LOG8371 : Ingénierie de la qualité en logiciel

Software Testing

Hiver 2017

Fabio Petrillo
Chargé de Cours

POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE

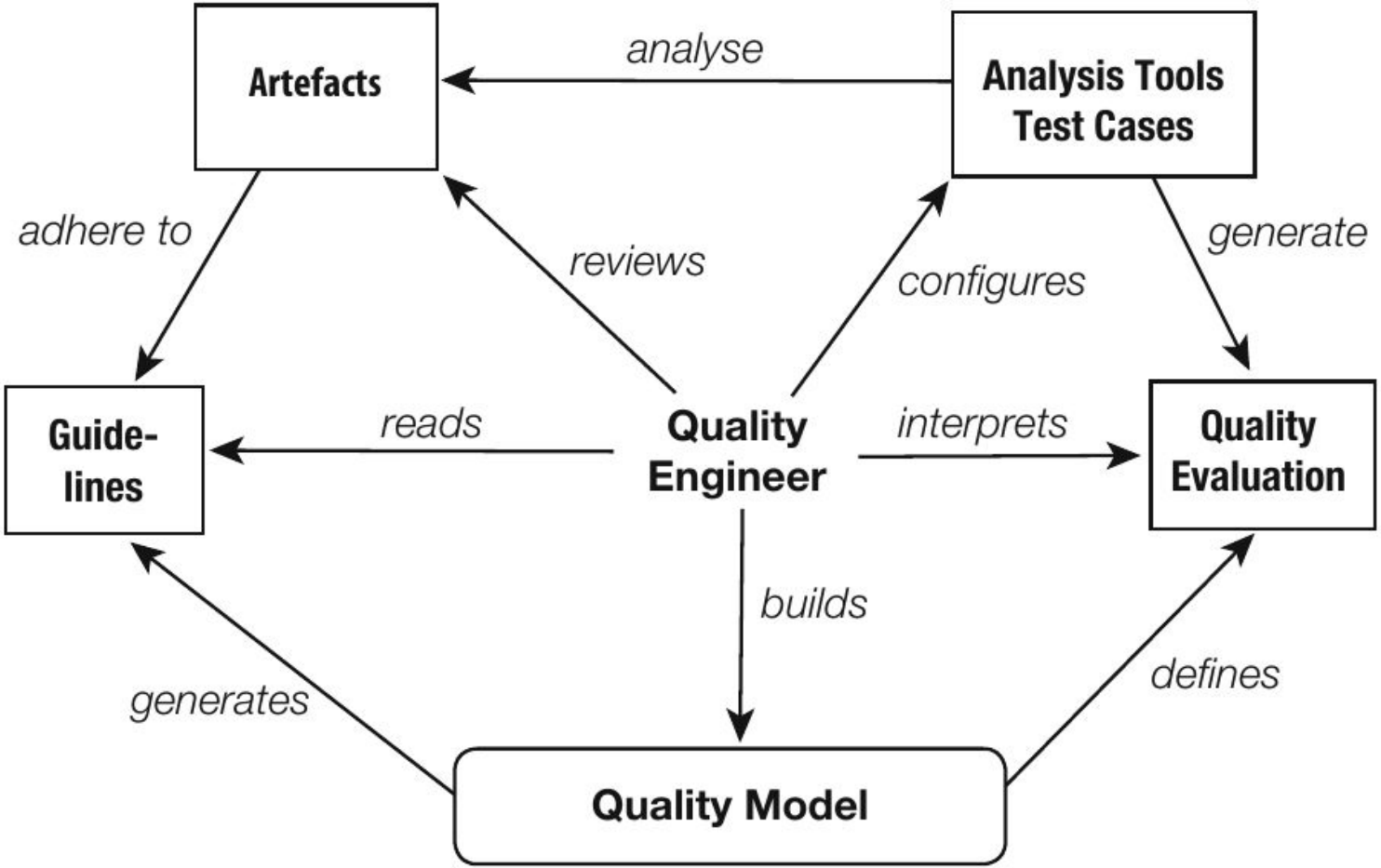
100 most cited word

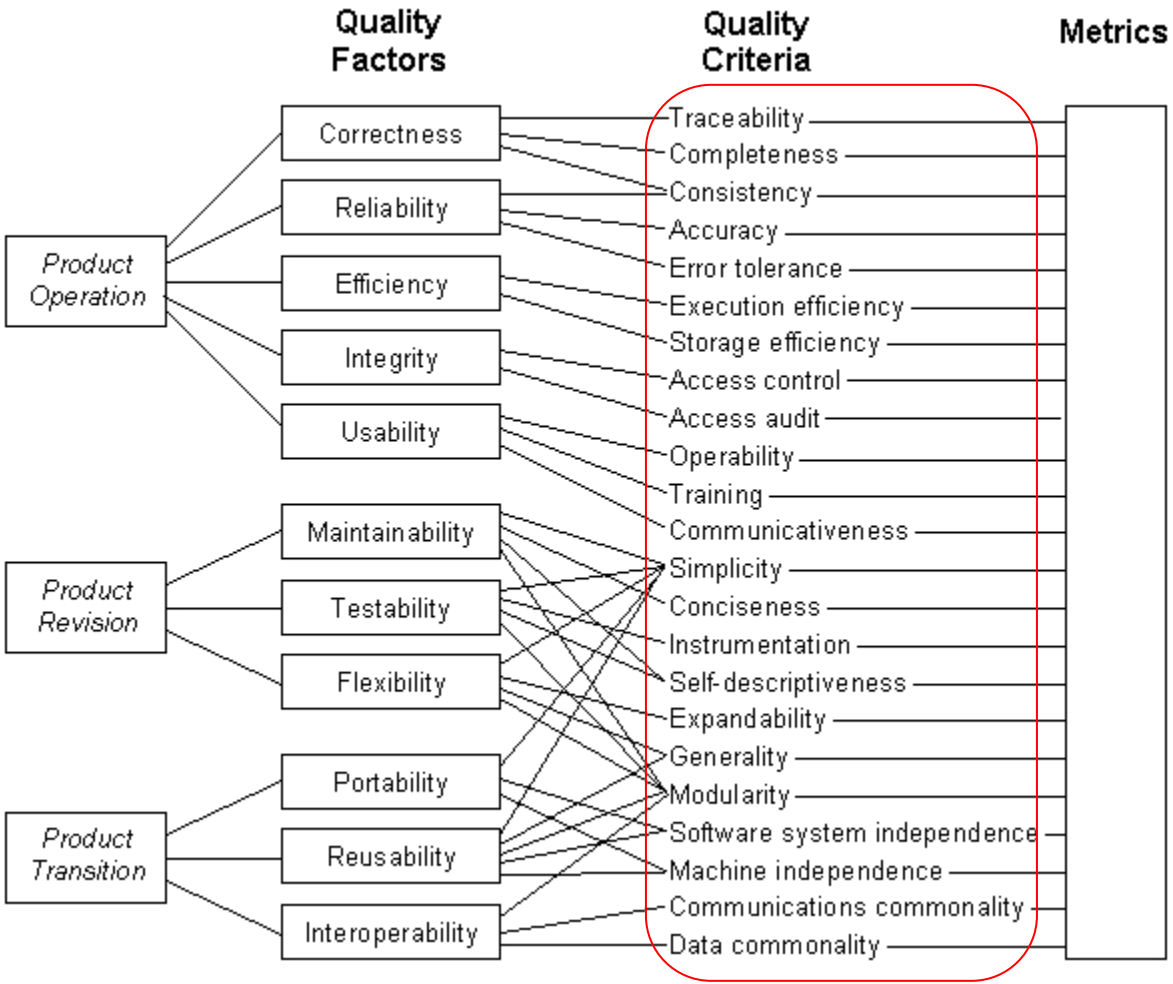
100 most cited word

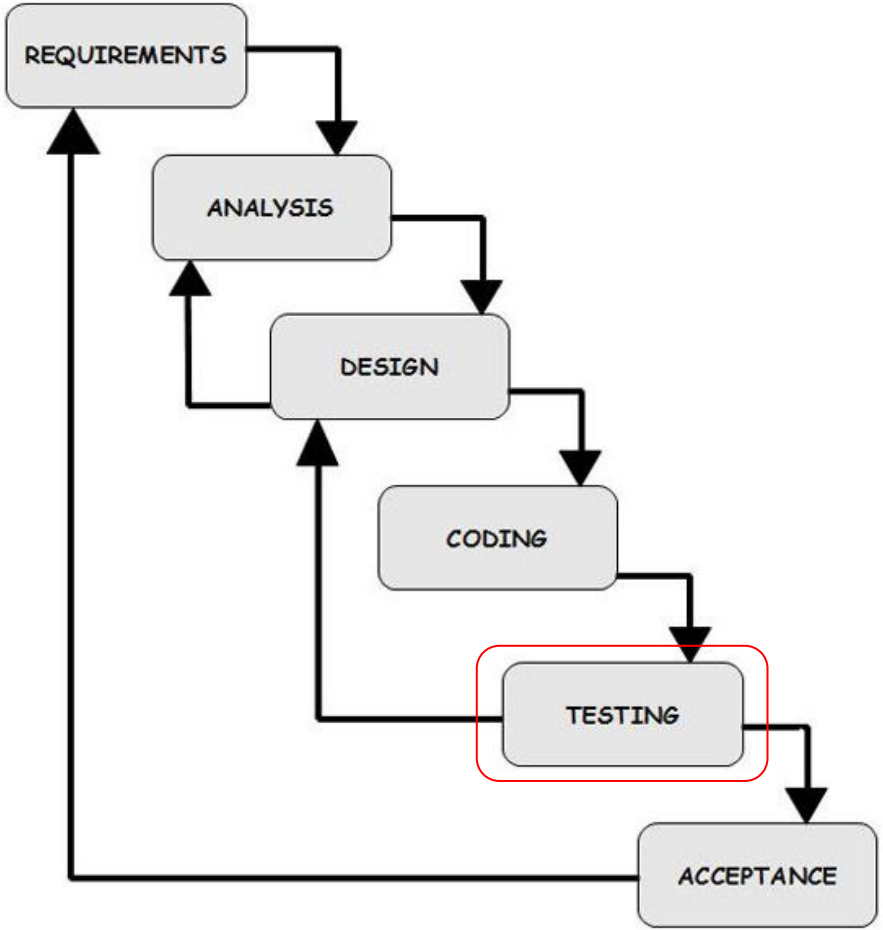
Senior Quality Assurance Analyst - Dropbox

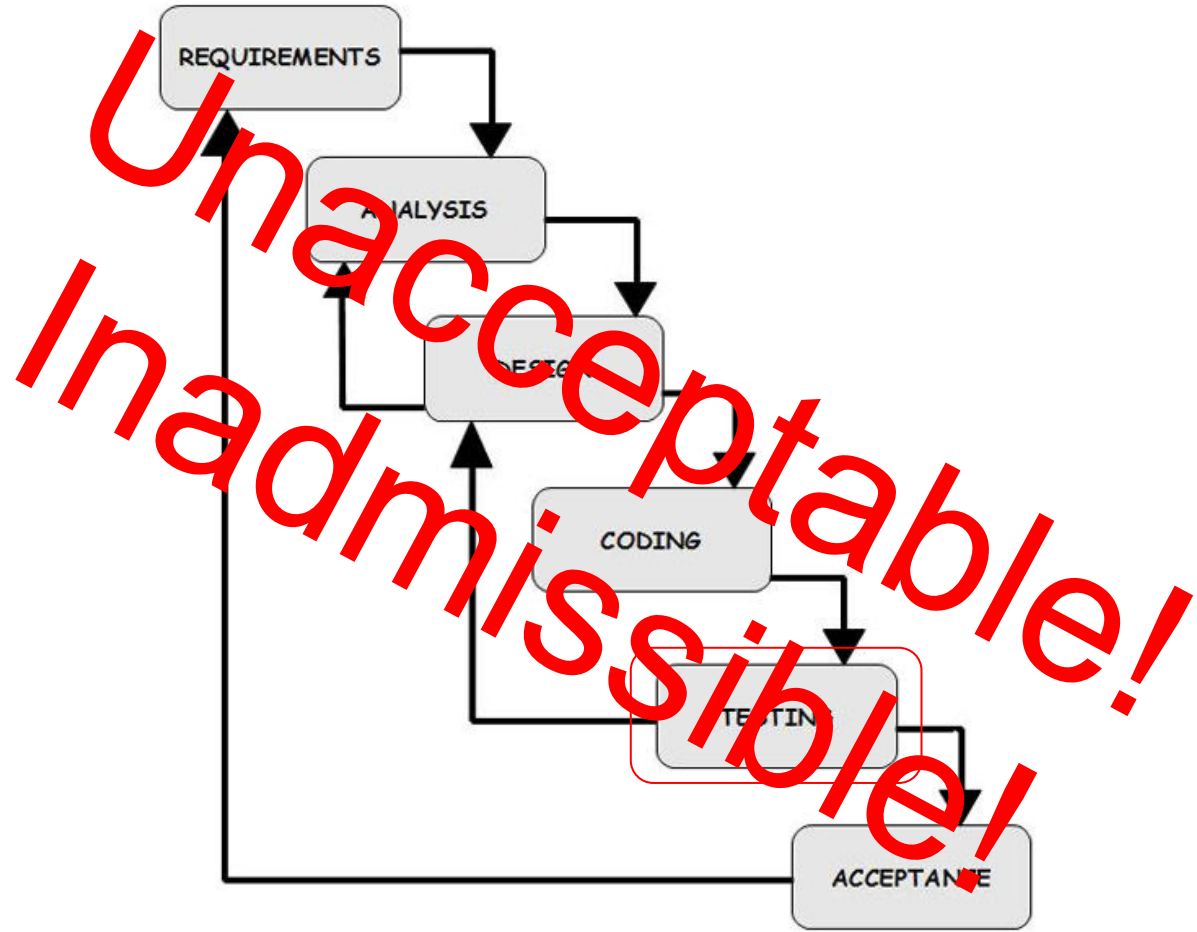
*“As the Senior Quality Assurance Analyst, you will **define, plan, and build-out QA testing automation** using leading **best