Agile and Industry Case Studies

Dr. John H Robb, PMP, IEEE SEMC
UTA Computer Science and Engineering

Motivation

- This presentation is intended to

- Discuss Agile approaches
  - Provide you with some insight into Agile team practices to let you see the latest in Agile approaches and how test maps into this
  - Prepare you for interviewing and being able to discuss these concepts
  - To prepare you to take the ISTQB Agile Tester extension certification if you should need to do so early in your career

- To show you some of the leading unit testing tools used in industry

- To provide you with some examples of industry case studies using test

The Agile Project

- The information from these slides is from the Agile Tester extension from the ISTQB website:
  - http://www.istqb.org/certification-path-root/agile-tester-extension.html
- A tester on an Agile project will work differently than one working on a traditional project.
- The members in an Agile project communicate with each other early and frequently.
- The Agile Manifesto contains four statements of values:
  - Individuals and interactions *over* processes and tools
  - Working software *over* comprehensive documentation
  - Customer collaboration *over* contract negotiation
  - Responding to change *over* following a plan
- Individuals and Interactions
  - Agile development is very people-centered.
  - It is through continuous communication and interaction that teams work most effectively.

The Agile Project (cont.)

- Working Software
  - From a customer perspective, working software is much more useful and valuable than overly detailed documentation.
  - Working software (reduced functionality) is available much earlier in the development lifecycle provides significant time-to-market advantage.

- Customer Collaboration
  - Customers often find great difficulty in specifying the system that they require.
  - Collaborating directly with the customer improves the likelihood of understanding exactly what the customer requires.

- Responding to Change
  - Change is inevitable in software projects. These factors must be accommodated by the development process.
  - Flexibility in work practices and planning to embrace change is more important than simply adhering rigidly to a plan.

The Agile Manifesto

- Principles - the core Agile Manifesto values are captured in 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, at intervals of between a few weeks to a few 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.

The Agile Manifesto (cont.)

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.

Agile Is A Whole Team Approach

- Whole-Team Approach
  - The whole-team approach means involving everyone with the knowledge and skills necessary to ensure project success.
  - The team includes representatives from the customer and other business stakeholders who determine product features.
  - The team should be relatively small; successful teams have been observed with as few as three people and as many as nine.
  - The whole- team approach is supported through the daily stand-up meetings involving all members of the team, where work progress is communicated and any impediments to progress are highlighted.
  - The whole-team approach promotes more effective and efficient team dynamics.

Agile Is A Whole Team Approach (cont.)

- The use of a whole-team approach to product development is one of the main benefits of Agile development. Its benefits include:
  - Enhancing communication and collaboration within the team
  - Enabling the various skill sets within the team to be leveraged to the benefit of the project
  - Making quality everyone's responsibility
- The whole team is responsible for quality in Agile projects.
  - The essence of the whole-team approach lies in the testers, developers, and the business representatives working together in every step of the development process.
  - Testers will work closely with both developers and business representatives to ensure that the desired quality levels are achieved.
  - This includes supporting and collaborating with business representatives to help them create suitable acceptance tests, working with developers to agree on the testing strategy, and deciding on test automation approaches.

Agile Is A Whole Team Approach (cont.)

- The concept of involving testers, developers, and business representatives in all feature discussions is known as the power of three
- Early and Frequent Feedback
  - Agile projects have short iterations enabling the project team to receive early and continuous feedback on product quality throughout the development lifecycle.
  - One way to provide rapid feedback is by continuous integration
- When sequential development approaches are used, the customer often does not see the product until the project is nearly completed.
- By getting frequent customer feedback as the project progresses, Agile teams can incorporate most new changes into the product development process.
- Early and frequent feedback helps the team focus on the features with the highest business value, or associated risk, and these are delivered to the customer first.

Benefits of Early and Frequent Feedback

- The benefits of early and frequent feedback include:
  - Avoiding requirements misunderstandings, which may not have been detected until later in the development cycle when they are more expensive to fix.
  - Clarifying customer feature requests, making them available for customer use early. This way, the product better reflects what the customer wants.
  - Discovering (via continuous integration), isolating, and resolving quality problems early.
  - Providing information to the Agile team regarding its productivity and ability to deliver.
  - Promoting consistent project momentum.

