Software Quality Assurance

Module 10

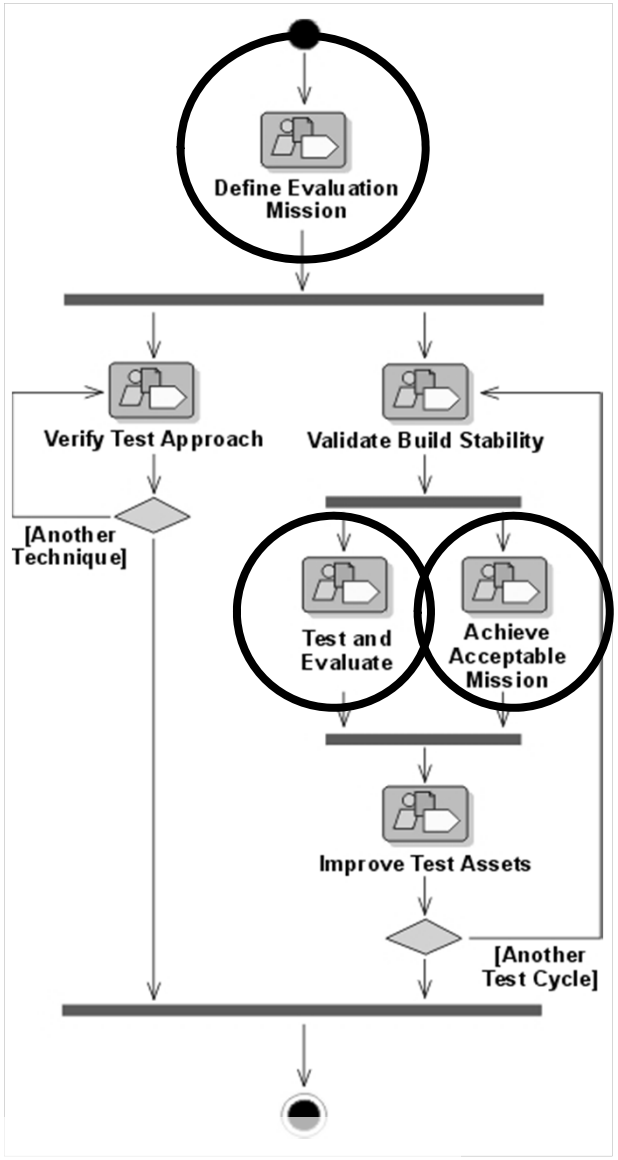
The RUP Workflow As Context

Objectives

- Identify the remaining workflow details of the RUP Test Discipline.
- Identify some additional key practices for successful software testing.
- Understand the testing workflow in the context of an Iteration

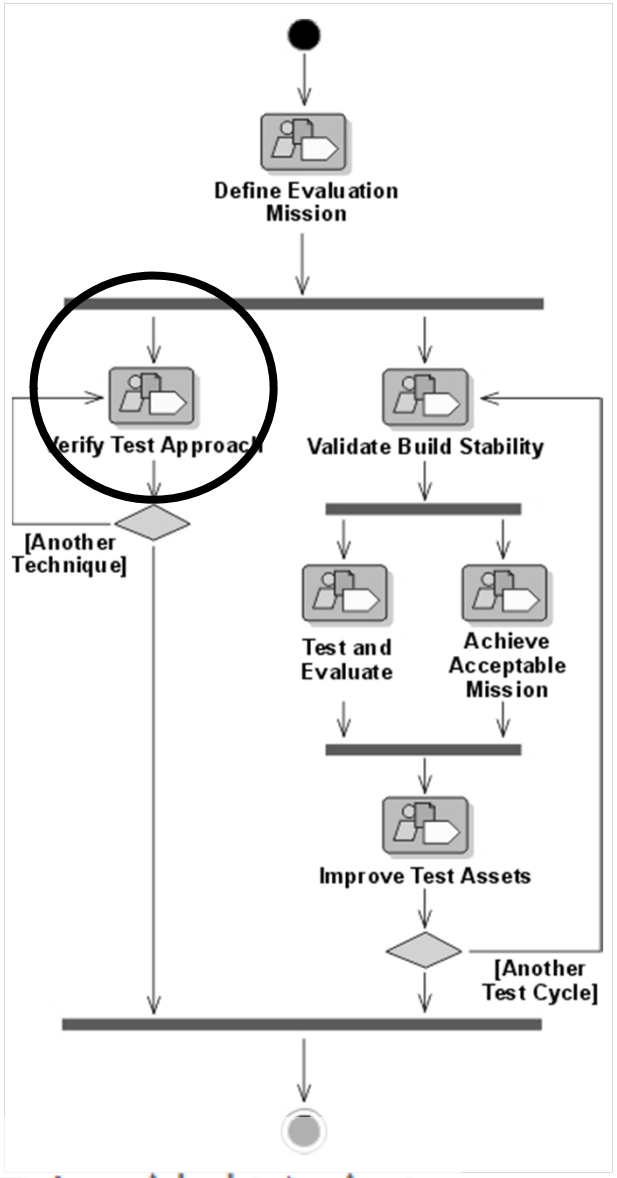
Review: Where We've Been

- Quick Review of RUP Workflow Details
- So far, we have covered:
  - Define Evaluation Mission
  - Test and Evaluate
  - Achieve Acceptable Mission



Verify Test Approach

- Other Workflow Details
  - ➔**Verify Test Approach**
  - Validate Build Stability
  - Improve Test Assets
- Review Test Workflow



Verify Test Approach

- The purpose of this workflow detail :
  - To achieve appropriate breadth and depth of the test effort to enable a sufficient evaluation of the Target Test Items — where sufficient evaluation is governed by the Test Motivators and Evaluation Mission.