practices**. In this role, you’ll have a wide range of responsibilities including setting the overall **testing automation strategy**, ensuring the right level of production monitoring is in place, and running manual testing.”*

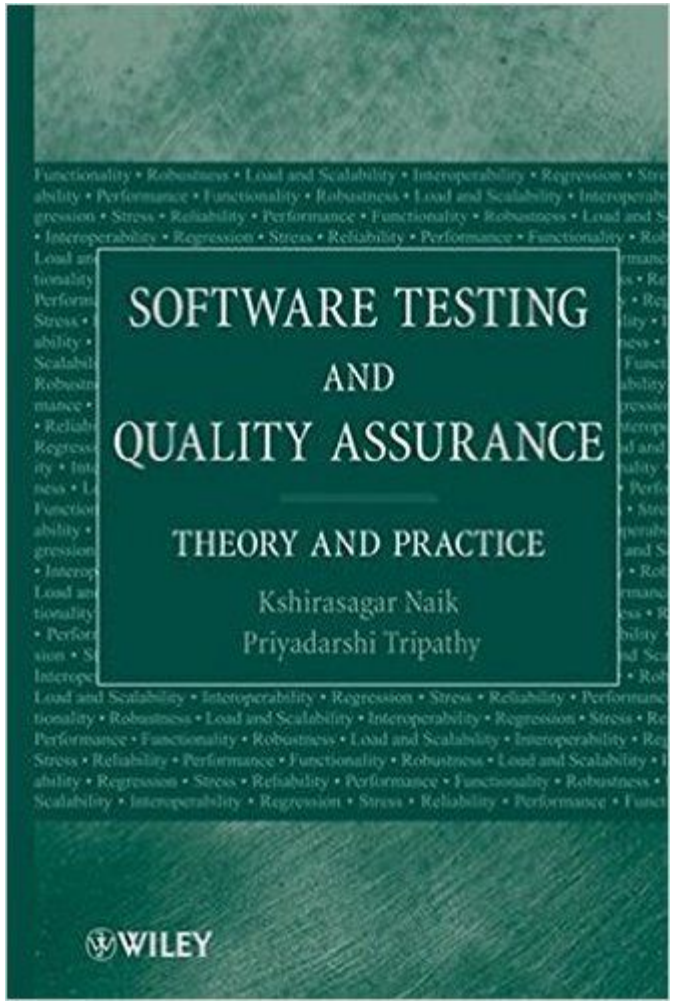
- Create **manual and automated test cases and test scripts**
- Execute **regression testing**, analyze **test results**, identify and log defects
- Analyze business requirements and technical design specifications to create appropriate **test strategies, test plans and test scripts**

