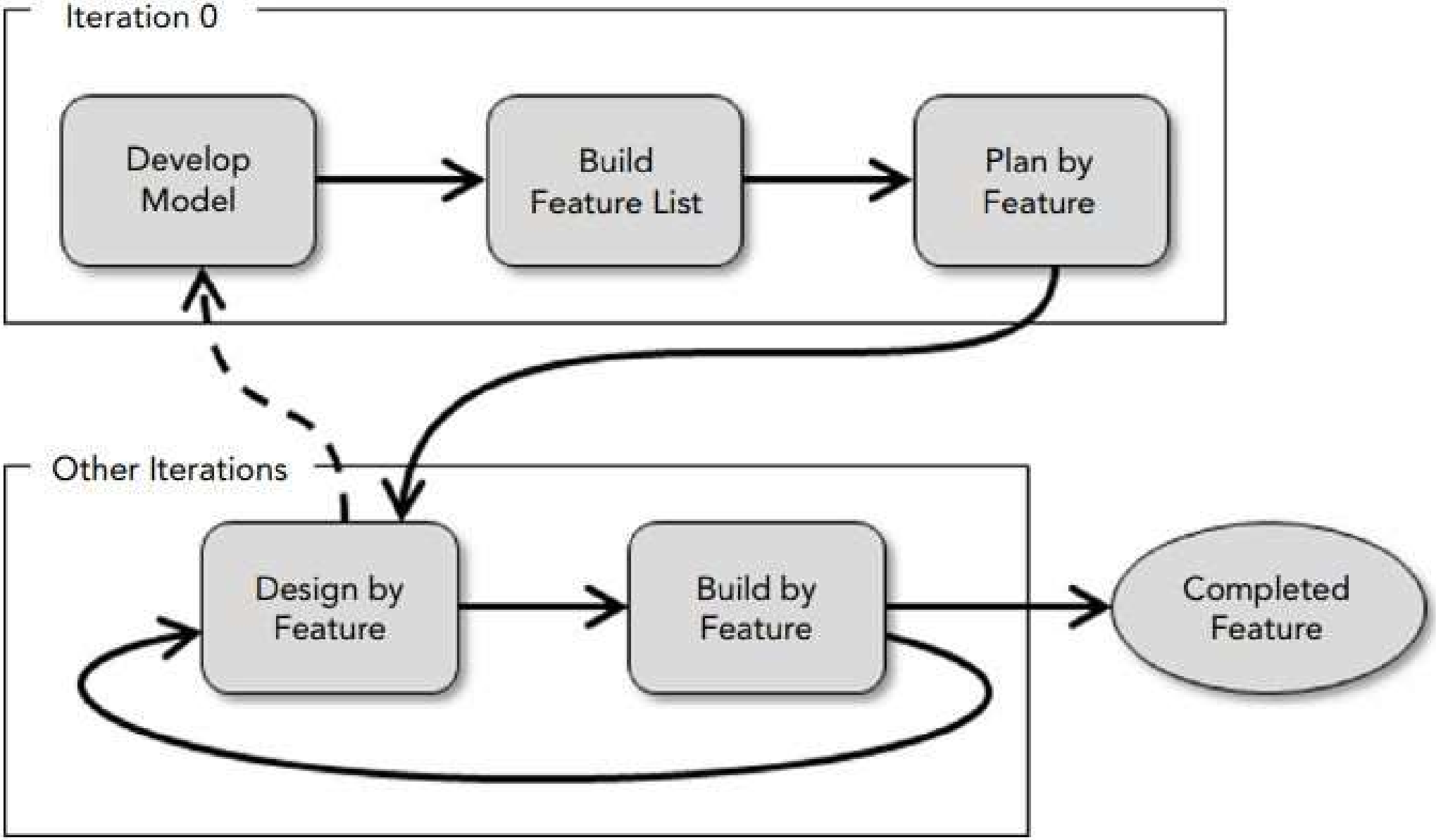


FIGURE 14-5: In FDD, the last two phases repeat for each feature iteration. The Design by Feature phase feeds changes back to the object model.

1. Develop a Model

- The team builds an object model for the application

2. Build a Feature List

- To build a list of the features that make up the application.
- FDD technically defines a feature as an **action/result/object** triple where the action generates the result related to the object.
- Eg: a feature might be “calculate the customer’s outstanding balance.”
- Here the action is “calculate,” the result is “outstanding balance,” and the object is “the customer.”

3. Plan by Feature

- The planning team prioritizes the features and builds an initial schedule.

4. Design by Feature

- A design package is produced for each feature.
- A chief programmer selects a small group of features that are to be developed within two weeks.
- Finally a design inspection is held.

5. Build by Feature

- To produce a feature is planned, the class owners develop code for their classes.
- After unit testing and successful code inspection, the completed feature is promoted to the main build.

FDD Milestones

- To keep track of everything that's going on during an iteration, FDD defines **6 milestones**.
- The first three milestones are completed during the **Design By Feature** activity, and the last three are completed during the **Build By Feature activity**

PHASE	MILESTONE	PERCENTAGE
Design by Feature	Domain Walkthrough	1%
	Design	40%
	Design Inspection	3%
Build by Feature	Code	45%
	Code Inspection	10%
	Promote to Build	1%

Lean Software Development

- Also called simply **Lean** or **LSD**
- Lean is an application of principles learned in lean manufacturing to software engineering.
- The idea behind Lean is to keep the application as **lean and fat-free** as possible
- Lean focuses on managing iterations of development.
- Lean focuses more closely on gathering the right requirements and ensuring that only essential ingredients get into each iteration.