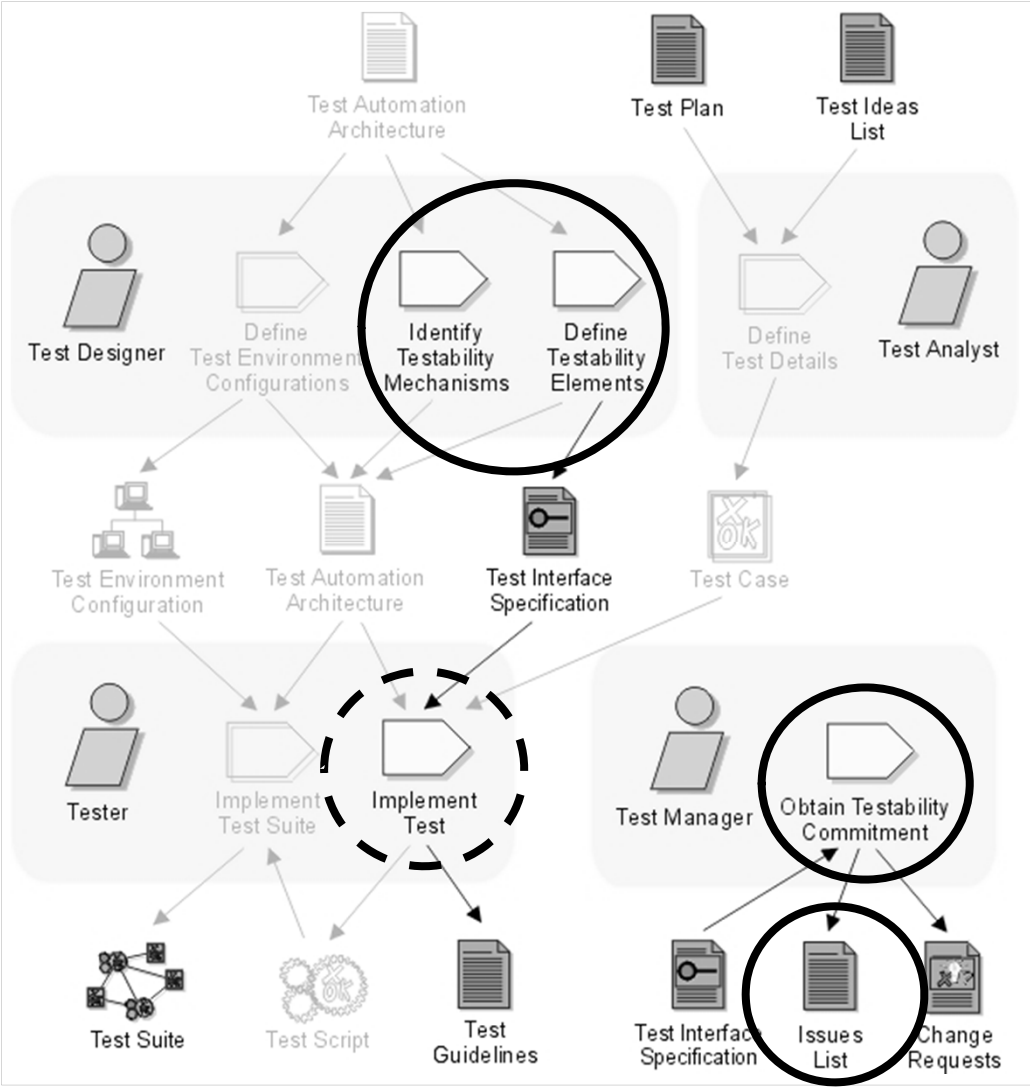
- For each iteration, this work focuses on:
  - Early verification that Test Approach will work
  - Establishing the supporting infrastructure
  - Obtaining the required testability
  - Identifying the scope, boundaries, limitations and constraints of each technique

Verify Test Approach - Content Outline

- Overall focus is on the workflow detail: Verify Test Approach
  - Definition of the workflow detail
    - Brief overview of activities and artifacts typical of the work
  - Checking whether your approach is workable
  - Considerations for testability

Verify Test Approach – Activities and Artifacts

- This section focuses on proving that the approach you plan to use is workable.
- We will spend time looking at activities that support the concept of *Testability.*
- Note that it is a good practice to implement tests as concrete proof that the approach will work.



Verify Test Approach - Purpose

- Will the approach work and produce accurate results?
- Will it fit the project constraints?
- Do the techniques give us adequate coverage?
- What risks remain?

Discussion Exercise 10.1: Testing Project Risks

- What are the most common risks in testing projects?
- What do you do about them?