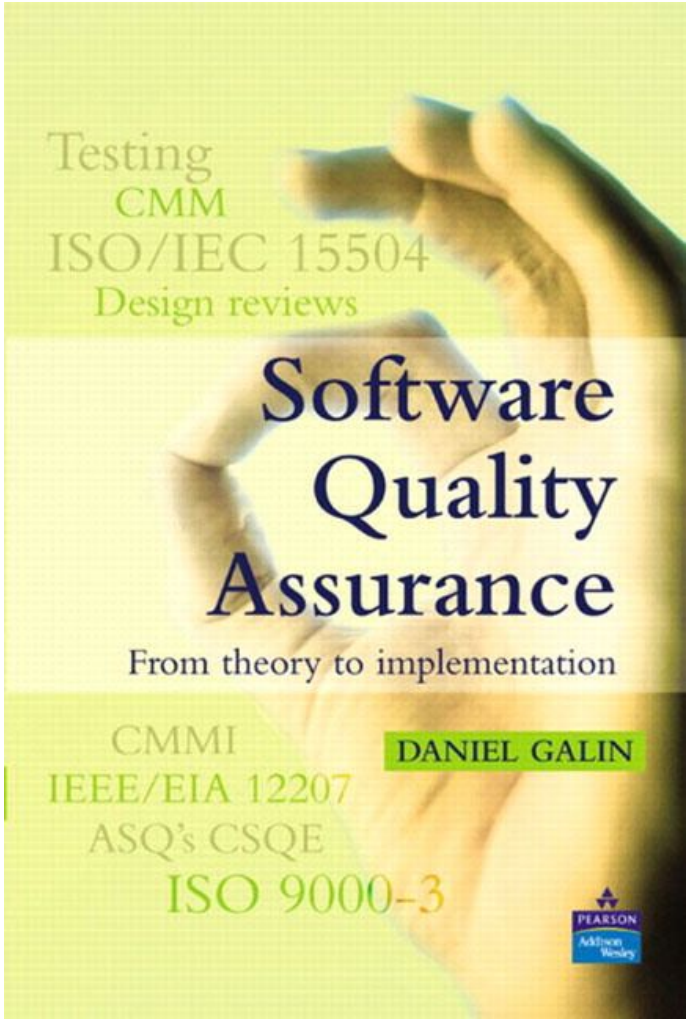

Artefacts
analyse
Analysis Tools
Test Cases
adhere to
reviews
configures
generate
Guide-
lines
reads
Quality
Engineer
interprets
Quality
Evaluation
builds
generates
defines
Quality Model


Quality Factors
Quality Criteria
Metrics
Product Operation
Product Revision
Product Transition
Correctness
Reliability
Efficiency
Integrity
Usability
Maintainability
Testability
Flexibility
Portability
Reusability
Interoperability
Traceability
Completeness
Consistency
Accuracy
Error tolerance
Execution efficiency
Storage efficiency
Access control
Access audit
Operability
Training
Communicativeness
Simplicity
Conciseness
Instrumentation
Self-descriptiveness
Expandability
Generality
Modularity
Software system independence
Machine independence
Communications commonality
Data commonality


REQUIREMENTS
ANALYSIS
DESIGN
CODING
TESTING
ACCEPTANCE


REQUIREMENTS
CODING
Unacceptable!
Inadmissible!


SOFTWARE TESTING
AND
QUALITY ASSURANCE
THEORY AND PRACTICE
Kshirasagar Naik
Priyadarshi Tripathy
WILEY


Testing
CMM
ISO/IEC 15504
Design reviews
Software
Quality
Assurance
From theory to implementation
CMMI
IEEE/EIA 12207
ASQ's CSQE
ISO 9000-3
DANIEL GALIN
PEARSON
Addison
Wesley

Definition of Testing [Myers' 1979]

*"**Testing** is the process of **executing** a program with intention of **finding errors**."*

Definition of Testing [Friedman and Voas, 1995]

*"Software testing is a **verification process** for software quality **assessment** and **improvement**."*

Thus, following that definition, testing is **static and dynamic** analysis!

Definition of Testing [Wagner 2013]

- Quality assurance activities performed by **running code**.
- All techniques that **execute the software system** and compare the **observed** behaviour with the **expected** behaviour.
- Testing is **always dynamic** quality assurance technique.
- Testers define test cases consisting of inputs and the expected outputs to **systematically analyse** all requirements.

Software testing objectives

- To identify and reveal as many errors as **possible** in the tested software.
- To bring the tested software, after correction of the identified errors and retesting, to an **acceptable level of quality**.
- To perform the required tests efficiently and effectively, within **budgetary** and **scheduling** limitations.
- To compile a record of software errors for use in error prevention (by corrective and preventive actions).
- Reduce the **risk** of failure
- Reduce the **cost** of testing
- Dijkstra: "*Testing can only reveal the presence of errors, never their absence*"
- **Bug-free software is still a utopian aspiration!**

“I have exhaustively tested the program.”

Concept of Complete Testing

- “I have **exhaustively** tested the program.” **????**
- **For most of the systems, complete testing is near impossible**
- The domain of possible **inputs** of a program is too **large** to be completely used in testing a system.
- The **design issues** may be too **complex** to completely test.
- It may **not be possible** to create **all** possible **execution environments** of the system.
- Thus, discovering all faults, is highly desirable, but it is a **near-impossible** task, and it may not be attempted.
- Best thing is to **select** a subset of the input domain **to test** a program.

Test case

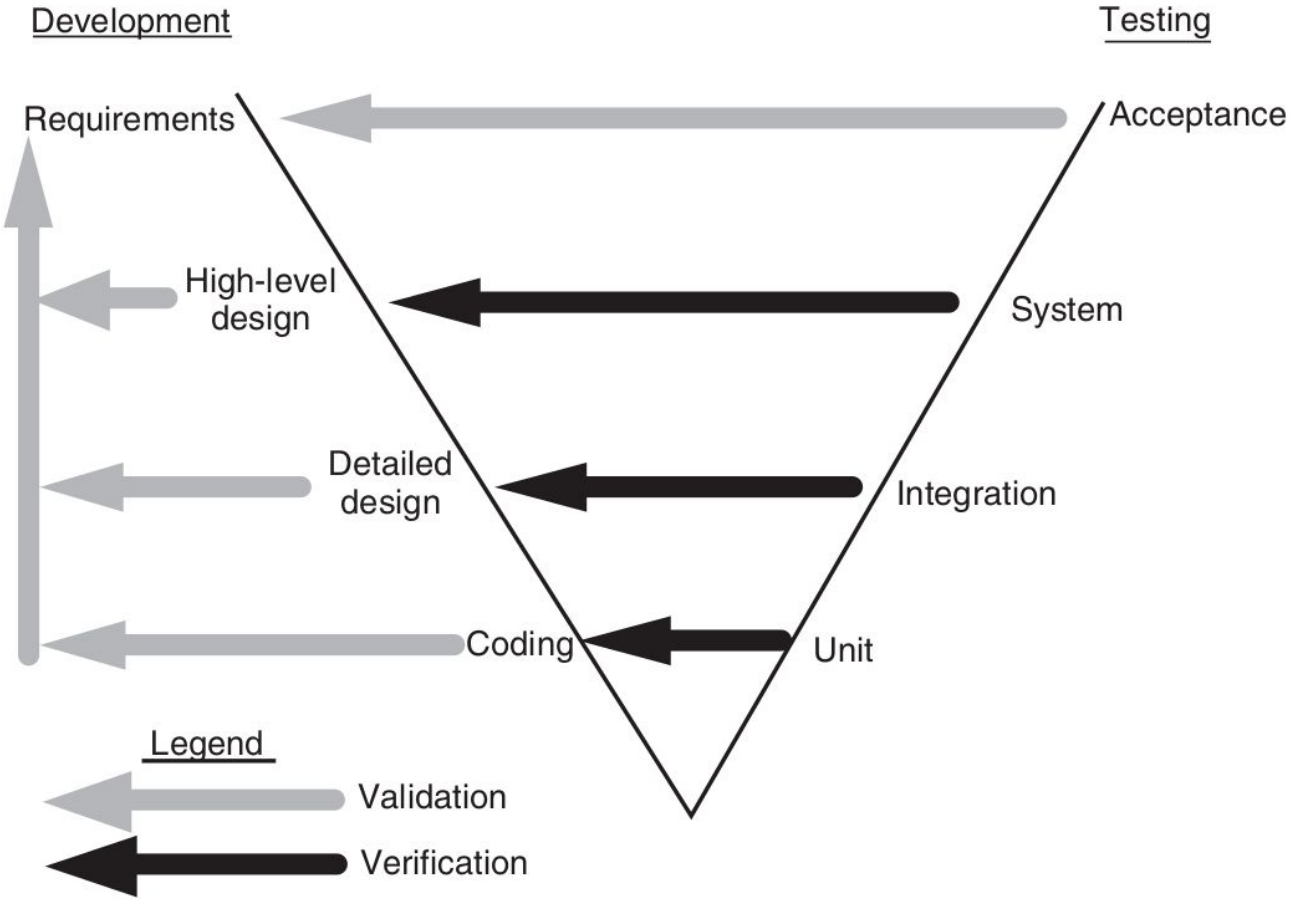
- a **test case** is a simple pair of < **input, expected outcome** >
- To function square root, some test cases:

TB_1: $<0, 0>$,
TB_2: $<25, 5>$,
TB_3: $<40, 6.3245553>$,
TB_4: $<100.5, 10.024968>$.

