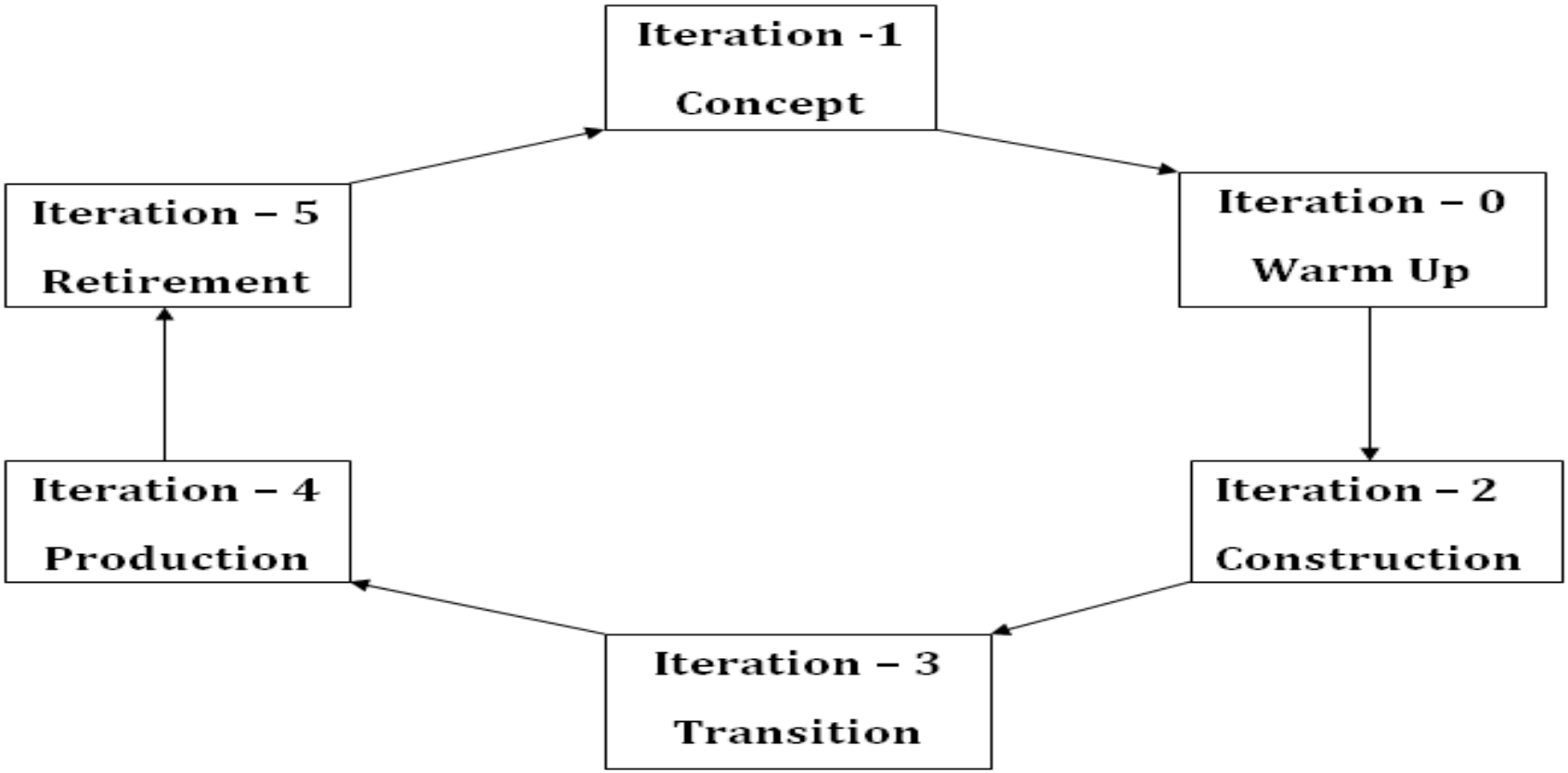
Unit - III

Agile Lifecycle



Agile Lifecycle

Iteration – 1 (Concept - Select the Project)

a. Identify Potential Project
b. Prioritize the Potential Project
c. Develop Initial Vision
d. Consider Project