Checking Whether Your Approach is Workable

- Risk analysis is a fundamental activity in RUP
- Analyze the risks in your plan for testing
- Ask how workable your test approach will be
  - There are an infinite number of tests for your program
    - What tests are you implicitly choosing not to run and what issues will you miss?
  - How much time can you spend not immediately finding defects?
    - Project management, training, documentation, …
  - If you skimp on those other tasks…
    - How much can you learn and improve?
    - How can complex tasks succeed?

Improving Testability

- Testability involves
  - **Visibility** – the tester can see (and understand) what is going on
  - **Control** – the tester can force something to happen.
- Systems don't magically become testable. Testability is designed in (or not).
  - Testability features are built into a program, and should be discussed when product requirements are discussed.
  - It's up to you to ask for them.

Improving Testability: Examples

- Visibility
- Component-based architecture
- Instrumentation for tracing and profiling

Improving Testability: Examples

- Visibility
  - Use standard UI controls, to facilitate GUI-level regression testing
  - Unique error information for every error message. Let the tester put the program into a chatty (e.g. debug) mode. An error message reports the function that generated the error, plus additional information about the type of internal test that was failed, the variable that failed it, and any notes from the programmer about what this might mean.
  - Trace logs

Improving Testability: Examples

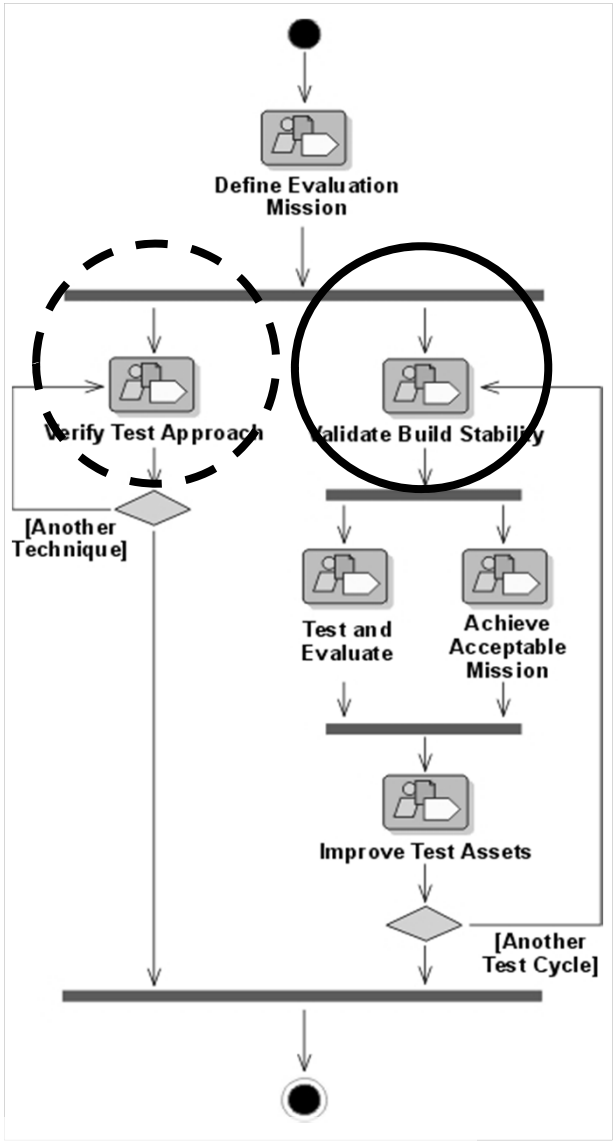
- Component-based architecture
  - API layer, beneath the user interface, that gives the tester programmatic control over anything that the UI exposes to the customer. For example, if the UI has a feature that lets the customer add two numbers, the API would give access to the same additional call. The UI may change (location of the dialog, visual design, language, etc.) but the API will change much less often.
  - Exposing interface contracts
  - This lets the tester create automated tests of the underlying logic of the program, which will need less maintenance, and which will probably survive longer.

Exercise 10.2: Improving Testability

1. Pick a product
2. Form project teams
   a. Each team takes a different part of a product
   b. Be specific about the part of a product
3. List 10 useful testability features
   a. 5 for visibility
   b. 5 for control
4. For each, answer:
   a. Is it available today?
   b. If so, do you use it?
   c. When would you have to request this to get it?
   d. What would it take (work / cost) to get it?

Validate Build Stability

- **Other Workflow Details**
  - Verify Test Approach
  - **➔Validate Build Stability**
  - Improve Test Assets
- Review Test Workflow