Lean Principles

- Eliminating Waste
- Amplifying learning
- Defer commitment (deciding as late as possible)
- Early delivery
- Empowering the team
- Building Integrity
- Optimize the whole

Agile Project Management

- One of the latest project management strategies that is mainly applied to project management practice in software development.
- In an agile project, the entire team is responsible in managing the team and it is not just the project manager's responsibility.
- The agile project management function should also demonstrate the leadership and skills in motivating others. This helps retaining the spirit among the team members and gets the team to follow discipline.

Responsibilities of an Agile Project Manager

- Responsible for maintaining the agile values and practices in the project team.
- The agile project manager removes impediments as the core function of the role.
- Helps the project team members to turn the requirements backlog into working software functionality.
- Facilitates and encourages effective and open communication within the team.

Responsibilities of an Agile Project Manager

- Holding agile meetings that discusses the short-term plans and to overcome obstacles.
- Enhances the tool and practices
- chief motivator of the team
- In agile projects, it is everyone's (developers, quality assurance engineers, designers, etc.) responsibility to manage the project to achieve the objectives of the project.

Development practices in Agile Development

- Test-Driven Development
- Continuous Integration
- Refactoring
- Pair Programming
- Simple Design
- User Stories

Test-driven development

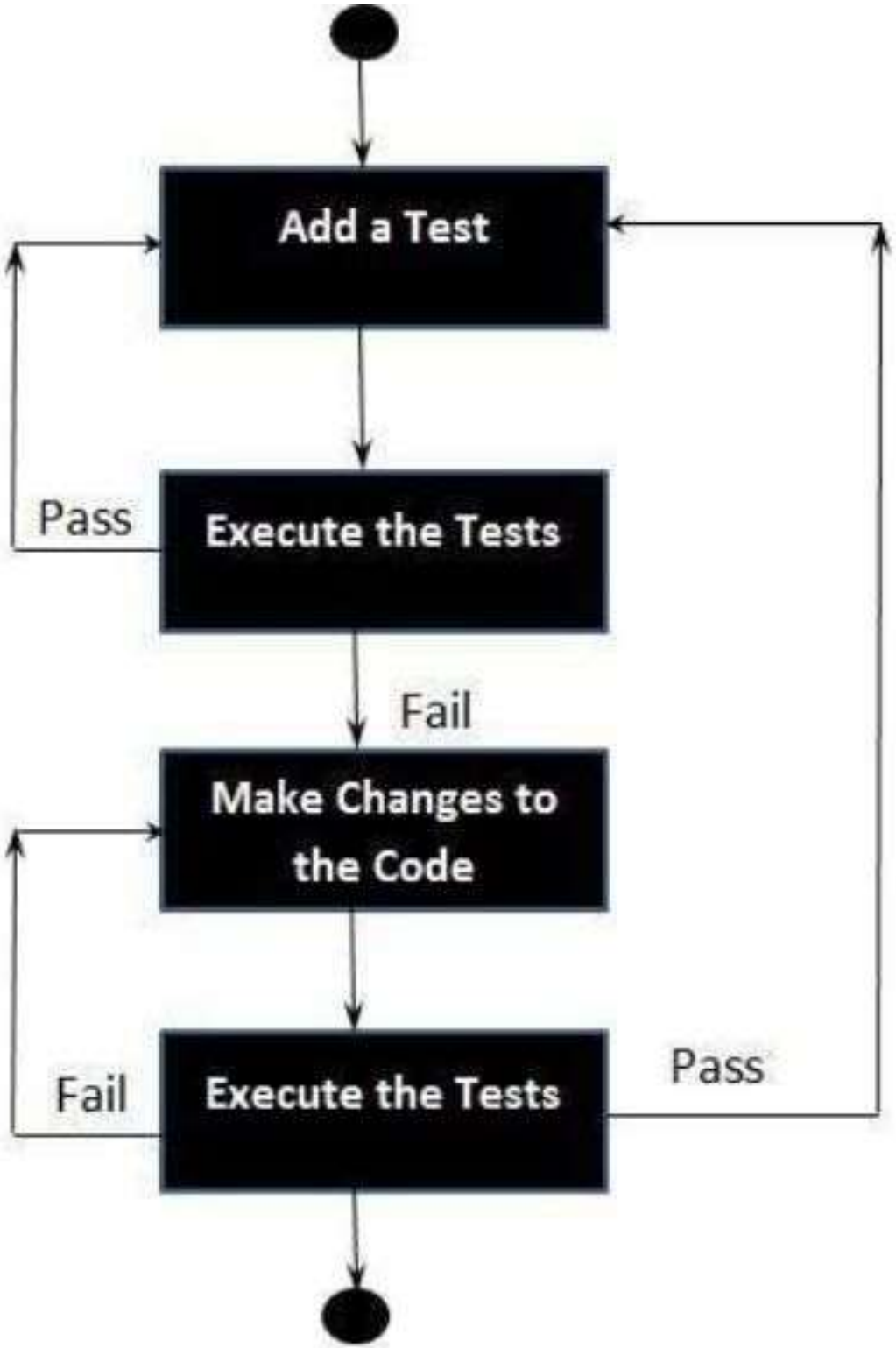
- TDD is a software development process that relies on the repetition of a very short development cycle:
- first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards

Test-driven development

- TDD can be defined as a programming practice that instructs developers to write new code only if an automated test has failed.
- This avoids duplication of code
- The primary goal of TDD is to make the code clearer, simple and bug-free.
- In TDD there are 3 activities are tightly interwoven: **coding, testing and design**

Test-Driven Development Process:

- Add a test
- Run all tests and see if the new one fails
- Write some code
- Run tests
- Refactor code
- Repeat


Add a Test
Pass
Execute the Tests
Fail
Make Changes to the Code
Fail
Execute the Tests
Pass

Benefits of TDD

- Early bug notification.
- Better Designed, cleaner and more extensible code
- Confidence to Refactor
- Good for teamwork
- Good for Developers

Common drawbacks

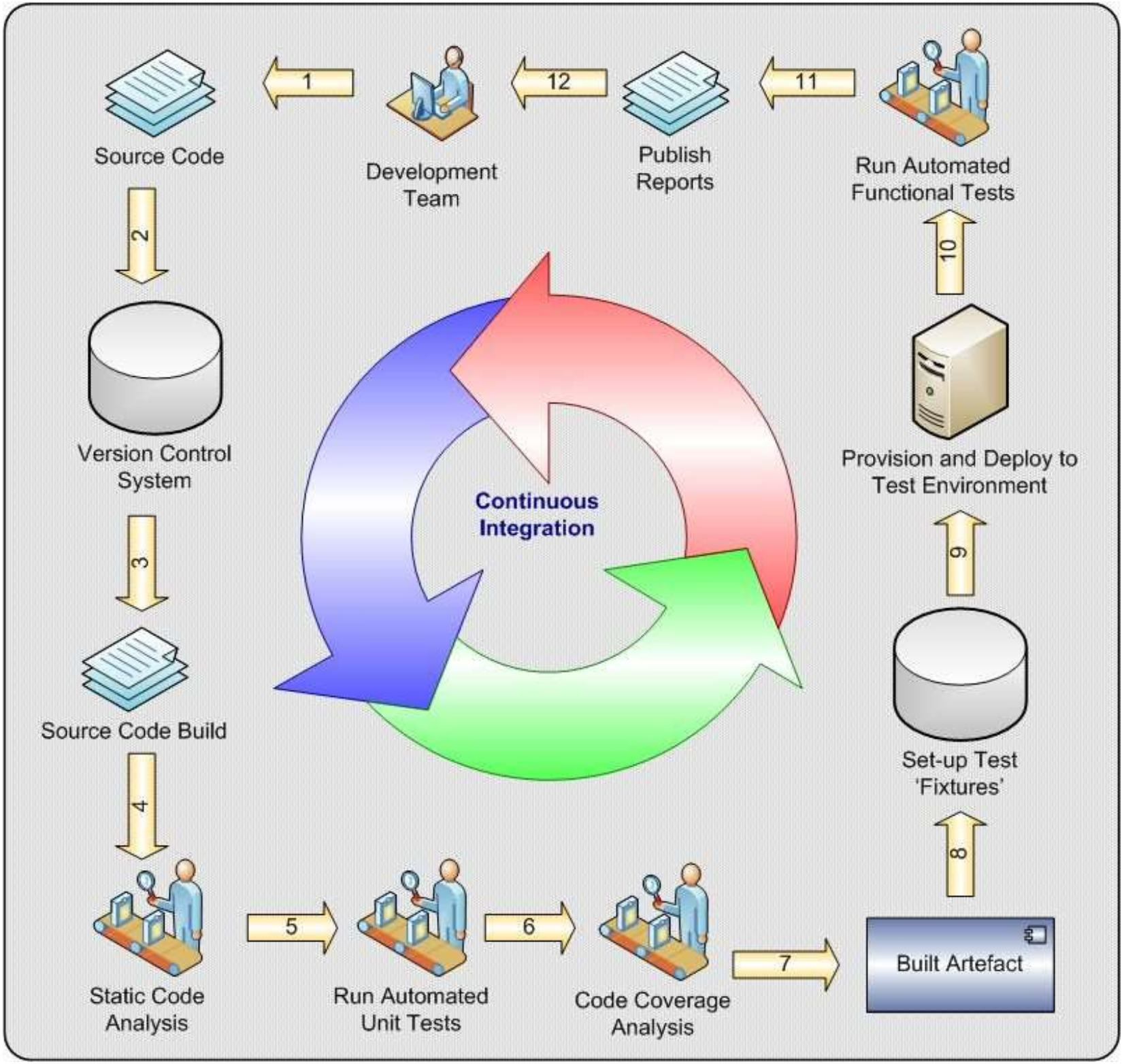
- Forgetting to run tests frequently
- writing too many tests at once
- writing tests that are too large or coarse-grained
- writing overly trivial tests, for instance omitting assertions
- writing tests for trivial code, for instance accessors

Continuous Integration

- It is a part of Extreme Programming (XP).
- The **main purpose of Continuous Integration** is to **prevent developers stepping over each other code** and **eliminate integration issues.**
- Continuous Integration (CI) is a practice in Software Engineering, where all the developers local working code base will be merged to share with a common repository several times during the product development.
- CI advocate integrating more than once per day – perhaps as many as tens of times per day.