Iteration - 0 (Warmup, Initiating the Project)

a. Active Stakeholder Participation

b. Obtain Funding & Support

c. Start Building the Team

d. Initial Requirement Envisioning

e. Initial Architectural Envisioning

f. Setup the Environment

Iteration - 2 (Construction Phase)

Known for delivering a working system which meets the changing needs of stakeholders.

a. Collaborative Development

b. Model Storming

c. Test Driven Designing

d. Confirmatory Testing

e. Evolution of Documentation (if necessary)

f. Internal Deployment of Software

Iteration - 3 (Transition Phase)

a. Active Stakeholders Participation

b. Final System Testing, Acceptance Testing, Pilot Testing

c. Finalize Supporting Documentation

d. Training of End User & Production Staff (if required)

e. Deployment of the system into production.

Iteration - 4 (Production – Operate & Support)

a. Operate the System

b. Support the System

c. Identify Defects & Enhancements

Iteration - 5 (Retirement – Removal from Production)

a. Remove the Final Version of the System

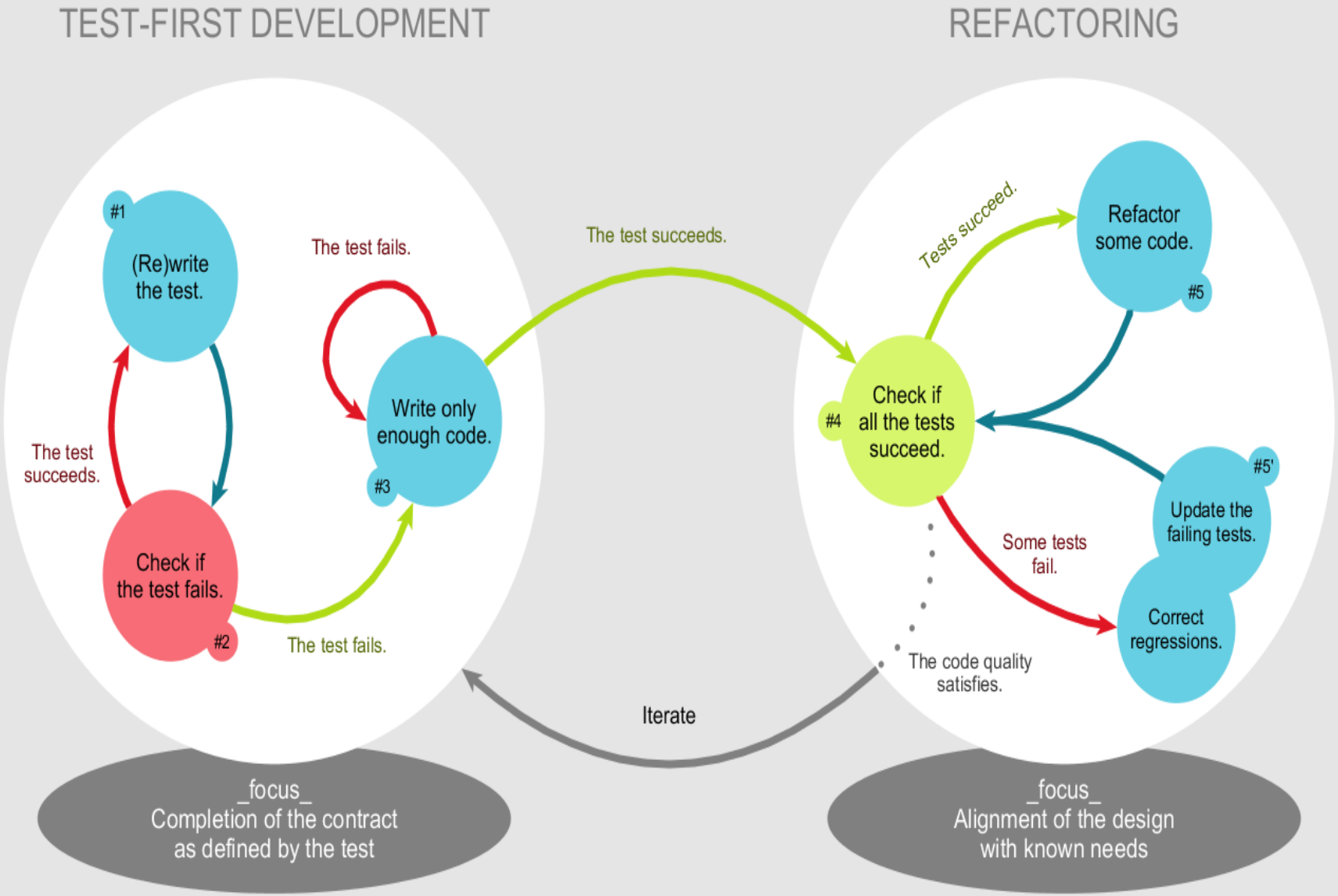
b. Data Conversion

c. Migrate Users

d. Update the Enterprise Model

Test Driven Development (TDD)

- Test-driven development is related to the test-first programming concepts of extreme programming .
- Software development process that relies on the repetition of a very short development cycle.
- Requirements are turned into very specific test cases, then the software is improved to pass the new tests, only.
- TDD encourages simple designs and inspires confidence.