Extreme Programming

- There are several Agile approaches, each of which implements the values and principles of the Agile Manifesto in different ways. In this presentation, three representatives of Agile approaches are considered: Extreme Programming (XP), Scrum, and Kanban.

- Extreme Programming
  - XP embraces five values to guide development: communication, simplicity, feedback, courage, and respect.
  - XP describes a set of principles as additional guidelines: humanity, economics, mutual benefit, self- similarity, improvement, diversity, reflection, flow, opportunity, redundancy, failure, quality, baby steps, and accepted responsibility.
  - XP describes thirteen primary practices: sit together, whole team, informative workspace, energized work, pair programming, stories, weekly cycle, quarterly cycle, slack, ten-minute build, continuous integration, test first programming, and incremental design.
  - Many of the Agile software development approaches in use today are influenced by XP and its values and principles. For example, Agile teams following Scrum often incorporate XP practices.

Scrum

- Scrum is an Agile management framework which contains the following constituent instruments and practices:
  - Sprint: Scrum divides a project into iterations (called sprints) of fixed length (usually two to four weeks).
  - Product Increment: Each sprint results in a potentially releasable/shippable product (called an increment).
  - Product Backlog: The product owner manages a prioritized list of planned product items. The product backlog evolves from sprint to sprint (called backlog refinement).
  - Sprint Backlog: At the start of each sprint, the Scrum team selects a set of highest priority items (called the sprint backlog) from the product backlog.
  - Definition of Done: To make sure that there is a potentially releasable product at each sprint’s end, the Scrum team discusses and defines appropriate criteria for sprint completion.

Scrum (cont.)

  - Timeboxing: Only those tasks, requirements, or features that the team expects to finish within the sprint are part of the sprint backlog. If the development team cannot finish a task within a sprint, the associated product features are removed from the sprint and the task is moved back into the product backlog.
  - Transparency: The development team reports and updates sprint status on a daily basis at a meeting called the daily scrum.

- Scrum defines three roles:
  - Scrum Master: ensures that Scrum practices and rules are implemented and followed, and resolves any violations, resource issues, or other impediments that could prevent the team from following the practices and rules. This person is not the team lead, but a coach.
  - Product Owner: represents the customer, and generates, maintains, and prioritizes the product backlog. This person is not the team lead.
  - Development Team: develop and test the product. The team is self-organized: There is no team lead, so the team makes the decisions.

Kanban

- Scrum (as opposed to XP) does not dictate specific software development techniques (e.g., test first programming). In addition, Scrum does not provide guidance on how testing has to be done in a Scrum project.

- Kanban
  - Kanban is a management approach that is sometimes used in Agile projects.
  - The general objective is to visualize and optimize the flow of work within a value-added chain.

User Stories

- Collaborative User Story Creation
  - User stories are written to capture requirements from the perspectives of developers, testers, and business representatives.
  - This shared vision is accomplished through frequent informal reviews while the requirements are being written.
  - The user stories must address both functional and non-functional characteristics. Each story includes acceptance criteria for these characteristics and is complete when the criteria have been satisfied.
  - Typically, the tester’s unique perspective will improve the user story by identifying missing details or non-functional requirements.
  - A tester can contribute by asking business representatives open-ended questions about the user story, proposing ways to test the user story, and confirming the acceptance criteria.
- According to the 3C concept, a user story is the conjunction of three elements:
  - Card, Conversation, Confirmation (acceptance criteria)

Retrospectives

- In Agile development, a retrospective is a meeting held at the end of each iteration to discuss what was successful, what could be improved, and how to incorporate the improvements and retain the successes in future iterations.
- Retrospectives can result in test-related improvement decisions focused on test effectiveness, test productivity, test case quality, and team satisfaction.
- They may also address the testability of the applications, user stories, features, or system interfaces. Root cause analysis of defects can drive testing and development improvements.
- In general, teams should implement only a few improvements per iteration. This allows for continuous improvement at a sustained pace.