Testing Activities

- Identify an objective to be tested
  - A clear purpose must be associated with every test case
- Select inputs
- Compute the **expected** outcome
- Set up the execution environment of the program
- Execute the program
- Analyze the test result
  - pass, fail , and inconclusive

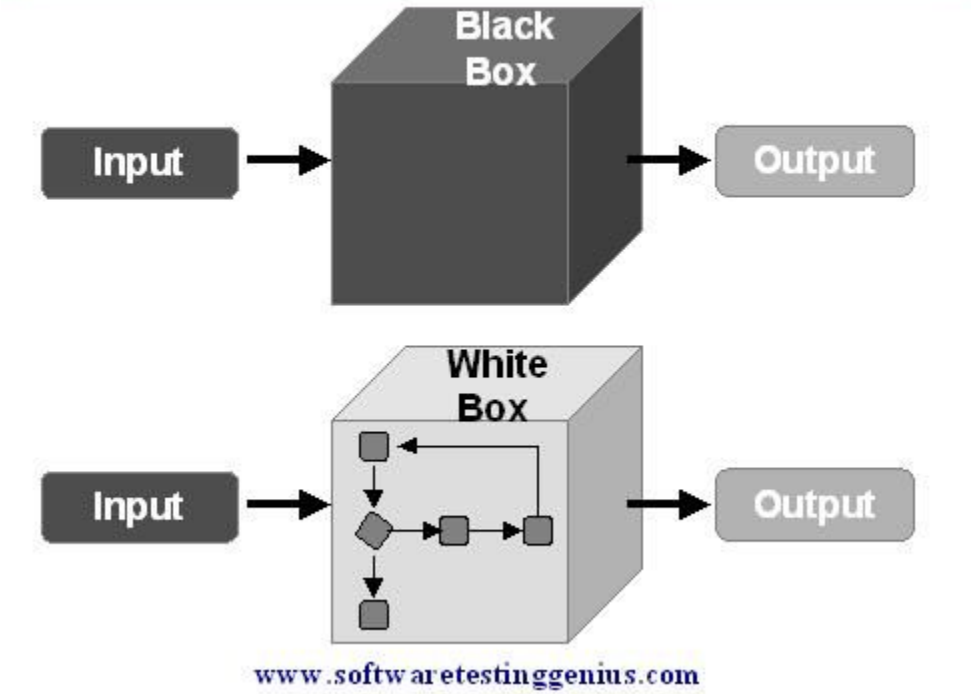
Testing Levels (V Model)



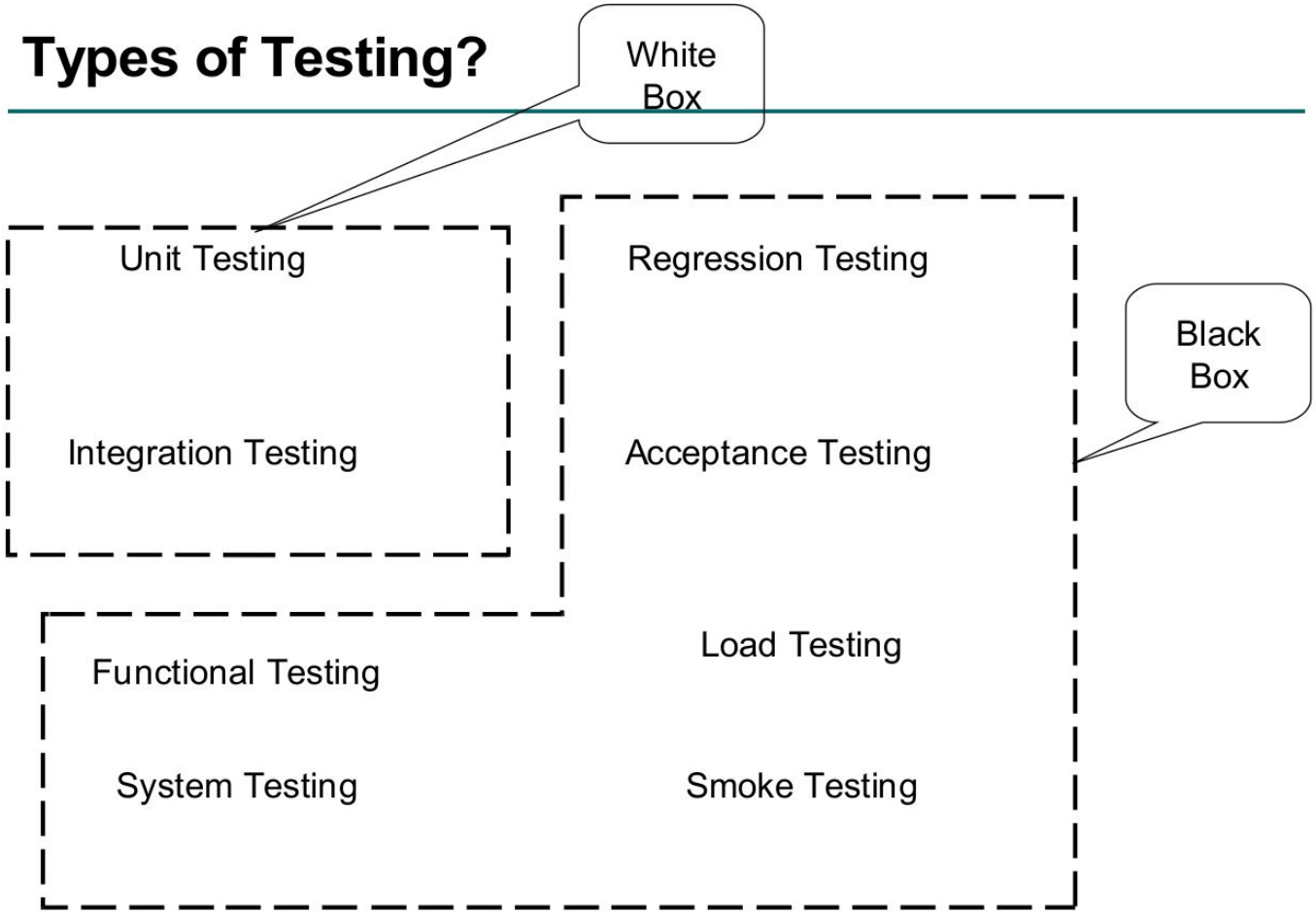
Sources of information for test case selection

- Requirements and functional specifications
  - Plain text, diagrams, equations
- Source code
- Input and output domains
- Operational profile
  - quantitative characterization of how a system will be used
  - samples of system usage
- Fault model
  - Previously encountered faults are an excellent source of information in designing new test cases
  - error guessing
  - fault seeding: fault injection to test the effectiveness of the test suite

White box and Back box testing



Types of Testing?



Unit Test
is the topic of
the next course.

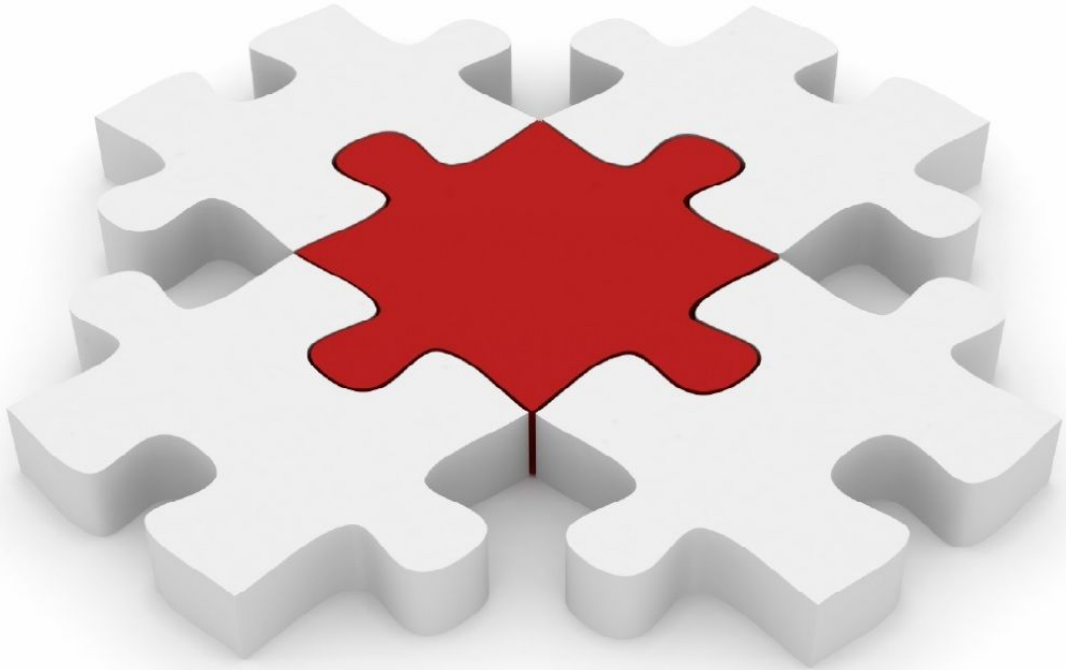
System Integration Testing

- A software module, or component, is a self-contained element of a system.
- Modules have well-defined **interfaces** with other modules.
- Constructing a working system from the pieces is not a straightforward task, because of numerous **interface errors**.
- ***Integration testing*** is to **assemble** a reasonably stable system in a **laboratory environment** such that the integrated modules and components in the actual environment of the system.
- **System integration testing** is a **systematic** technique for **assembling** a software system while conducting tests to **uncover errors** associated with **interfacing**.