CI common practices

- use of a version control tool (CVS, SVN, Git, etc.)
- an automated build and product release process
- to trigger unit and acceptance tests "every time any change is published to version control"
- If a single test failing, alerting the team of a "broken build" so that the team can reach a stable, releasable baseline again soonest
- optionally, the use of a tool such as a continuous integration server, which automates the process of integration, testing and reporting of test results


Source Code
1
Development Team
12
Publish Reports
11
Run Automated Functional Tests
2
10
Version Control System
Continuous Integration
Provision and Deploy to Test Environment
3
9
Source Code Build
Set-up Test 'Fixtures'
4
8
Static Code Analysis
5
Run Automated Unit Tests
6
Code Coverage Analysis
7
Built Artefact

Cycle of Continuous Integration

1. Developers work to transform the requirements or stories into source code using the programming language of choice.
2. They periodically check-in (commit) their work into a version control system (VCS)
3. It initiates the build process when it encounters a change.
4. Static analysis is performed on the source code
5. Run Automated unit tests

6. Calculate the production code by the unit tests is measured using a coverage analysis tool.
7. A binary artefact package is created.
8. Prepare for functional testing by setting up the test fixtures.
9. Prepare for functional testing by provisioning a test environment
10. Functional tests are executed.
11. Generate reports
12. The process is continuous...

Advantages of Continuous Integration

- Enables a quick feedback mechanism on the build results
- Helps collaboration between various product teams within the same organization
- Decreases the risk of regression
- Maintains version control within for various product releases and patch releases.
- Reduces the technical debt within the code.
- Allows earlier detection and prevention of defects
- Reduces manual testing effort

Disadvantages of Continuous Integration

- Continuous Integration tools maintenance and their administration have associated costs to it.
- Continuous Integration guidelines need to be well established before starting it.
- Test automation is a rare skill in the market and existing testers may have to be trained on that.
- Full fledge test coverage is required to see the benefits to automation.
- Teams sometimes depend too much on the unit testing and ignore automation and acceptance testing.

Refactoring

- Code Refactoring is the process of clarifying and simplifying the design of existing code, without changing its behaviour.
- "**Refactoring** is the process of changing a software system in such a way that it **does not alter the external behaviour of the code yet improves its internal structure."**

NEED AND SIGNIFICANCE OF REFACTORING

- **Upgrades over time**
  - Code outdated
  - The old code might also be using libraries that are not maintained or perhaps are now non-existent.
- **Upgrades - security**
  - Keeping your code up-to-date makes it easier to apply security patches when they are needed
- **Preparation for new, incoming features**
  - Most code will have to be altered in different places to work with new things you need to implement.

Reasons why Refactoring is Important

- To avoid bad smells in code
- To improve the design of software/application.
- To make software easier to understand.
- To find bugs
- To make program run faster.
- To fix existing legacy database
- To support revolutionary development
- To provide greater consistency for user.

Refactoring benefits:

- Makes code more readable.
- Cleanup code and makes it well ordered.
- Removes redundant, unused code and comments.
- Improves performance.
- Makes some things more generic.
- Keeps code DRY (Don’t Repeat Yourself)
- Combines and dispose “Like” or “Similar” code.
- Splitting out long functions into more manageable bite.
- Create re-usable code.
- Better class and function cohesion.

Bad Smells in Code

- Duplicated code
- Long Method
- Large Class
- Long Parameter List
- Big switch statements
- High coupling with the objects....

REFACTORING TECHNIQUES

- Extract Method
- Inline Method
- Extract Variable
- Inline Temp
- Replace Temp with Query

1) Extract Method

- You have a code fragment that can be grouped together. Turn the fragment into a method whose name explains the purpose of the method.

2) Inline Method

- Reverse of Extract method
- When a method body is more obvious than the method itself, use this technique.
- Replace calls to the method with the method's content and delete the method itself.

3) Extract Variable

- If an expression that is hard to understand, then separate its parts in separate variables that are self-explanatory.

4) Inline Temp

- The references to the variable with the expression itself.

5) Replace Temp with Query

- Query the method instead of using a variable.

Pair Programming

- All production code is written by pairs of programmers working together at the same workstation
- One member drives the keyboard and writes code and test cases; the second watches the code, looking for errors and possible improvements
- The roles will switch between the two frequently
- Pair membership changes once per day; so that each programmer works in two pairs each day
  - This facilitates distribution of knowledge about the state of the code throughout the entire team

Pair programming

User stories

- A user story is a tool used in Agile software development to capture a description of a software feature from an end-user perspective.
- The user story describes the type of user, what they want and why.
- A user story helps to create a simplified description of a requirement.

A user story template format:

- As a **<role>**, I want **<feature>** so that **<reason>**

- Examples
- As a **user,** I want **to upload photos** so that **I can share photos with others**.
- As an **administrator**, I want to **approve photos before they are posted** so that **I can make sure they are appropriate.**

Example user stories

- Students can purchase monthly parking passes online.
- Parking passes can be paid via credit cards.
- Parking passes can be paid via PayPal.
- Professors can input student marks.
- Students can obtain their current seminar schedule.
- Students can order official transcripts.
- Students can only enrol in seminars for which they have prerequisites.
- Transcripts will be available online via a standard browser.