Continuous Integration

- Continuous Integration
  - Delivery of a product increment requires reliable, working, integrated software at the end of every sprint.
  - Continuous integration addresses this challenge by merging all changes made to the software and integrating all changed components regularly, at least once a day.
  - Configuration management, compilation, software build, deployment, and testing are wrapped into a single, automated, repeatable process.
  - Since developers integrate their work constantly, build constantly, and test constantly, defects in code are detected more quickly.

Continuous Integration (cont.)

- Following the developers' coding, debugging, and check-in of code into a shared source code repository, a continuous integration process consists of the following automated activities:
  - Static code analysis: executing static code analysis and reporting results
  - Compile: compiling and linking the code, generating the executable files
  - Unit test: executing the unit tests, checking code coverage and reporting test results
  - Deploy: installing the build into a test environment
  - Integration test: executing the integration tests and reporting results
  - Report (dashboard): posting the status of all these activities to a publicly visible location or e- mailing status to the team

Continuous Integration (cont.)

- An automated build and test process takes place on a daily basis and detects integration errors early and quickly.
- Continuous integration allows Agile testers to run automated tests regularly, in some cases as part of the continuous integration process itself, and send quick feedback to the team on the quality of the code.
- These test results are visible to all team members, especially when automated reports are integrated into the process.
- Automated regression testing can be continuous throughout the iteration. Good automated regression tests cover as much functionality as possible, including user stories delivered in the previous iteration
- In addition to automated tests, organizations using continuous integration typically use build tools to implement continuous quality control during build as opposed to applying quality control after completing all development.

Continuous Integration Benefits

- Allows earlier detection and easier root cause analysis of integration problems and conflicting changes
- Gives the development team regular feedback on whether the code is working
- Keeps the version of the software being tested within a day of the version being developed
- Reduces regression risk associated with developer code refactoring due to rapid re-testing of the code base after each small set of changes
- Makes progress toward the completion of the product increment visible, encouraging developers and testers
- Eliminates the schedule risks associated with big-bang integration
- Provides constant availability of executable software throughout the sprint for testing, demonstration, or education purposes
- Reduces repetitive manual testing activities
- Provides quick feedback on decisions made to improve quality and tests

Continuous Integration Challenges

- Continuous integration tools have to be introduced and maintained
- The continuous integration process must be defined and established
- Test automation requires additional resources and can be complex to establish
- Thorough test coverage is essential to achieve automated testing advantages
- Continuous integration requires the use of tools, including tools for testing, tools for automating the build process, and tools for version control.

Release and Iteration Planning

- For Agile lifecycles, two kinds of planning occur
  - Release planning
  - Iteration planning
- Release planning
  - Looks ahead to the release of a product, often a few months ahead of the start of a project.
  - Defines and re-defines the product backlog, and may involve refining larger user stories into a collection of smaller stories.
  - Provides the basis for a test approach and test plan spanning all iterations. Release plans are high-level.
  - Business representatives establish and prioritize the user stories for the release.
  - Based on these user stories, project and quality risks are identified and a high-level effort estimation is performed.

Release Planning (cont.)

- Testers are involved in release planning and especially add value in the following activities:
  - Defining testable user stories, including acceptance criteria
  - Participating in project and quality risk analyses
  - Estimating testing effort associated with the user stories
  - Defining the necessary test levels
  - Planning the testing for the release

Iteration Planning

- Iteration planning
  - Looks ahead to the end of a single iteration and is concerned with the iteration backlog.
  - The team selects user stories from the prioritized release backlog, elaborates the user stories, performs a risk analysis for the user stories, and estimates the work needed for each user story.
  - The number of stories selected is based on established team velocity and the estimated size of the selected user stories.
  - After the contents of the iteration are finalized, the user stories are broken into tasks, which will be carried out by the appropriate team members.

Iteration Planning (cont.)