Test Driven Development Cycle

- **Phase 1 - Add a test**

  In test-driven development, each new feature begins with writing a test. Write a test that defines a function or improvements of a function, which should be very succinct.

  To write a test, the developer must clearly understand the feature's specification and requirements. Can make use of use cases and user stories to cover the requirements

Test Driven Development Cycle

- **Phase 2 - Run all tests and see if the new test fails**

  This validates that the test harness is working correctly, shows that the new test does not pass without requiring new code because the required behavior already exists.

  The new test should fail for the expected reason. This step increases the developer's confidence in the new test.

Test Driven Development Cycle

- **Phase 3 - Write the code**

  The next step is to write some code that causes the test to pass. The new code written at this stage is not perfect and may, for example, pass the test in an inelegant way. That is acceptable because it will be improved in Phase 5.

  At this point, the only purpose of the written code is to pass the test. The programmer must not write code that is beyond the functionality that the test checks.

Test Driven Development Cycle

- **Phase 4 - Run tests**

  If all test cases now pass, the programmer can be confident that the new code meets the test requirements, and does not break or degrade any existing features.

  If they do not, the new code must be adjusted until they do.

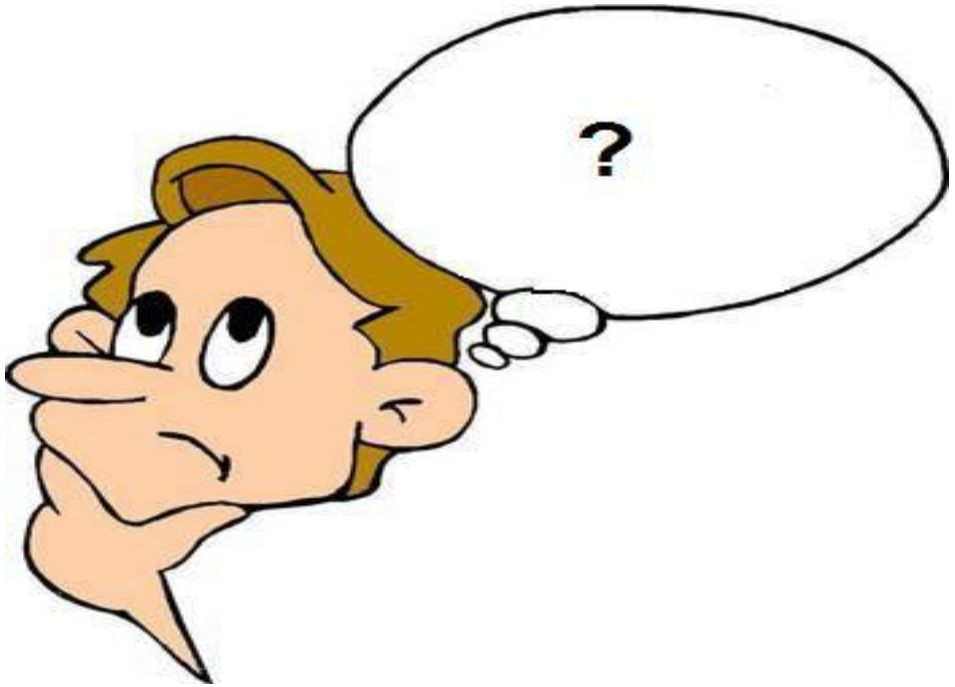
  How ???????