Validate Build Stability

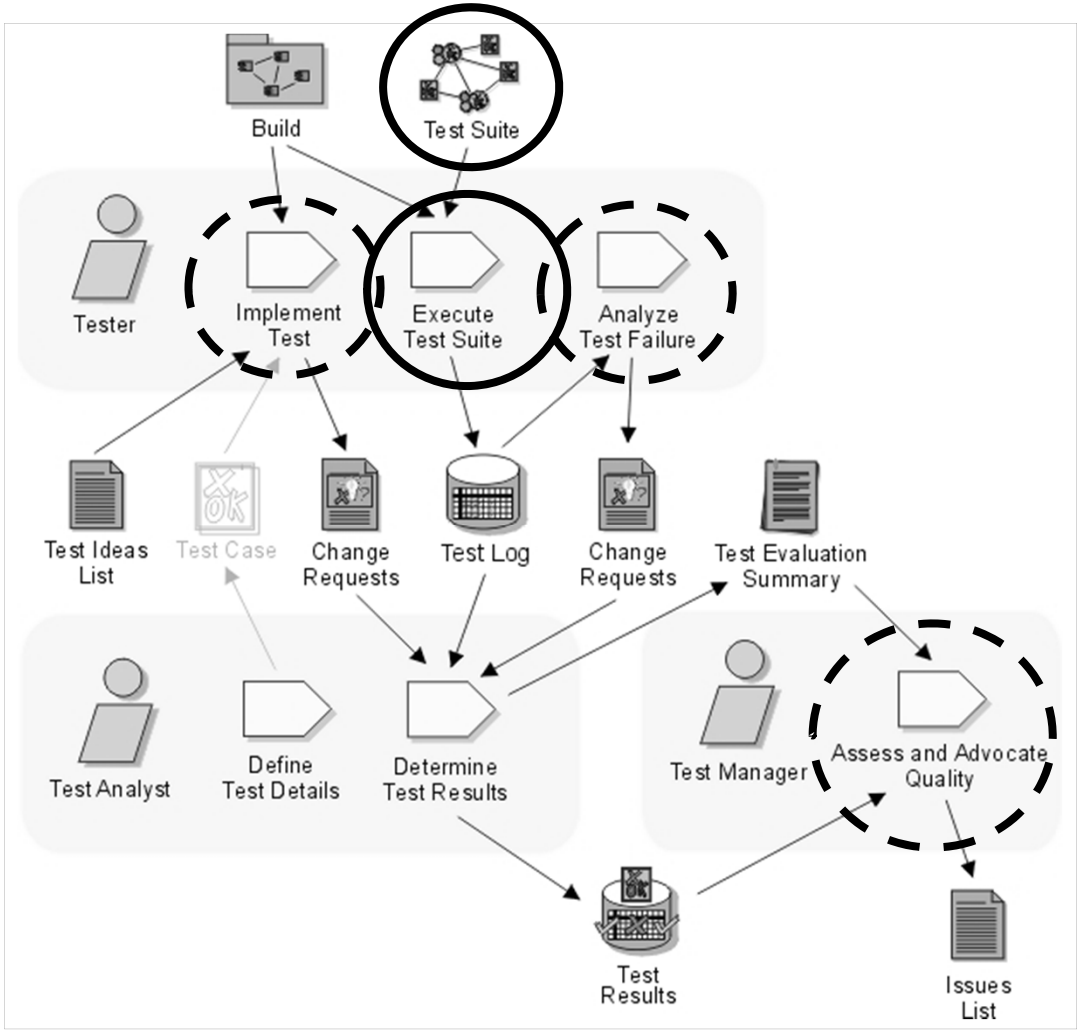
- The purpose of this workflow detail :
  - To Validate that the build can be tested / evaluated. This helps to prevent wasted testing effort.
- For each build to be tested, this work focuses on:
  - Assessing the stability and testability of the build
  - Confirming the expectation of the development work delivered in the build
  - Deciding to accept the build as worth further testing, guided by the evaluation mission
  - If new build is rejected, current build is still used
- This work is also referred to as a *smoke test*, *build verification test*, *build regression test*, *sanity check* or *acceptance into testing*.

Validate Build Stability - Content Outline

- Overall focus is on the workflow detail: *Validate Build Stability*
  - Definition of the workflow detail
    - Brief overview of activities and artifacts typical of the work
  - Why build verification testing is so important
  - The scope of build verification tests
  - Automation

Validate Build Stability – Activities and Artifacts

- This section focuses on *Build Verification Tests* (BVT's)
- We will discuss some key issues associated with BVT's including deciding on appropriate *Test Suites.*
- A BVT is specialized Test and Evaluate work, so you will notice many common elements.



Why Build Verification Tests are So Important (1/2)

- Frequent builds are a valuable risk management activity
  - Errors detected early
  - Errors traceable quickly to a small number of changes
  - Design changes reviewed quickly
  - As the project nears completion, more people depend on details of the user interface and functionality of the program. They need to know about changes (and to review them) as soon as possible.

Why Build Verification Tests are So Important (2/2)

- Bad builds can waste an enormous amount of testing time
- Build verification is part of the configuration management process. It prevents versions
  - Delivered with wrong components
  - With key test-blocking errors, or without critical (agreed) fixes to errors, or without key features
  - That are less efficiently testable than planned

Organizational Considerations

- Who should run the BVT?
  - Programmers? Testers? Configuration mgmt technicians? Release engineering team?
- Who should create and maintain the BVT?
- Group independence matters less for BVT
  - Structure the BVT so that any authorized person can run it, from any group
  - Allow different groups to add cases to the BVT, but leave pruning to the test group