- Testers are involved in iteration planning and especially add value in the following activities:
  - Participating in the detailed risk analysis of user stories
  - Determining the testability of the user stories
  - Creating acceptance tests for the user stories
  - Breaking down user stories into tasks (particularly testing tasks)
  - Estimating testing effort for all testing tasks
  - Identifying functional and non-functional aspects of the system to be tested
  - Supporting and participating in test automation at multiple levels of testing

Test Planning

- Release and iteration planning should address test planning as well as planning for development activities.
- Particular test-related issues to address include:
  - The scope of testing, the extent of testing for those areas in scope, the test goals, and the reasons for these decisions.
  - Who will carry out the test activities.
  - The test environment and test data needed, when they are needed, and whether any additions or changes to the test environment and/or data will occur prior to or during the project.
  - The timing, sequencing, dependencies, and prerequisites for the functional and non-functional test activities (e.g., how frequently to run regression tests, which features depend on other features or test data, etc.)
  - The project and quality risks to be addressed.
  - Consideration of the time and effort needed to complete the required testing activities.

Test Planning (cont.)

- Testers must understand the differences between testing in traditional lifecycle models (e.g., sequential such as the V-model or iterative such as RUP) and Agile lifecycles in order to work effectively and efficiently.
- The Agile models differ in terms of the way testing and development activities are integrated, the project work products, the names, entry and exit criteria used for various levels of testing, the use of tools, and how independent testing can be effectively utilized.
- Testers should remember that organizations vary considerably in their implementation of lifecycles.

Testing and Development Activities

- One of the main differences between traditional lifecycles and Agile lifecycles is the idea of very short iterations, each iteration resulting in working software that delivers features of value to business stakeholders.
- These iterations are highly dynamic, with development, integration, and testing activities taking place throughout each iteration, and with considerable parallelism and overlap.
- Testing activities occur throughout the iteration, not as a final activity.
- Testers, developers, and business stakeholders all have a role in testing, as with traditional lifecycles.
  - Developers perform unit tests as they develop features from the user stories.
  - Testers then test those features.
  - Business stakeholders also test the stories during implementation. Business stakeholders might use written test cases, but they also might simply experiment with and use the feature in order to provide fast feedback to the development team.

Test Automation

- Test automation at all levels of testing occurs in many Agile teams, and this can mean that testers spend time creating, executing, monitoring, and maintaining automated tests and results.
- While developers will focus on creating unit tests, testers should focus on creating automated integration, system, and system integration tests.
- This leads to a tendency for Agile teams to favor testers with a strong technical and test automation background.
- One core Agile principle is that change may occur throughout the project. Therefore, lightweight work product documentation is favored in Agile projects. Changes to existing features have testing implications, especially regression testing implications.
- The use of automated testing is one way of managing the amount of test effort associated with change.

Daily Stand Up

- The daily stand-up meeting includes all members of the Agile team including testers. At this meeting, they communicate their current status. The agenda for each member is [Agile Alliance Guide]:
- What have you completed since the last meeting?
- What do you plan to complete by the next meeting?
- What is getting in your way?
- Any issues that may block test progress are communicated during the daily stand-up meetings, so the whole team is aware of the issues and can resolve them accordingly.

Agile Testing Methods

- For Agile there are three main tenets of testing methods
  - Test-Driven Development
  - Acceptance Test-Driver Development
  - Behavior-Driven Development
- Test-Driven Development (primarily unit level)
  - Test-driven development (TDD) is used to develop code guided by automated test cases. The process for test-driven development is:
  - Add a test that captures the programmer's concept of the desired functioning of a small piece of code
  - Run the test, which should fail since the code doesn't exist
  - Write the code and run the test in a tight loop until the test passes
  - Refactor the code after the test is passed, re-running the test to ensure it continues to pass against the refactored code
  - Repeat this process for the next small piece of code, running the previous tests as well as the added tests

Agile Testing Methods (cont.)

- Acceptance Test-Driven Development
  - Acceptance test-driven development defines acceptance criteria and tests during the creation of user stories.
  - Acceptance test-driven development creates reusable tests for regression testing.
  - Acceptance test-driven development allows quick resolution of defects and validation of feature behavior. It helps determine if the acceptance criteria are met for the feature.

- Behavior-Driven Development
  - Behavior-driven development allows a developer to focus on testing the code based on the expected behavior of the software.
  - Because the tests are based on the exhibited behavior of the software, the tests are generally easier for other team members and stakeholders to understand.