Test Driven Development Cycle

- **Phase 5 - Refactor code**

  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.

Enlist any five benefits and drawbacks of Test Driven Development.



Benefits & Drawbacks of TDD

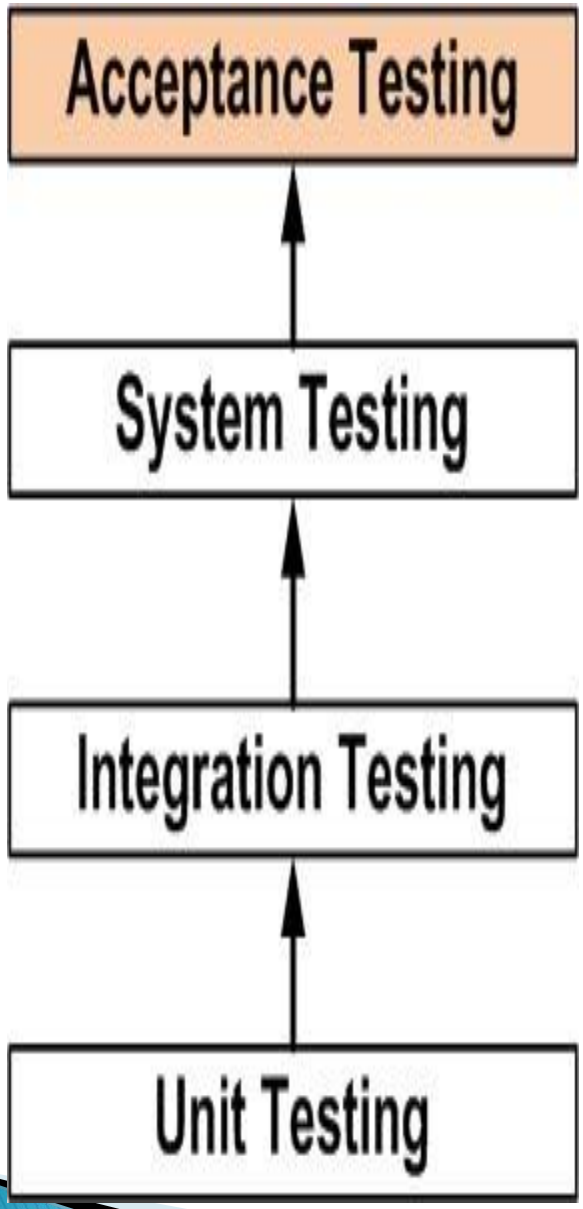
+ Code Coverage is good
+ Parallel Regression Testing is Performed
+ Simplified Debugging
+ System Documentation as Test Cases

- Complex to convert requirements to test cases
- Lot many testing cycles are involved
- Modification to existing code may be tough
- Use of TDD tools is only the game of experts.

Acceptance Testing

- An acceptance test is a formal description of the behavior of a software product, generally expressed as an example or a usage scenario.

- In many cases the aim is that it should be possible to automate the execution of tests by a software tool, either ad-hoc to the development team or off the shelf.



- **Acceptance Testing** is a level of the software testing where a system is tested for acceptability.

- The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

Types of Acceptance Testing

- ***Internal Acceptance Testing*** (Also known as Alpha Testing) is performed by members of the organization that developed the software but who are not directly involved in the project (Development or Testing). Usually, it is the members of Product Management, Sales and/or Customer Support.

- ***External Acceptance Testing*** is performed by people who are not employees of the organization that developed the software.

1. ***Customer Acceptance Testing*** is performed by the customers of the organization that developed the software. They are the ones who asked the organization to develop the software. [This is in the case of the software not being owned by the organization that developed it.
2. ***User Acceptance Testing*** (Also known as Beta Testing) is performed by the end users of the software. They can be the customers themselves or the customers' customers.

Benefits

- Encourages closer collaboration between developers on the one hand and customers, users or domain experts on the other, as they entail that business requirements should be expressed

- Provide a clear and unambiguous "contract" between customers and developers; a product which passes acceptance tests will be considered adequate.