Automation of BVTs and Build Process

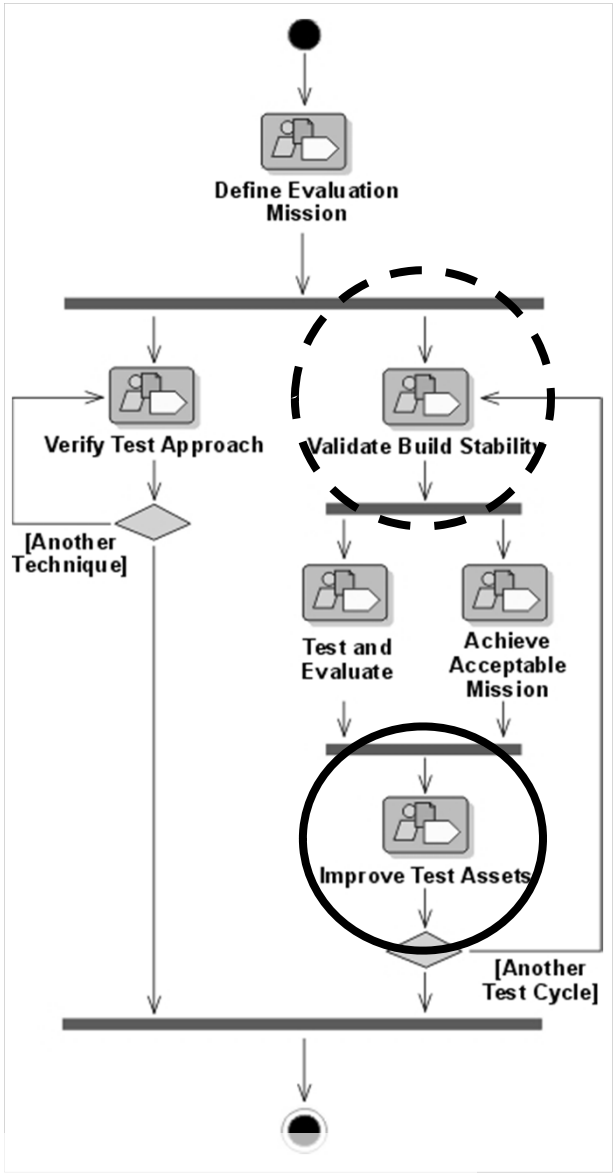
- Most of the BVTs should be automated
  - Rapid ROI (Return On Investment) : you run the tests every build anyway
- BVTs allow an automated build-verify-deploy sequence
  - Maintain / refactor BVTs to prevent false negatives
- Mixed BVTs often desirable
  - Primarily automated, extended with
  - Manual testing for short term issues
    - Build-critical bugs are fixed
    - Specific work items delivered

Discussion Exercise 10.3: Build Verification Tests

- Describe your build process
- Describe your build verification testing
  - How successful or unsuccessful is your BVT?
- What would you change?

Improve Test Assets

- **Other Workflow Details**
  - Verify Test Approach
  - Validate Build Stability
  - ➔**Improve Test Assets**
- Review Test Workflow



Improve Test Assets

- The purpose of this workflow detail :
  - To maintain and improve the test assets. This is important especially if the intention is to reuse the assets developed in the current test cycle in subsequent test cycles.

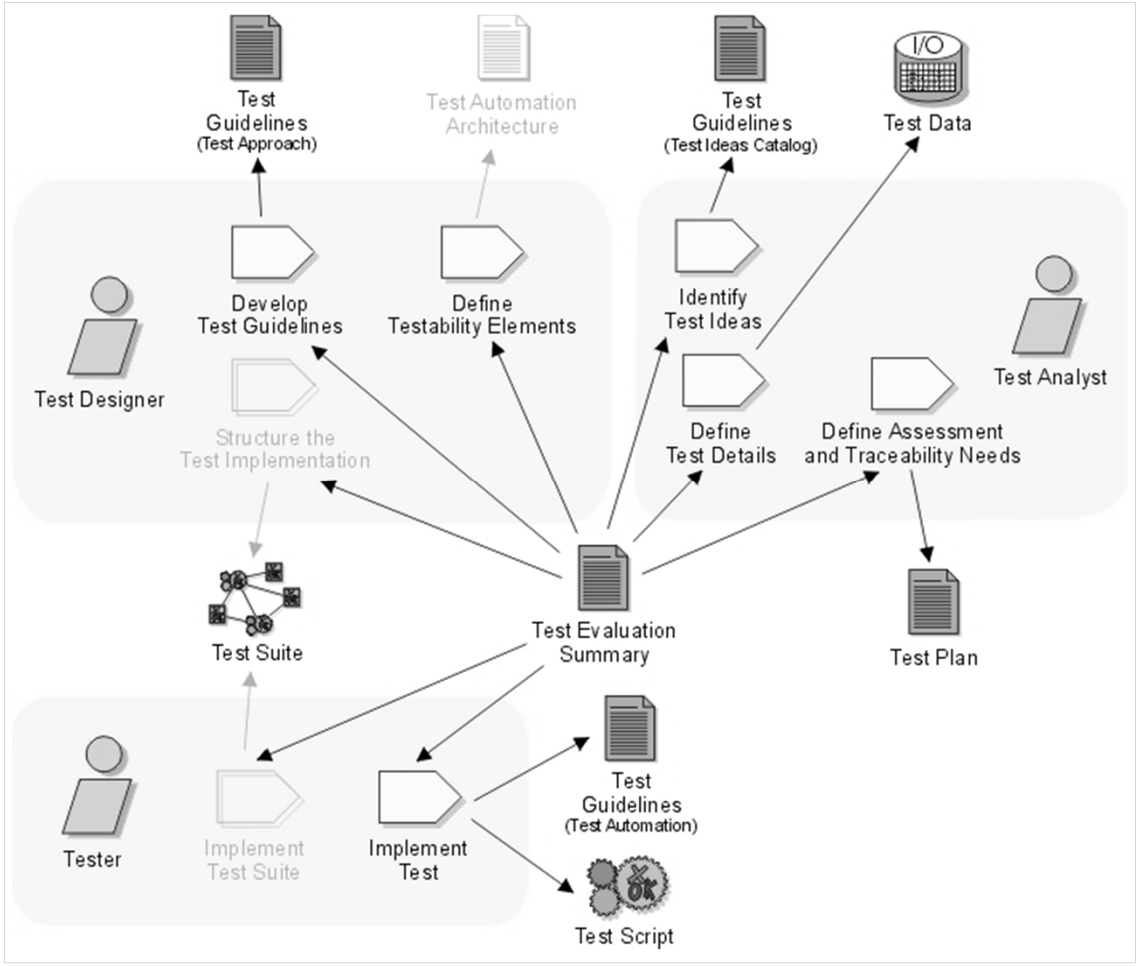
- For each test cycle, this work is focused mainly on:
  - Assembling Test Scripts into additional appropriate Test Suites
  - Removing test assets that no longer serve a useful purpose or have become uneconomic to maintain
  - Maintaining Test Environment Configurations and Test Data sets
  - Exploring opportunities for reuse and productivity improvements
  - Conducting general maintenance of and making improvements to the maintainability of test automation assets
  - Documenting lessons learned—both good and bad practices discovered during the test cycle