Test Completion Criteria

- Unit testing
  - 100% decision coverage where possible, with careful reviews of any infeasible paths
  - Static analysis performed on all code
  - No unresolved major defects (ranked based on priority and severity)
  - No known unacceptable technical debt remaining in the design and the code [Jones11]
  - All code, unit tests, and unit test results reviewed
  - All unit tests automated
  - Important characteristics are within agreed limits (e.g., performance)

Test Completion Criteria (cont.)

- Integration testing
  - All functional requirements tested, including both positive and negative tests, with the number of tests based on size, complexity, and risks
  - All interfaces between units tested
  - All quality risks covered according to the agreed extent of testing
  - No unresolved major defects (prioritized according to risk and importance)
  - All defects found are reported
  - All regression tests automated, where possible, with all automated tests stored in a common repository

Test Completion Criteria (cont.)

- System testing
  - End-to-end tests of user stories, features, and functions
  - All user personas covered
  - The most important quality characteristics of the system covered (e.g., performance, robustness, reliability)
  - Testing done in a production-like environment(s), including all hardware and software for all supported configurations, to the extent possible
  - All quality risks covered according to the agreed extent of testing
  - All regression tests automated, where possible, with all automated tests stored in a common repository
  - All defects found are reported and possibly fixed
  - No unresolved major defects (prioritized according to risk and importance)

Microsoft Test Approach

- From - "How We Test Software at Microsoft", December 2008, Microsoft Press, ISBN 0735624259
- Microsoft employs as many software testers as developers.
- This book—written by three of Microsoft's most prominent test professionals—shares the best practices, tools, and systems used by the company's 9,000-strong corps of testers.
- Chapters on
  - Functional Testing Techniques
  - Structural Testing Techniques
  - Model based testing
- Microsoft makes heavy use of the test approaches we have been using
  - BVA/Equivalence partitions
  - Basis Path testing
  - Statement/decision/condition level coverage
- http://www.amazon.com/How-We-Test-Software-Microsoft/dp/0735624259

Google Test Approach

- How Google Tests Software, April 2012, Addison-Wesley Professional, 0321803027
- Describes testing responsibilities for software engineers (SWEs), software engineers in a test role (SETs), and Test Engineers (TEs).
- Much deals with the roles and responsibilities of the TE, and in the Google model, they do have a higher-level role in testing.
- In summary, the roles are:
  - SWEs develop unit tests for the their software
  - SETs develop tools to enable testing without external dependencies and develop automated functional tests
  - TEs coordinate the overall testing activities for a product and focus on the user by doing exploratory testing

Google Test Approach (cont.)

- Tests are categorized as small, medium, large or enormous.
  - Small tests are basically unit tests where everything external is mocked out, and are expected to execute in less than 100 ms.
  - Medium tests involve external subsystems, and can use database access, but typically run on one machine (use no network services), and are expected to run in under a second.
  - Large and enormous tests run a complete application, including all external systems required. They can be nondeterministic because of the complexity, and are expected to complete in 15 minutes and 1 hour respectively.
- Small tests lead to code quality, and medium, large and enormous tests lead to product quality.
- The common test execution environment for running the tests has been developed over time, and has several nice features. It will automatically kill tests that take too long to run (thus the time limits mentioned above).

Google Test Approach (cont.)

- The continuous integration system uses dependency analysis to run only tests affected by a certain change, thus being able to pinpoint exactly which change broke a certain test.
- There is also a good story about a 10-minute test plan. James Whittaker did an experiment where he forced people to come up with a test plan for a product in 10 minutes.
- The idea was to boil it down to the absolute essentials. Because of the time constraint, most people just made lists or grids - no text.
- In his opinion, this is the most useful level - it is quick to come up with and doesn't need a lot of busy-work filling out sections in a document template, and still it's a useful basis for coming up with test cases.
- The common theme in all cases was that people based the plan on capabilities that required testing.

Google Test Approach (cont.)

- Chapter 5, "Improving How Google Tests Software"
  - One of the problems Whittaker sees with testing is that testers are not part of the product development team.
  - Instead, they exist in their own organization, and this separation of testers and developers gives the impression that testing is not part of the product.
  - The focus of testing is often the testing activities and artifacts (the test cases, the bug reports etc.), not the product being tested. But customers don't care about testing per se, they care about products.
  - Whittaker's opinion is that testing should be the responsibility of all the developers working on the product. It should be their responsibility to test the product and to develop the appropriate tools (with some exceptions, for instance security testing).