- Minimizes the chance and severity both of new defects and regressions (defects impairing functionality previously reviewed and declared acceptable)

Agile Testing Lifecycle

Exploratory Testing

- Exploratory testing is about exploring, finding out about the software –
  - what it does ?
  - what it doesn't do ?
  - what works ?
  - what doesn't work ?

  The tester is constantly making decisions about what to test next and where to spend the (limited) time.

- Exploratory testing is a hands-on approach in which testers are involved in minimum planning and maximum test execution.
- The planning involves the creation of a test charter, a short declaration of the scope of a short (1 to 2 hour) time-boxed test effort, the objectives and possible approaches to be used.
- The test design and test execution activities are performed in parallel typically without formally documenting the test conditions, test cases or test scripts.

- Test logging is undertaken as test execution is performed, documenting the key aspects of what is tested, any defects found and any thoughts about possible further testing.

- It can also serve to complement other, more formal testing, helping to establish greater confidence in the software.

Regression Testing

- Regression testing is a type of software testing which verifies that software, which was previously developed and tested, still performs correctly after it was changed or interfaced with other software.
- Changes may include software enhancements, patches, configuration changes, etc.
- The purpose of regression testing is to ensure that changes such as those mentioned above have not introduced new faults.

- One of the main reasons for regression testing is to determine whether a change in one part of the software affects other parts of the software.
- Common methods of regression testing include re-running previously completed tests and checking whether program behavior has changed and whether previously fixed faults have re-emerged.
- Regression testing can be performed to test a system efficiently by systematically selecting the appropriate minimum set of tests needed to adequately cover a particular change.

The various regression testing techniques are:

- **Retest all**

  This technique checks all the test cases on the current program to check its integrity. Though it is expensive as it needs to re-run all the cases, it ensures that there are no errors because of the modified code.

- **Regression test selection**

  Unlike Retest all, This technique runs a part of the test suite (owing to the cost of retest all) if cost of selecting the part of test suite is less than retest all technique.

- **Test case prioritization**

  Prioritize the test cases so as to increase a test suite's rate of fault detection. Test case prioritization techniques schedule test cases so that the test cases that are higher in priority are executed before the test cases that have a lesser priority.

- **Hybrid**

  This technique is a hybrid of Regression Test Selection and Test Case Prioritization.

Enlist any two drawbacks & two benefits of Regression Testing.

Benefits & Drawbacks

+ It provides certainty that the changes made to the software have not affected the existing functionalities.

+It is performed to test a system efficiently by systematically selecting the appropriate minimum set of tests.

-Lot of unnecessary overheads

-Regression testing tools is not equipped to handle database application

Risk Based Testing

- A risk is the probability of occurrence of any failure.
- It is the probability that an undetected software bug may have a negative impact on the user of a system.
- No. of Risks may vary during the development and hence cannot be pre assumed and finalized.

Types of Risks –

- **Business or Operational**

1. High use of a subsystem, function or feature
2. Criticality of a subsystem, function or feature, including the cost of failure

- **Technical**

1. Geographic distribution of development team
2. Complexity of a subsystem or function

- **External**

1. Sponsor or executive preference
2. Regulatory requirements

- **E-business failure-mode related**

1. Static content defects
2. Web page integration defects
3. Functional behavior-related failure
4. Service (Availability and Performance) related failure
5. Usability and Accessibility-related failure
6. Security vulnerability
7. Large scale integration failure

Risk Handling

Following are the activities for risk handling :

1. Risk Identification
2. Risk Analysis
3. Risk Mitigation
4. Risk Neutralization
5. Risk Removal
6. Risk Assessment & Retrospection

- **Risk-based testing** (RBT) is a type of software testing that functions as an organizational principle used to prioritize the tests of features and functions in software, based on the risk of failure, the function of their importance and likelihood or impact of failure.

- Risk-based testing uses risk (re-)assessments to steer all phases of the test process, i.e., test planning, test design, test implementation, test execution and test evaluation.

Agile Test Automation

Agile Test Automation is the application of agile development principle to test automation problem.

Test Automation means tools to support all aspects of a test project not just the test execution.

Test Automation is directed by the testers, internals or externals.

Test Automation progresses when supported by dedicators (Tool Smiths).