- User stories are not documented in detail
  - we work out the scenario with the customer "face-to-face"; we give this scenario a name
  - the name is written on an **index card**
    - developers then write an estimate on the card based on the detail they got during their conversation with the customer
- The index card becomes a "**token**" which is then used to drive the implementation of a requirement based on its priority and estimated cost

Agile Testing

- A software testing practice that follows the principles of agile software development is called Agile Testing.
- Agile is an iterative development methodology, where requirements evolve through collaboration between the customer and self-organizing teams and agile aligns development with customer needs.

Advantages of Agile Testing

- Agile Testing Saves Time and Money
- Less Documentation
- Regular feedback from the end user
- Daily meetings can help to determine the issues well in advance

Principles of Agile Testing

- **Testing is NOT a Phase:** Agile team tests continuously and continuous testing is the only way to ensure continuous progress.
- **Testing Moves the project Forward:** When following conventional methods, testing is considered as quality gate but agile testing provide feedback on an ongoing basis and the product meets the business demands.
- **Everyone Tests:** In conventional SDLC, only test team tests while in agile including developers and stakeholders test the application.

- **Shortening Feedback Response Time:** In conventional SDLC, only during the acceptance testing, the Business team will get to know the product development, while in agile for each and every iteration, they are involved and continuous feedback shortens the feedback response time and cost involved in fixing is also less.
- **Clean Code:** Raised defects are fixed within the same iteration and thereby keeping the code clean

- **Reduce Test Documentation:** Instead of very lengthy documentation, agile testers use reusable checklist, focus on the essence of the test rather than the incidental details.
- **Test Driven:** In conventional methods, testing is performed after implementation while in agile testing, testing is done while implementation.

Agile design principles

- The Single-Responsibility Principle (SRP)
- The Open/Closed Principle (OCP)
- The Liskov Substitution Principle (LSP)
- The Dependency-Inversion Principle (DIP)
- The Interface Segregation Principle (ISP)

The Single-Responsibility Principle

- A class should have only one reason to change
  - If a class has more than one responsibility, the responsibilities can become coupled
    - changes to one can impact the other
  - If a class has a single responsibility, you can limit the impact of change with respect to that responsibility to this one class
- Example
  - Class Rectangle with draw() and area() methods
    - draw() is only used by GUI apps, separate these responsibilities into different classes

Open-Closed Principle

- A class should be open to extension but closed to modification
  - Allows clients to code to class without fear of later changes
- ***"Open for extensions."***
  - The behaviour of the code can be extended. When the requirements of the project are modified, the code can be extended by implementing the new requirements
- ***"Closed for modifications."***
  - The implementation of the new requirements does not need modifications on the already existing code

Liskov Substitution Principle

- Subclasses need to respect the behaviors defined by their superclasses
  - if they do, they can be used in any method that was expecting the superclass
  - Likov's Substitution Principle states that if a program module is using a Base class, then the reference to the Base class can be replaced with a Derived class without affecting the functionality of the program module.
  - New derived classes are extending the base classes without changing their behavior.

LSP -- Example

- If for each object **o1** of type **S**

  there is an object **o2** of type **T**

  such that for all programs **P** defined in terms of **T**, the behavior of **P** is unchanged when **o1** is substituted for **o2** then S is a subtype of T.

Dependency- Inversion Principle

- High-level modules should not depend on low-level modules;

  Both should depend on abstractions
- Abstractions should not depend on details.

  Details should depend on abstractions.
- In response to structured analysis and design, in which stepwise refinement leads to the opposite situation
  - high-level modules depend on lower-level modules to get their work done

- DIP attempts to "**invert**" the dependencies that result from a structured analysis and design approach
  - High-Level modules tend to contain important policy decisions and business rules related to an application; they contain the "identity" of an application
    - If they depend on low-level modules, changes in those modules can force the high-level modules to change!
    - High-level modules should not depend on low-level modules in any way

Interface Segregation Principle

- It states that clients should not be forced to implement interfaces they don't use. Instead of one fat interface many small interfaces are preferred based on groups of methods, each one serving one sub module.
- A class with a “fat” interface has groups of methods that each service different clients
  - This coupling is bad however, since a change to one group of methods may impact the clients of some other group

- This principle deals with the disadvantages of “fat” interfaces. Classes whose interfaces are not cohesive have “fat” interfaces.
- In other words, the interfaces of the class can be broken up into groups of methods. Each group serves a different set of clients. Thus, some clients use one group of methods, and other clients use the other groups.
- ISP acknowledges that there are objects that require noncohesive interfaces; however, it suggests that clients should not know about them as a single class. Instead, clients should know about abstract base classes that have cohesive interfaces.