Improve Test Assets - Content Outline

- Overall focus is on the workflow detail: Improve Test Assets
  - Definition of the workflow detail
    - Brief note of activities and artifacts typical of the work
  - Considerations for Improving Test Assets
  - Maintenance Costs
  - Artifacts you might consider improving

Improve Test Assets – Activities and Artifacts

- This section focuses on making sure regular improvements are made to the test effort.
- We will discuss some of the key concerns in making ongoing improvements to test assets and activities.



Considerations for Improving Test Assets

- Point of this workflow detail is simple and direct.
  - You make a large investment in test ideas, test cases and scripts (whether manual or automated), test documentation, tester training materials, and so on.
  - These are your long term assets.
  - It is important to protect and increase their value.
- Every iteration in every project involves change.
  - Changes may make parts of some assets outdated.
  - Change is inevitable, so plan a maintenance strategy.
  - If you cannot maintain your assets efficiently, you risk losing their value over time.

Discussion Exercise 10.4: Cost Considerations

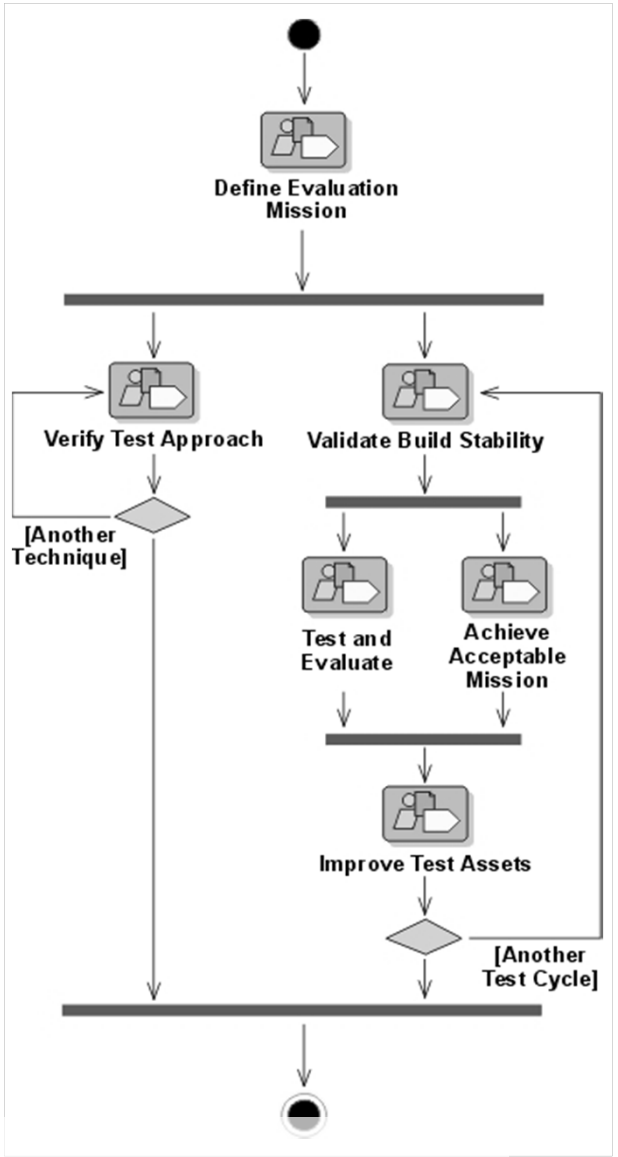
- What Assets need to be maintained?
  - Test Scripts?
  - Test Suites?
  - Test Cases?
  - Test-Ideas Lists?
  - Change Requests?
  - Test-Idea Catalogs?
  - Automation Frameworks?
  - Others?
- Can you afford to maintain them all?