- Test tool smiths advocates for testability features and produce tools to exploit those features.
- They gather & apply a wide variety of tools to support testing.
- Test Automation is organized to fulfill short term goals.
- Long term test automation task requires a compelling business case (model).
- With these test automation it is expected that they respond quickly to requests for assistance from testers.
- They seek out test productivity problems.

- Investigates all possible solutions in concern with the tester.
- Apply technology to improve the test process.
- Advocates for specific testability features in product.
- Research available tools & learn how to use them.
- Gather tools that developers & testers produce and review upcoming product plans to assess automation possibility.

Agile ALM
(Application Lifecycle Management)

- Agile Application Lifecycle Management (Agile ALM) is a central platform that allows teams using Agile methods, alone or in combination with other development methodologies (e.g., RUP, waterfall), to manage all phases of the software development lifecycle from requirements through release.

- By uniting business users and developers and providing cross-team visibility, Agile ALM enables organizations to achieve a faster time to market and higher quality software releases while reducing infrastructure costs.

- Application Lifecycle Management (ALM) is the management of the software application lifecycle from initial development to final release.

- ALM encompasses all of the practices, processes, and tools that aid in managing an application's lifecycle from both a business and development perspective. Key capabilities of an ALM platform include the ability to handle change management, workflow, source code management, task management, testing and bug tracking, lab management, reporting and analytics.

- To facilitate seamless distributed development, an ALM platform should also include a central repository “in the cloud” for managing all of the various types of content created (i.e. code, tasks, roles, requirements, and other artifacts) as well as a system for establishing traceability and accountability across the ALM platform's many processes, locations, and tool types.

Proper implementation of Agile ALM leads to:

- Increase in productivity
- Reduction of cost
- Team collaboration
- Removal of redundancy
- Practical approach of coding and deployment
- Makes software development fun

Roles in Agile Project

An agile project comprise of various roles–

1. Product Owner

2. Project Manager

3. Product Developer

4. Architects (Optional)

5. Quality Assurance Experts

Role of Product Owner

- Responsible for market, business case, and competitive analysis
- Responsible for long and short term product vision
- Responsible for ROI and Net Profit
- Prioritizes features for releases based upon expected ROI
- Writes Acceptance Criteria
- Writes user stories
- Makes tradeoff decisions between scope and schedule.

Role of Project Manager

- Manages planning process
- Manages overall program schedule
- Drives multiple releases/projects
- Facilitates Release Planning & Retrospective
- Provides access to tools and people
- Owns all action items for the project until he/she finds the right owner

- Owns reporting on project status, to all directions
- Coordinates other release support
- Responsible for risk assessment & mitigation
- The Role is a peer to the Product Manager and the Engineering Manager on the release/project
- Educates/Enforces agreed upon processes & methodology rules
- Educates/Enforces roles and responsibilities

Role of Product Developer

- Estimates size of backlog items
- Translation of backlog items into engineering design and logical units of work (tasks)
- Evaluation of technical feasibility
- Implementation of backlog items
- Writes unit tests first to the contract of the interface and other abstract classes
- Writes and verifies code which adheres to the acceptance criteria

Roles of Architects

- Responsible for end-to-end cross functional system design and communication
- Works with the PM to group features based upon the Architectural Elements which support them, an influence on priorities
- Tests Architectural Elements with executable and testable design (abstract interfaces, aka the contract)
- Facilitates technical decision; incorporates feedback and emergent patterns from the team back in to the overall design

- Produces alternate Design Concepts & detailed approach
- Ensures the Design goals – Performance, Modularity, Reliability, Maintainability, Reusability, Internationalization and Accessibility – are met
- Ensures technical cohesion and helps write the technical contract in interfaces and other abstract objects and data entities
- Leads design review & provides feedback

Roles of QA Experts

- Writes test plans which enforce the acceptance criteria of features
- Keeps all test plans and cases updated to changing requirements
- Continually integrates the code base with automated builds and functional-level regression tests
- Notifies when production is blocked due to errors in development
- Measuring Quality

- Defining Quality
- Improving Quality
- Enforces QA Best Practices
- Provides visibility into end-to-end product quality
- Owns QA Health status of releases/projects

Agile in Distributed Teams (De Agile)

Agile development in distributed environment may lead to various issues :

- Communication
- Language & Culture
- Time Zone
- Progress Tracking
- Workload Distribution
- Misunderstanding the Target
- Continuous Integration

Agile in Distributed Teams (De Agile)

- When the team is geographically distributed communication can sometimes become a challenge.
- It's important to anticipate this and address it proactively. Distributed teams require strong leadership.
- Leaders of distributed teams should facilitate optimal levels of communication among the members of the team.