Agile Testing Lifecycle

The agile testing lifecycle includes the following 5 phases:

1) Agile Testing Planning Meet up
2) Daily Scrums
3) Agility Review
4) Release Readiness
5) Impact analysis


Impact Assessment
Agility Test Planning
AGILE TESTING
Release Readiness
Daily Scrums
Agility Review Meeting

1. Agility test planning

All stakeholders come together to plan the schedule of testing process, meeting frequency and deliverables

2. Daily scrums

This includes the everyday standup morning meeting to catch up on the status of testing and set the goals for whole day

3. Agility Review

Weekly review meetings with stakeholders meet to review and assess the progress against milestones

4. Approval meetings for Deployment

At this stage we review the features that have been developed/ implemented are ready to go live or not

5. Impact Assessment

Gather inputs from stake holders and users, this will act as feedback for next deployment cycle

JUnit FRAMEWORK

- **JUnit** is an open source framework designed by Kent Beck, Erich Gamma for the purpose of writing and running test cases for java programs.
- In the case of web applications JUnit is used to test the application without server.
- This framework builds a relationship between development and testing process.
- **JUnit** is an **automated testing tool**

JUNIT FRAMEWORK

- JUnit is a unit testing framework designed for the Java programming language.
- JUnit has played an important role in the development of TDD frameworks.
- It is one of a family of unit testing frameworks which is collectively known as the **xUnit** that originated with **SUnit**.
  - **xUnit** is the collective name for several unit testing frameworks
  - **SUnit** is a unit testing framework for the programming language Smalltalk

Advantages of JUnit

- It’s used to test an existing class.
- Using JUnit we can save testing time.
- Using JUnit, If we want to test the application (for web applications) then server is not required so the testing becomes fast.

Unit Testing Tools

- C++ cppUnit
- .Net csUnit
- C Cunit
- Java Junit
- PHP PHPUnit
- Python PyUnit
- .Net Nunit
- Visual Basic VBUnit

Testing tools for java

1. Arquillian

- It is a highly innovative and extendible testing platform for JVM
- Easily create automated integration, functional, run test and acceptance tests for Java.

2. Jtest (Parasoft JTest)

- It is an automated testing and static analysis software made by Parasoft.
- Unit test-case generation and execution, static code analysis, data flow static analysis, and metrics analysis, regression testing, run-time error detection.

3. TestNG

- TestNG is a testing framework designed for the Java programming language and inspired by JUnit and NUnit.
- TestNG was primarily designed to cover a wider range of test categories such as unit, functional, end-to-end, integration, etc.

4. JWalk

- JWalk is designed as a unit testing toolkit for the Java programming language.

User stories

- seek to combine the strengths of written and verbal communication, where possible supported by a picture

- **Written requirements**
  - can be well thought through, reviewed and edited
  - provide a permanent record
  - are easily shared with groups of people
  - time consuming to produce
  - may be less relevant or superseded over time
  - can be easily misinterpreted

- **Verbal requirements**

  - instantaneous feedback and clarification
  - information-packed exchange
  - easier to clarify and gain common understanding
  - easily adapted to any new information known at the time
  - can spark ideas about problems and opportunities

What is a User Story?

- A concise, written description of a piece of functionality that will be valuable to a user (or owner) of the software.
- User stories are equally understandable by developers and customers
- Stories are:
  - User's needs
  - Product descriptions
  - Planning items
  - Tokens for a conversation
  - Mechanisms for deferring conversation

User Story Cards have 3 parts

1. **Description** - A written description of the user story for planning purposes and as a reminder

2. **Conversation** - A section for capturing further information about the user story and details of any conversations. Communication with team.

3. **Confirmation** - A section to convey what tests will be carried out to confirm the user story is complete and working as expected (acceptance test)

User Story Description

Steps:

- Start with a title.
- Add a concise description using the templates.
- Add other relevant notes, specifications, or sketches
- Before building software write acceptance criteria (how do we know when we're done?)

What is criteria for good story?

- **I**ndependent
- **N**egotiable
- **V**aluable to users or customers
- **E**stimatable
- **S**mall
- **T**estable

- **Independent** – User Stories should be as independent as possible. Dependent stories should not be done in same sprint
- **Negotiable** – a User Story is not a contract. It is not a detailed specification. It is a reminder of features for the team to discuss and collaborate to clarify the details near the time of development.
- **Valuable** – User Stories should be valuable to the user (or owner) of the solution. They should be written in user language. They should be features, not tasks.

- **Estimatable** – User Stories need to be possible to estimate. They need to provide enough information to estimate, without being too detailed.
- **Small** – User Stories should be small. Not too small and not too big.
- **Testable** – User Stories need to be worded in a way that is testable, i.e. not too subjective and to provide clear details of how the User Story will be tested.

Prioritize stories in a backlog

- Agile customers or product owner prioritize stories in a backlog
- A collection of stories for a software product is referred to as the **product backlog**
- The backlog is prioritized so that the most valuable items have the highest priorities
- **User Story Mapping** is an approach to Organize and Prioritize user stories

Exploratory testing

- Exploratory testing plays an important role in agile testing.
- It is a simultaneous approach where the testers learn about the system, perform test design and write test cases.
- Exploring the software to discover what the software does and doesn't do
- Exploratory testing means **the testing of software without any specific plans or schedules**.