Workbook Page: Artifacts To Consider Improving

- Test guidelines
  - Test approach
  - Test ideas catalog
  - Test automation
- Test data
- Test script
- Test suite
- Test automation architecture
- Test environment configuration
- Test plan
- Test evaluation summary

Module 10 - Content Outline (Agenda)

- Other Workflow Details
  - Verify Test Approach
  - Validate Build Stability
  - Improve Test Assets
- **Review Test Workflow**



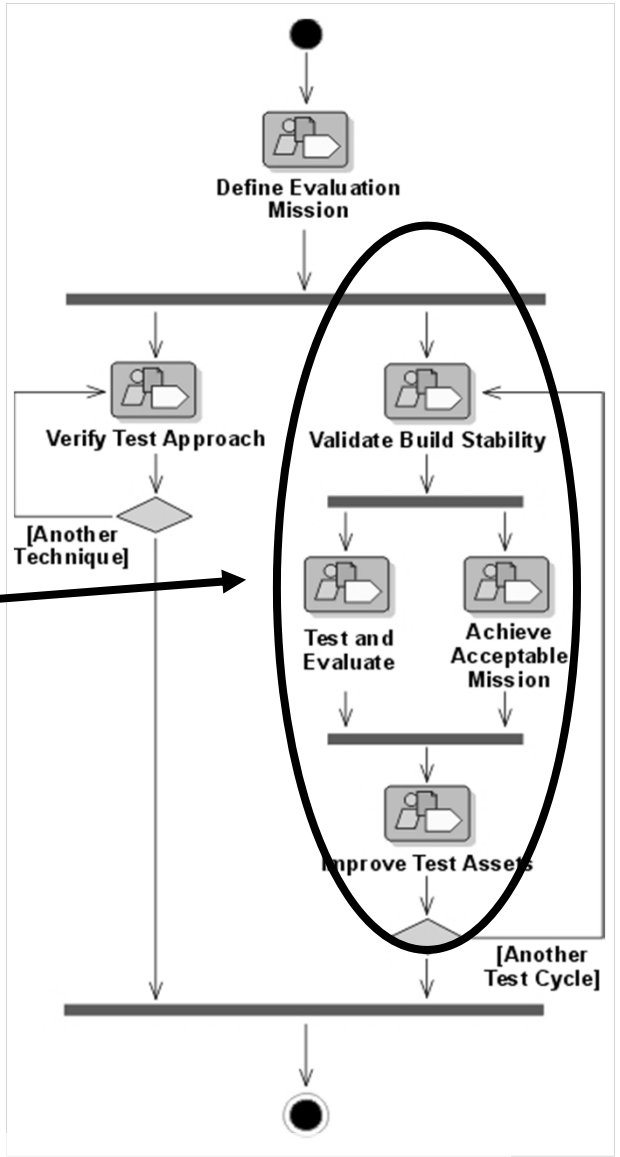
Review: Iterations

Inception	Elaboration		Construction			Transition	
Preliminary Iteration	Architect. Iteration	Architect. Iteration	Devel. Iteration	Devel. Iteration	Devel. Iteration	Transition Iteration	Transition Iteration

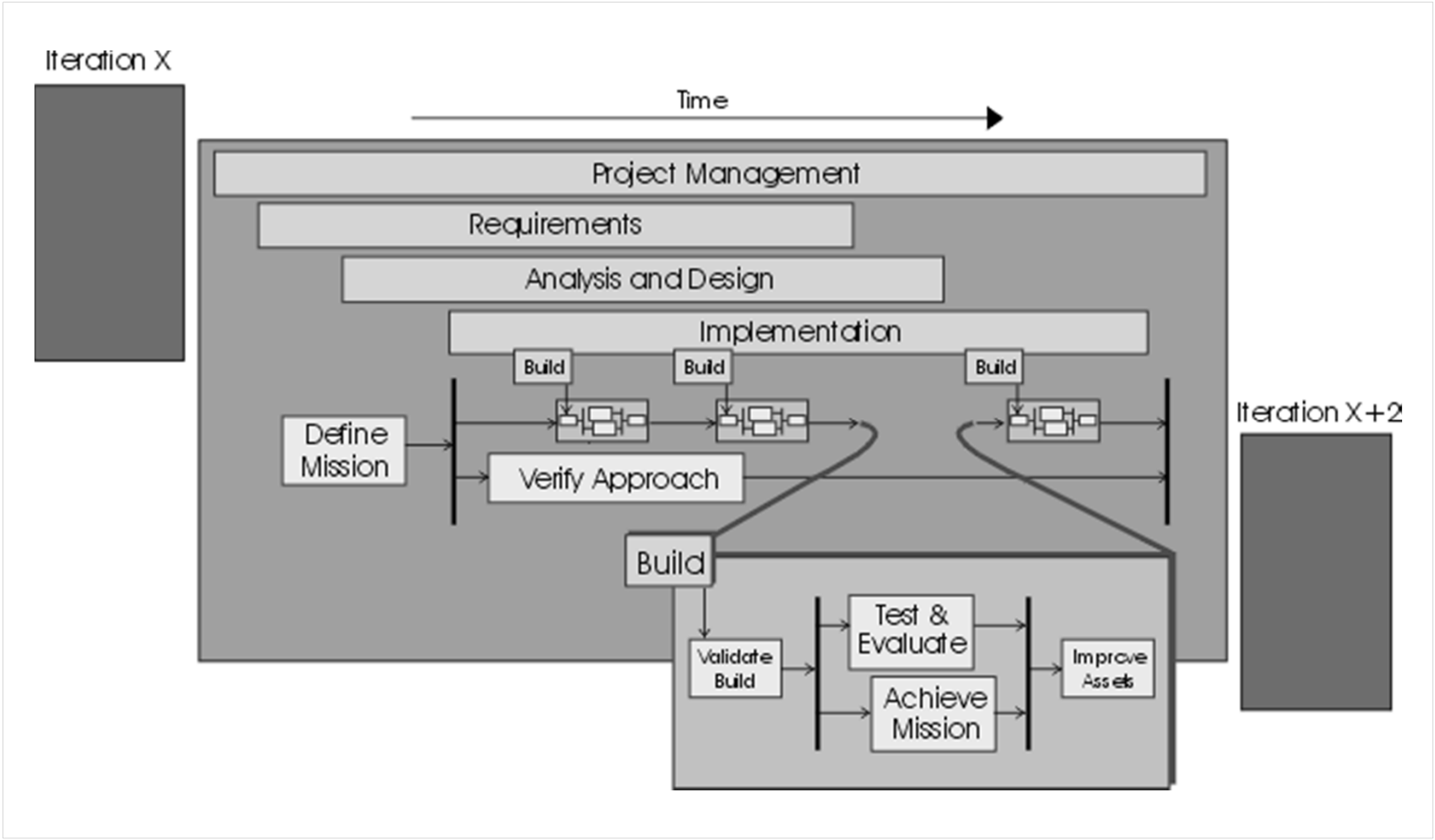
Each iteration results in an executable release (internal or external). Iterations are the “heartbeat” or rhythm of the project and a governing principle for testing in RUP.