System Integration Testing - Importance

- Different modules are generally created by groups of **different developers**.
- **Unit testing** of individual modules is carried out in a **controlled environment** by using test drivers and stubs.
- Some modules are **more error prone** than other modules, because of their inherent **complexity**. It is essential to identify the **ones** causing **most failures**.
- Perry and Evangelist reported in 1987 that interface errors accounted for **up to a quarter of all errors in the systems they examined**.
- Today, it is probably **worse**!

System Integration Testing



Different type of Interfaces

- **Procedure Call Interface**: a procedure in one module calls a procedure in another module.
  - Module SalaryManager -> (new EmployeeModel()).increaseSalary(10000);
- **Shared Memory Interface**: a block of memory is shared between two modules.
  - Shared filesystem
- **Message Passing Interface**: One module prepares a message by initializing the fields of a data structure and sending the message to another module.
  - Client/Server - Database
  - Web applications - HTTP, SOAP, REST

Advantages of integration testing

- Defects are detected **early**.
- It is **easier** to fix defects detected earlier.
- We get earlier feedback on the **health and acceptability** of the individual modules and on the overall system.
- **Scheduling** of defect fixes is flexible, and it can overlap with development.

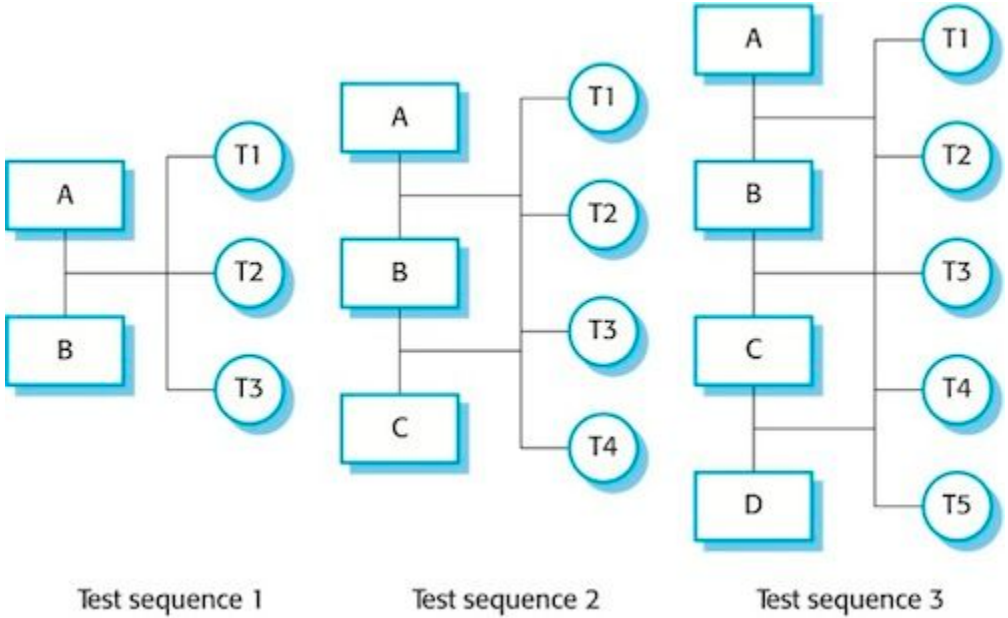
Granularity of System Integration

- **Intrasystem** Testing: combining modules together to build a cohesive system
- **Intersystem** Testing: interfacing independently tested systems
- **Pairwise** Testing: many intermediate levels of system integration testing between the above two extreme levels, namely intrasystem testing and intersystem testing.

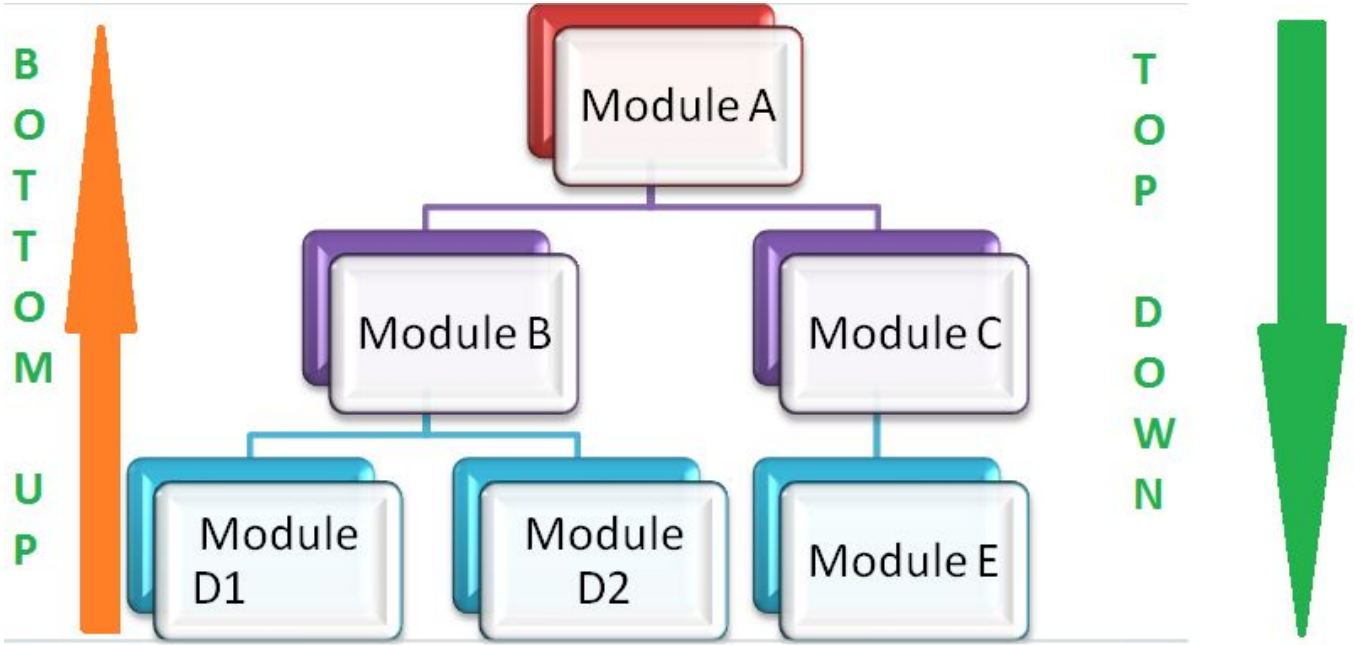
Integration testing Techniques

- Integration testing **need not wait** until all the modules of a system are coded and unit tested.
- Integration testing can [or have to] begin **as soon as the relevant** modules are available.
- Common approaches
  - Incremental
  - Top down
  - Bottom up
  - Sandwich
  - Big bang