Google Test Approach (cont.)

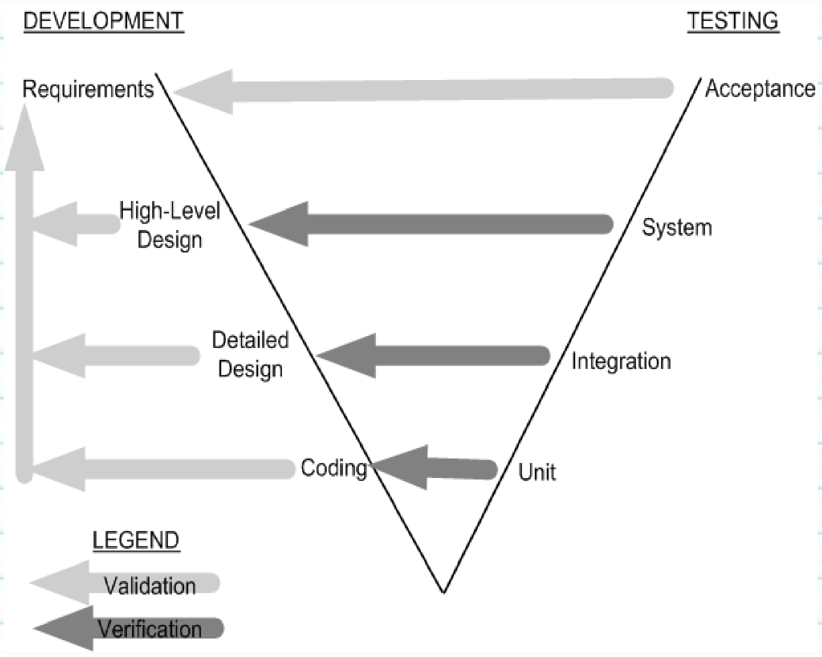
- Jim Whittaker

"...stop treating development and test as separate disciplines. Testing and development go hand in hand. Code a little and test what you built. Then code some more and test some more. Test isn't a separate practice; it's part and parcel of the development process itself. Quality is not equal to test. **Quality is achieved by putting development and testing into a blender and mixing them until one is indistinguishable from the other.**"

"...few schools systematically teach software testing. This makes hiring good testers a challenge for any company, because the right mix of coding and testing skills is truly rare... TEs are rare individuals. They are technical, care about the user, and understand the product at a system and end-to-end perspective... It's a small wonder Google, or any company for that matter, struggles to hire them."

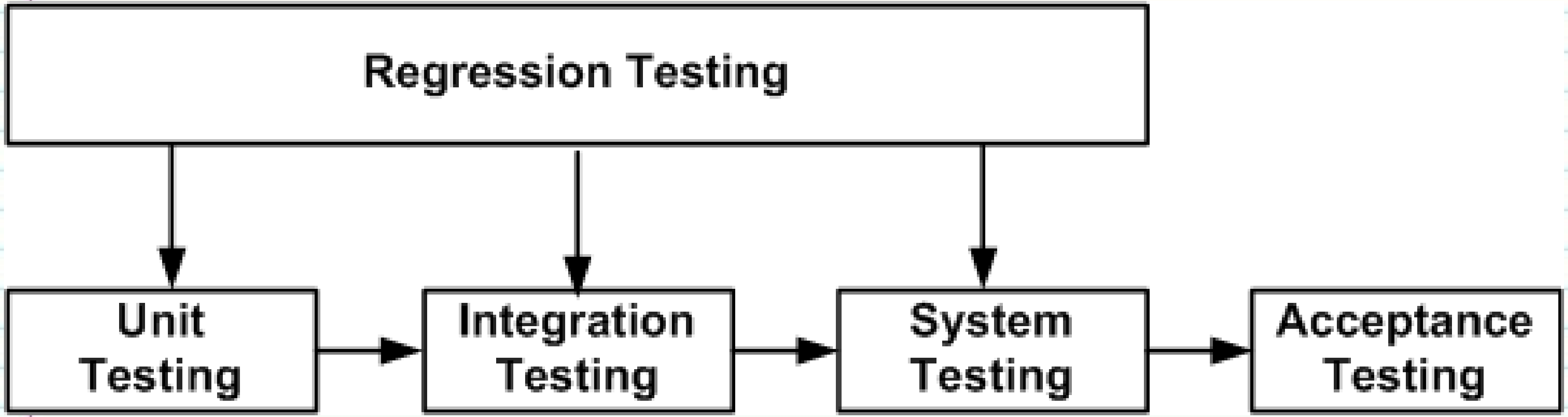
Heavy Industry Case Study

- This section addresses how aerospace, automotive, and medical industries test software