The Test Workflow Occurs in Each Iteration

- This diagram represents testing work within an Iteration.
- All workflow details can be performed in a single Iteration.
- There are usually several Test Cycles within the Iteration.

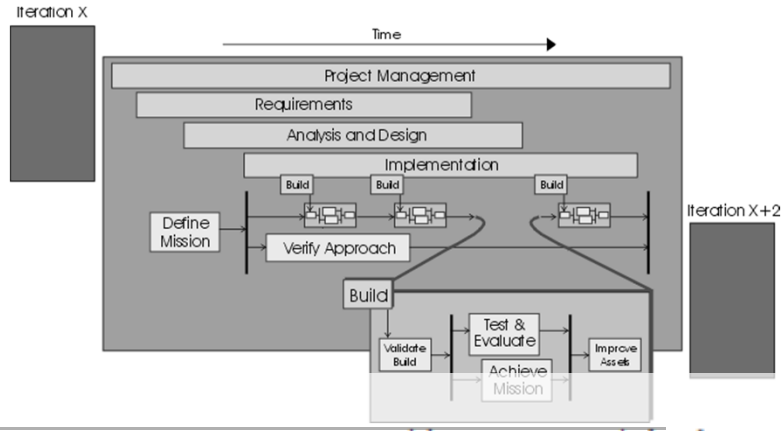


Each Build Is a Candidate for a Cycle of Testing



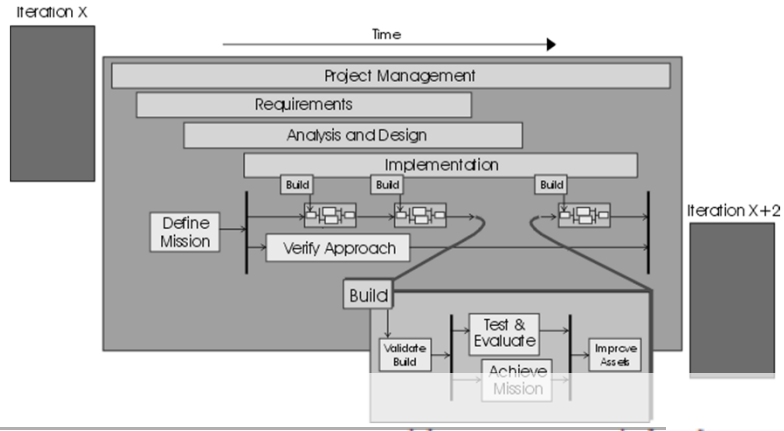
Each Build Is a Candidate for a Cycle of Testing

- Within an iteration, each software build is a potential candidate to be tested. Within each iteration – and for each build to be tested within the iteration – the challenge is to:
  - Focus on the most important factors that will motivate the test effort
  - Target your approach accordingly
  - Conduct appropriate tests to explore each motivating factor
  - Provide ongoing objective assessment of your findings in a timely manner
- These are all aspects of what is often referred to as “Context-Driven Testing”.



Each Build Is a Candidate for a Cycle of Testing

- Note that there are various considerations why a build may or may not be tested:
  - The build cycle is too frequent (e.g. daily), requiring too much overhead to complete a cycle of testing for each build.
  - The build is too unstable and/ or incomplete. Again, build verification testing helps to address this risk.
- In cases where a specific software build won't be tested, it typical for testing to continue on an earlier build.



Module 10 - Review

- What does it mean to Verify the Test Approach?
  - Name one key aspect that needs to be considered?
- Why are Build Verification Tests Important?
- When should you consider improving your test assets?