- Team members should also be encouraged and empowered to communicate actively.

- Removing hierarchical organizational barriers further multiples the flow of information within the project.

- Open and active communication increases the overall success and velocity of a Distributed Agile team.

Balancing Agility

- Started with a self-assessment
- Increased pace of change and need for agility
- Software dependability and need for discipline
- Ability to quickly satisfy stakeholders' evolving value props
- Increasing gap between supply & demand
- Ability to cope with existing and emerging technical challenges

Steps for Balancing Agility

- Graph your organization's "typical project"
- Identify major needs for organizational change

  –With respect to balancing agility and discipline

  –With respect to future trends and environmental risks
- Identify most critical first steps for improvement

- Identify pilot project for applying first steps

  –Application, staffing

  –Needs for education, teambuilding, culture change

- Summarize in bulleted strategic plan

- Self-assessment is recommended for your organization

  –Locate your place in the home ground space

  –Identify areas of change and how they might move your location

  –Look for ways to integrate methods to better respond to change

Agile Projects on Cloud

Cloud computing and virtualization make it easy for agile development teams to seamlessly combine multiple development, test and production environments with other cloud services.

Here are six important ways cloud computing and virtualization enhance agile software development.

1. **Cloud Computing Provides an Unlimited Number of Testing and Staging Servers.**

When agile development is used without virtualization or clouds, development teams are limited to one physical server per development, staging and production server need. However, when virtual machines or cloud instances are used, development teams have practically an unlimited number of servers available to them. They do not need to wait for physical servers to become free to begin or continue their work.

2. **It Turns Agile Development Into a Truly Parallel Activity.**

You may use agile development but still experience delays in provisioning server instances and in installing necessary underlying platforms such as database software.

How-To: Keep Cloud Projects Agile—and Simple ?

Although an agile methodology aims to squeeze all the inefficiencies and delays out of software development, in practice it becomes a serial activity. Cloud computing can push it toward becoming a more parallel activity than it once was. This leads to more efficient, more effective and better utilized agile software development teams.

3. **It Encourages Innovation and Experimentation.**

Being able to spawn as many instances as needed enables agile development groups to innovate. If a feature or a story looks interesting, a team can spawn a development instance quickly to code it and test it out. There's no need to wait for the next build or release, as is the case when a limited number of physical servers are available. When adding cloud computing to agile development, builds are faster and less painful, which encourages experimentation.

4. **It Enhances Continuous Integration and Delivery.**

As stated, cloud instances and virtualization greatly enhance continuous integration and delivery. Builds and automated tests take time. Agile development groups may need to subsequently fix the code for tests that fail during the automated testing—and they need to do this again and again until the build passes all the tests.

5**. It Makes More Development Platforms and External Services Available.**

Agile development groups may need to use a variety of project management, issue management, and, if continuous integration is used, automated testing environments. A number of these services are available as Software as a Service (SaaS) offerings in the cloud.

- Agile development can use a combination of virtualization, private clouds and the public cloud at the Infrastructure as a Service (IaaS) level. Such offerings include Amazon Web Services, GoGrid, OpSource andRackSpace Cloud.
- Then comes the use of Platform as a Service (PaaS) instances such as the Oracle Database Cloud Service, the Google App Engine and the Salesforce.com platformforce.com, all of which include databases and language environments as services.

- Finally, there are a number of SaaS services that specifically assist agile development, including Saleforce.com, the Basecamp project management portal and TestFlight, which provides hosted testing automation for Apple iOS devices.

6**. It Eases Code Branching and Merging.**

In theory, agile development assumes that all features can be broken down into stories of uniform size and slated for builds. In practice, agile projects may encounter features whose development efforts last longer than a build or even a release. In code refactoring efforts, current releases may need to be enhanced with minor enhancements and used in production, all while a major redesign of code is going on.

Code branching is necessary in these cases. Code branching and merging involves juggling many versions of development and staging builds. With virtualization and cloud computing, you can avoid buying or renting additional physical servers for these purposes.