Characteristics

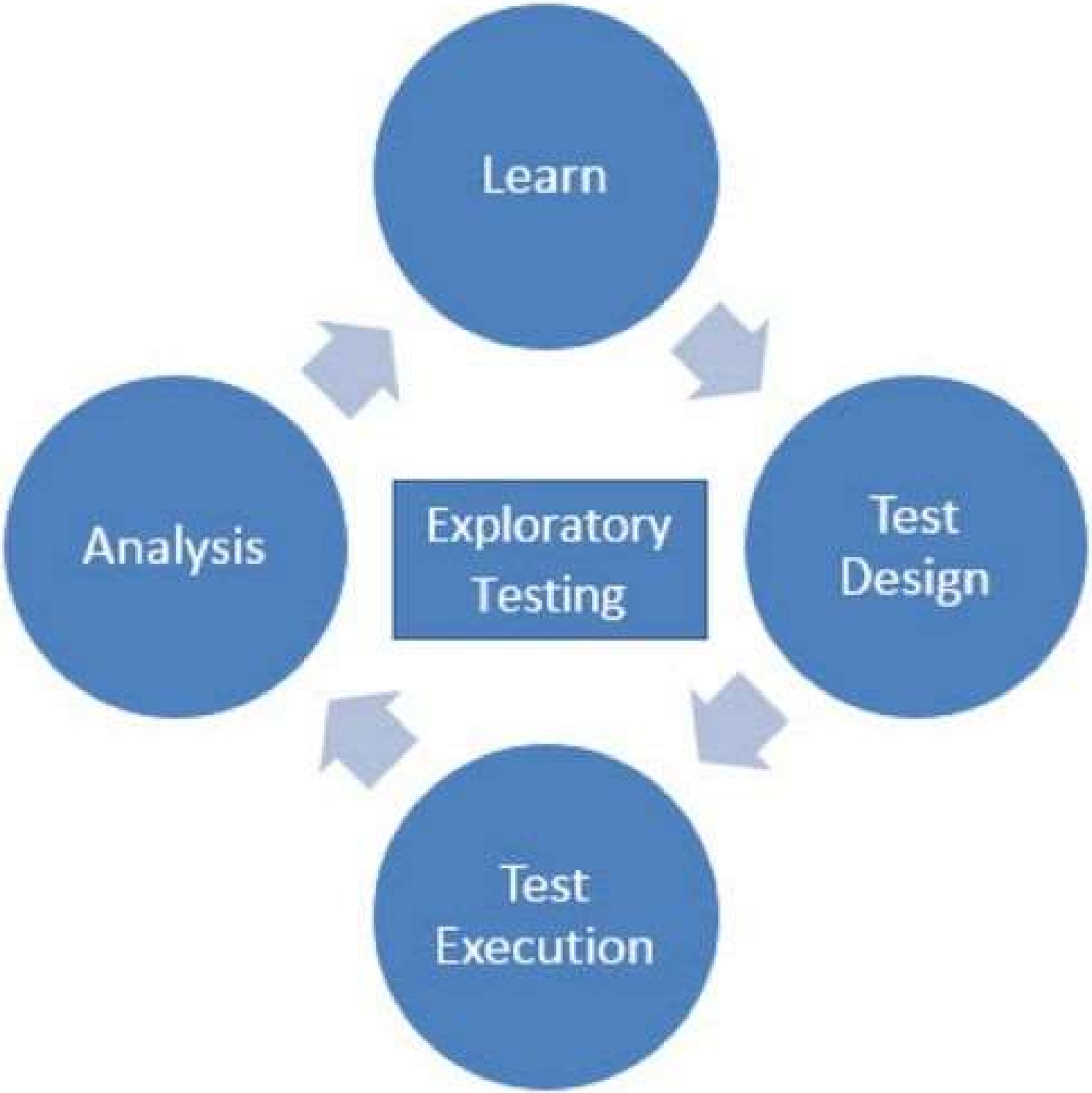
- application oriented
- planned and disciplined
- controllable and reliable
- risk minimizing
- Two sides of Exploratory Testing:
  - **for the tester**: freedom, flexibility, fun
  - **for the manager**: controllability, reliability, high quality

When to use Exploratory Testing?

- A common goal of exploration is the ***enquiry* for weak *areas*** of the program
- When there is little or **no specifications and requirements**
- When you have little or **no domain knowledge**
- When you **don't have time to specify**, script and test
- Exploratory Testing is extremely useful when faced with software that is
  - Untested
  - Unknown or
  - Unstable

How can we do Exploratory Testing?

- The process of exploratory testing is all about **discovery, investigation, and learning**.
- This emphasizes the personal freedom and responsibility of the individual tester.
- Test cases are not created in advance; instead, testers open the application and start browsing it to find defects.
- The focus of exploratory testing is more on testing as a “thinking” activity.
- Exploratory testing is a hands-on approach in which testers are involved in minimum planning and maximum test execution.


Learn
Test Design
Test Execution
Analysis
Exploratory Testing

Skills in Exploratory tester

- Knowledge of Test Design
- Observation Skills
- Critical Thinking
- Thinking Skills

Exploratory Testing Tools

Session Tester

- Session Tester is an exploratory testing tool for managing and recording Session-Based Testing.

Test Studio

- Test Studio helps to be more efficient in doing exploratory testing.

qTest Explorer

- qTest explorer is an intelligent capture technology.

Microsoft Test Manager (MTM)

- MTM helps you by records the actions a user performs as they work with their application.