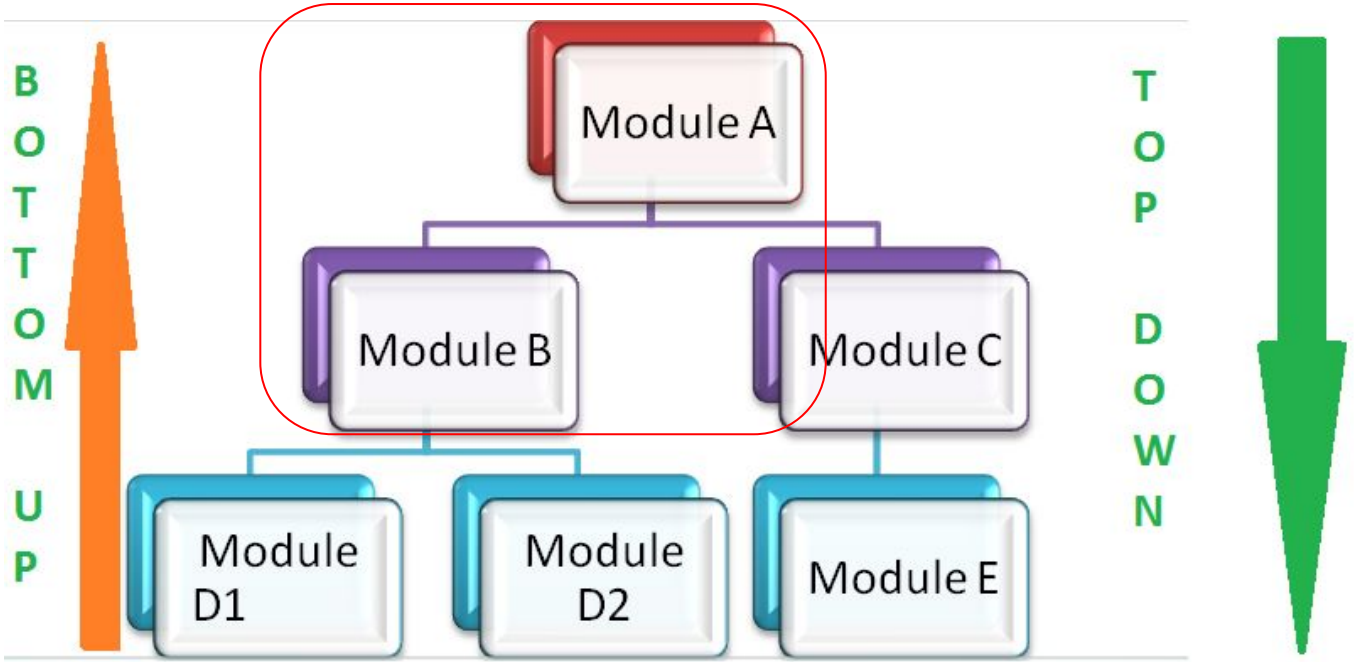
Incremental integration testing



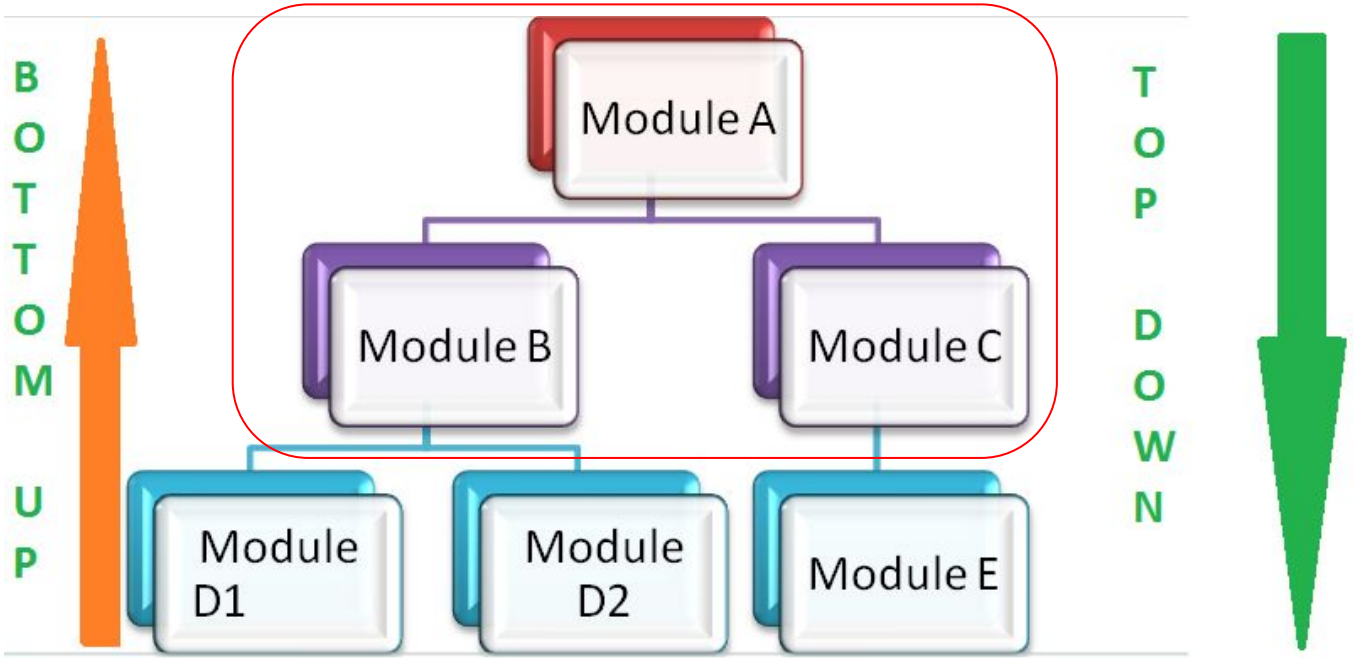
Top Down and *Bottom Up* integration testing



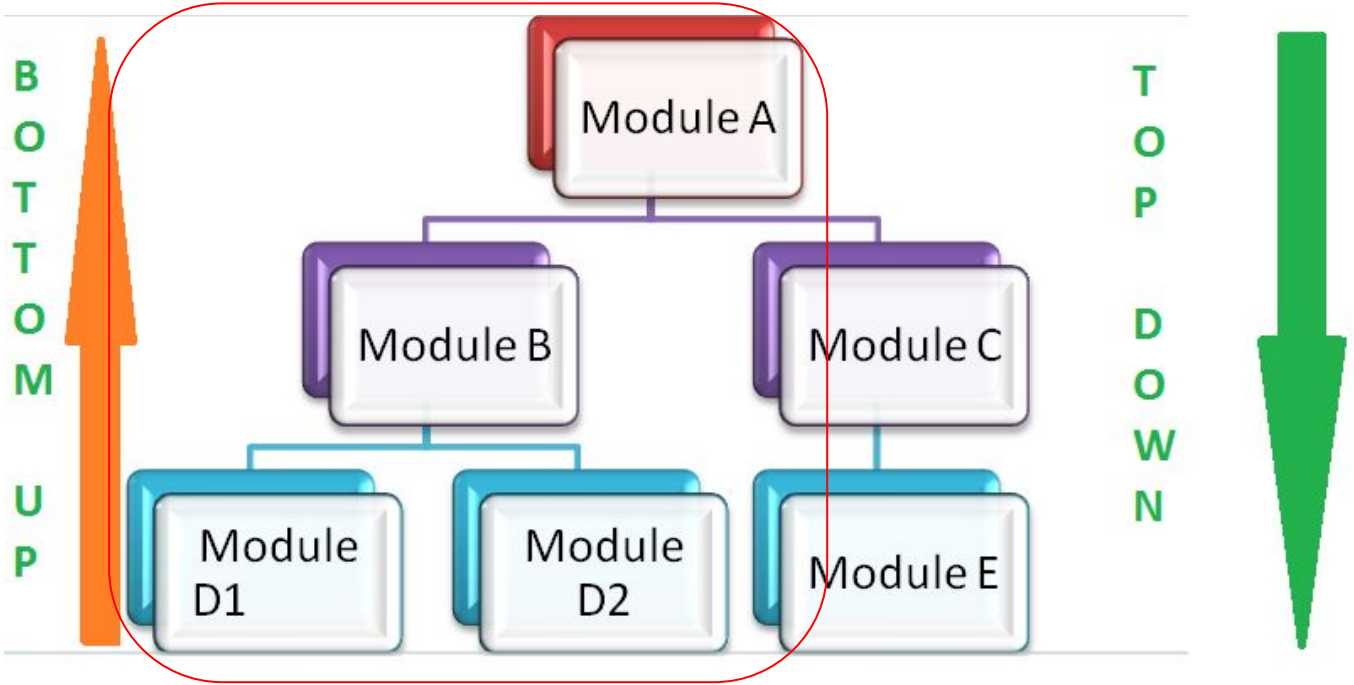
Top Down and *Bottom Up* integration testing