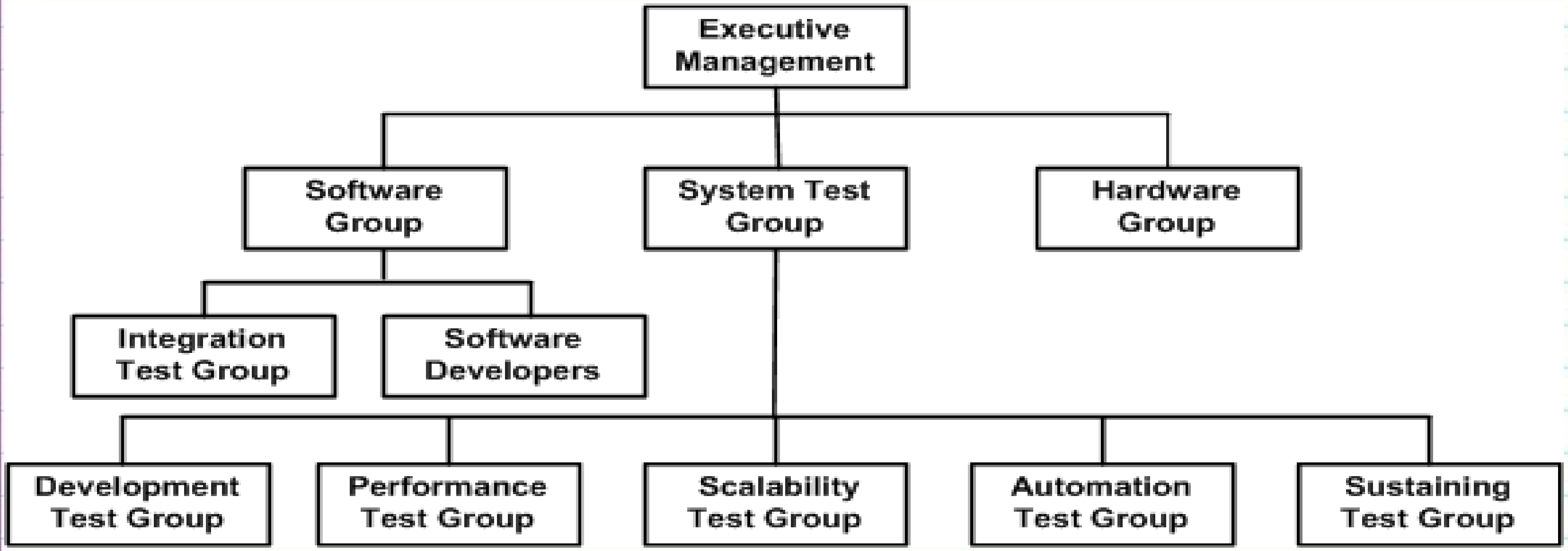
- Typical software engineer duties are to
  - Review software requirements (developed by system engineers)
  - Develop software design, code and unit tests
  - Support system level (independent team) testing

Re-Testing Level



- Unit Testing is typically only performed when the code is changed.
- Integration and System Level testing - there are typically two to five independent levels of these kinds of tests -
  - Based on the size and complexity of the system and
  - The number and levels of subsystems

Test Team Organization and Management



A Notional Organization Structure of test groups

- Software developers are typically in a separate organization from the System Test group.
- Sophisticated simulations of the to be delivered product (e.g., aircraft) exist in many industries
- Software and system test equipment consists of sophisticated stand-alone test rigs
  - Software unit testing is performed on a desktop - integration and system testing on a test station