REGRESSION TESTING

- Regression Testing is the **re-testing of a previously tested program** following modification to ensure that faults have not been introduced or uncovered as a result of the changes made

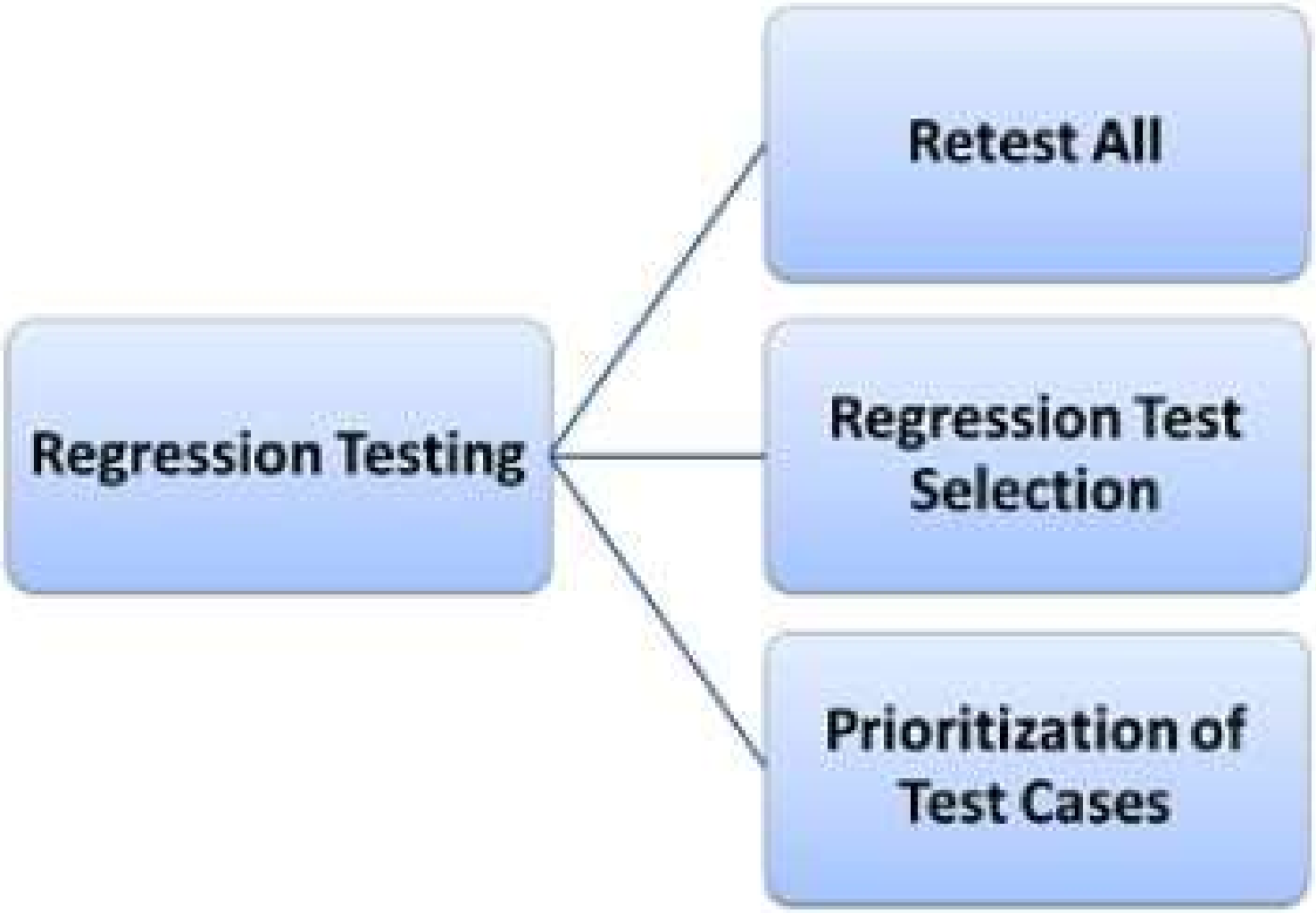
REGRESSION TESTING

- Regression Testing is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features.
- Regression Testing is nothing but full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.
- This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that old code still works once the new code changes are done.

Need of Regression Testing

- Change in requirements and code is modified according to the requirement
- New feature is added to the software
- Defect fixing
- Performance issue fix

Regression Testing Techniques



Retest All

- All the tests should be re-executed.
- This is very expensive as it requires huge time and resources.

Prioritization of Test Cases

- Prioritize the test cases depending on business impact, critical & frequently used functionalities.
- Selection of test cases based on priority will greatly reduce the regression test suite.

Regression Test Selection

- Instead of re-executing the entire test suite, it is better to select part of test suite to be run
- Test cases selected can be categorized as
  1) Reusable Test Cases
  2) Obsolete Test Cases
- Re-usable Test cases can be used in succeeding regression cycles.
- Obsolete Test Cases can't be used in succeeding cycles.

Regression Testing Tools

- **Selenium:** This is an open source tool used for automating web applications. Selenium can be used for browser based regression testing.
- **Quick Test Professional (QTP):** HP Quick Test Professional is automated software designed to automate functional and regression test cases. It uses VBScript language for automation. It is a Data driven, Keyword based tool.
- **Rational Functional Tester (RFT):** IBM's rational functional tester is a Java tool used to automate the test cases of software applications.