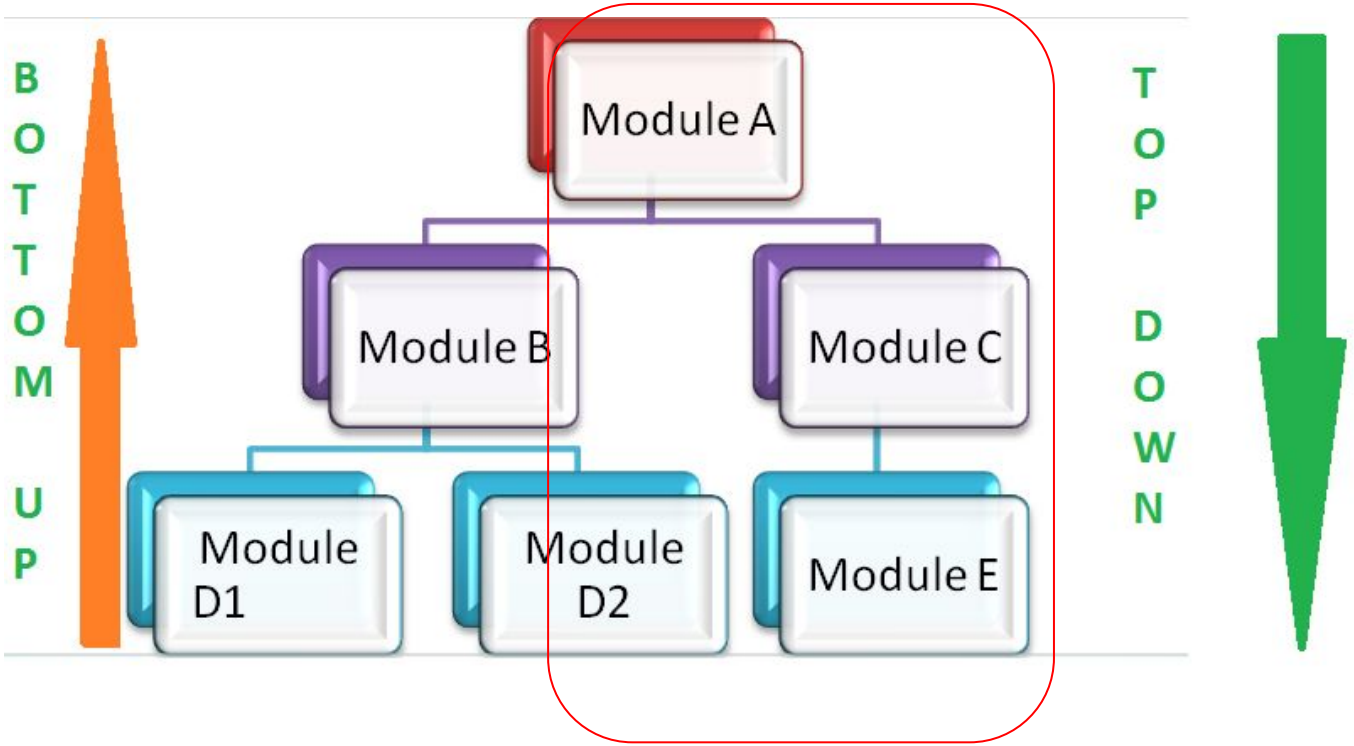
Top Down and *Bottom Up* integration testing



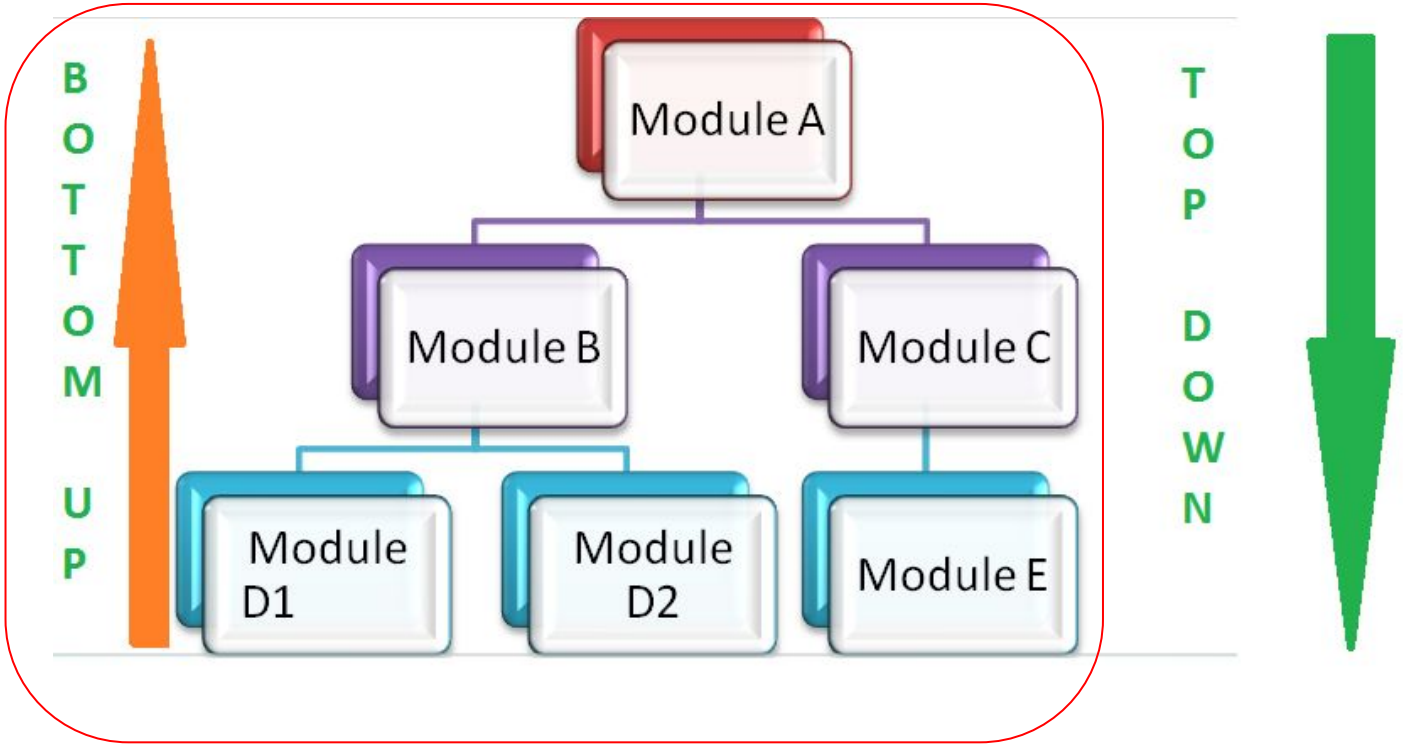
Top Down and *Bottom Up* integration testing



Top Down and *Bottom Up* integration testing



Top Down and *Bottom Up* integration testing



Sandwich and *Big Bang* integration testing

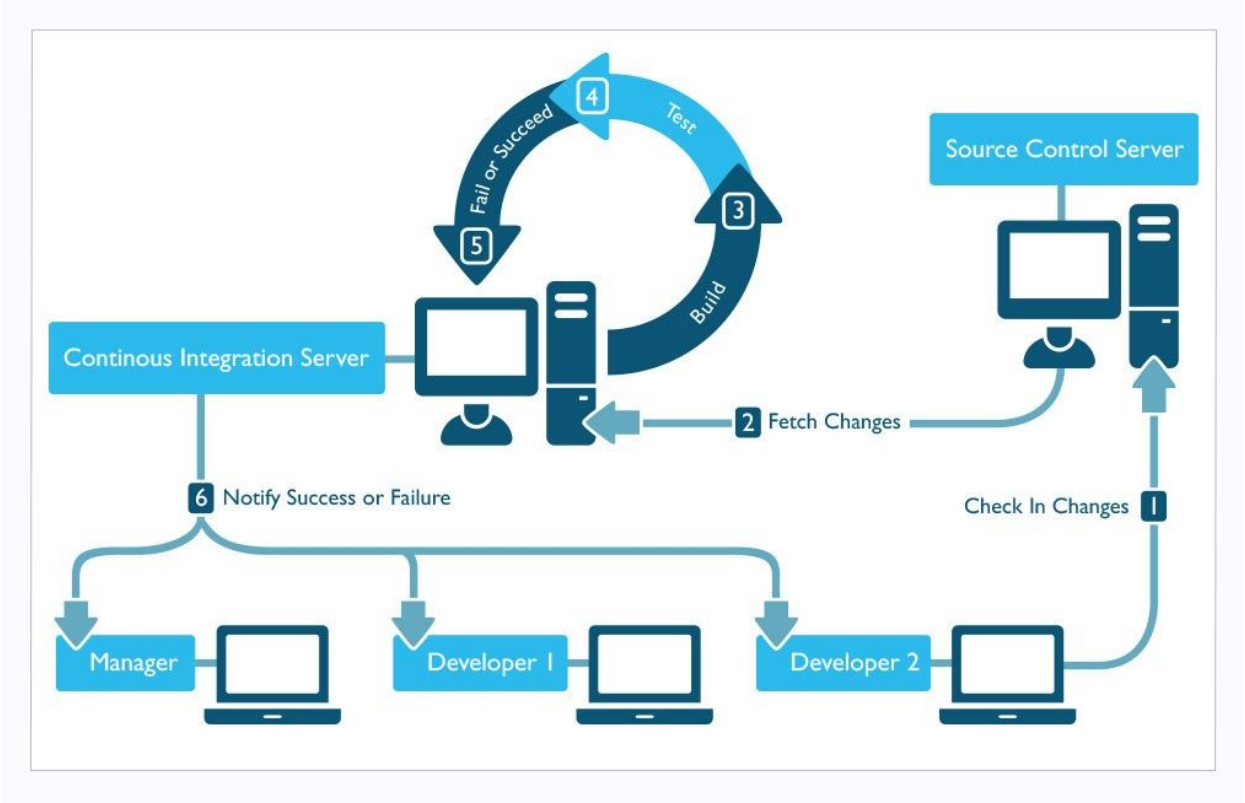
- Sandwich: a mix of the top-down and bottom-up approaches

- Big Bang
  - first all the modules are individually tested
  - all those modules are put together to construct the entire system which is testedas a whole
  - Okay for small systems

Empirical results about the approaches

- Solheim and Rowland measured the relative efficacy of top-down, bottom-up, sandwich, and big-bang integration approaches for software systems.
- **Top-down integration** strategies are **most effective** in terms of defect correction.
- **Top-down and big-bang** strategies produced the most **reliable** systems.

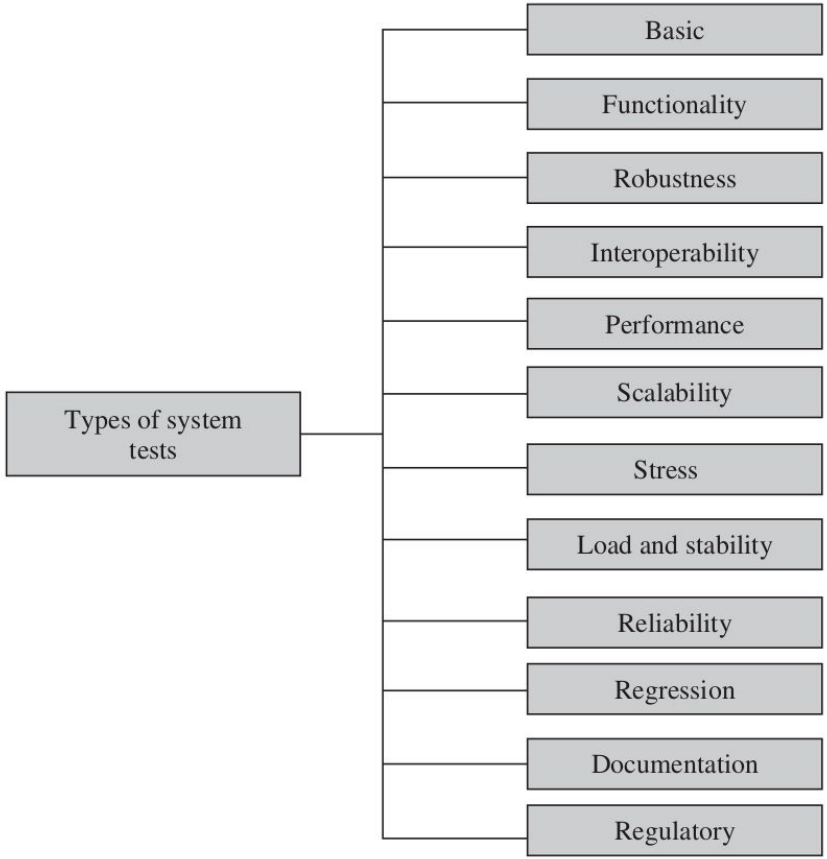
The goal -> **Continuous Integration**



System Testing

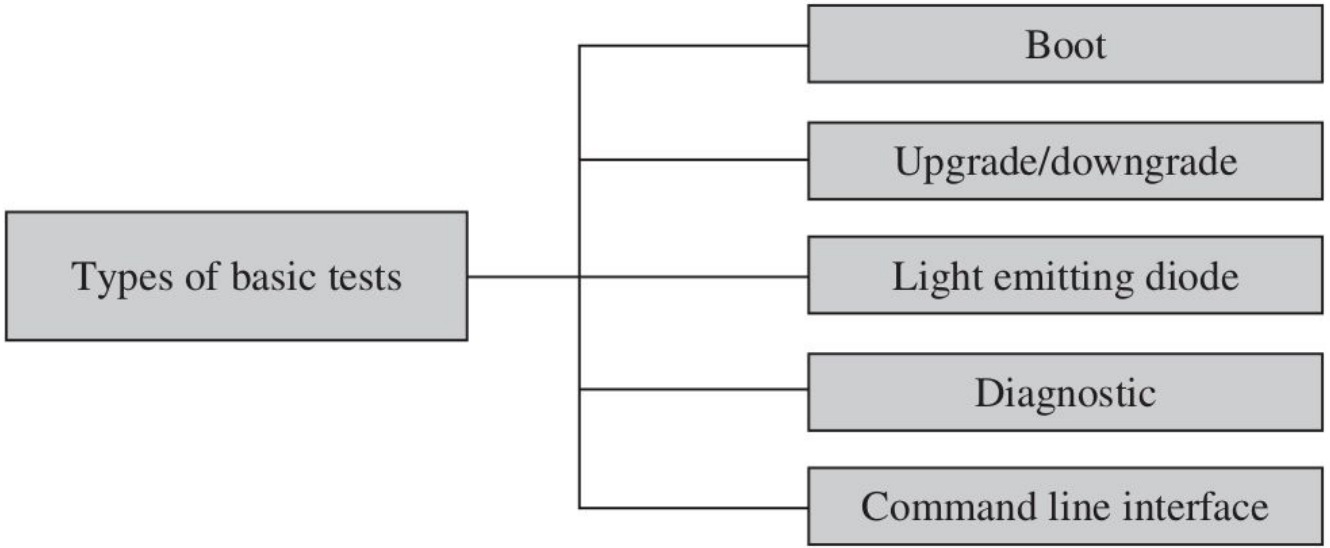
- Establish whether an implementation **conforms** to the **requirements** specified [**or not**] by the customers.
- System is **acceptable**
- **12 main types of system tests**

Types of system testes



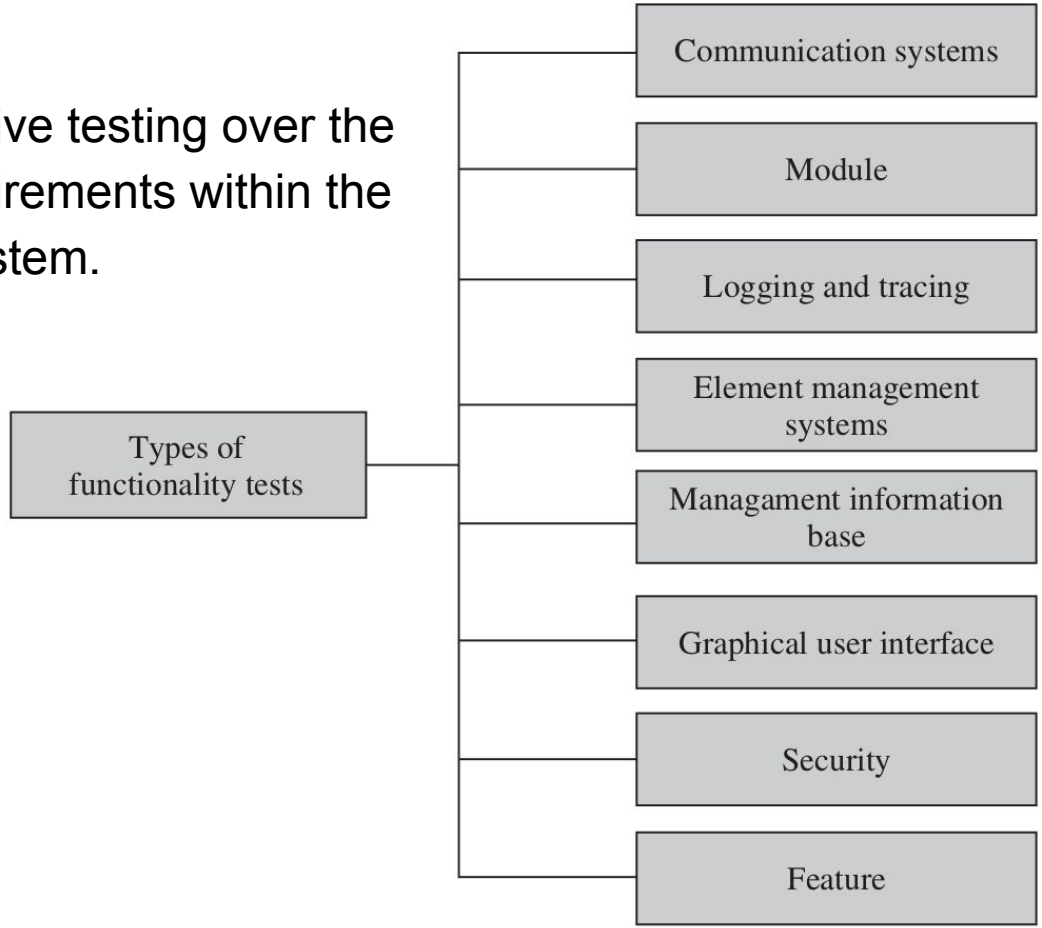
Basic tests (smoke test)

- provide an evidence that the system can be installed, configured, and brought to an operational state.



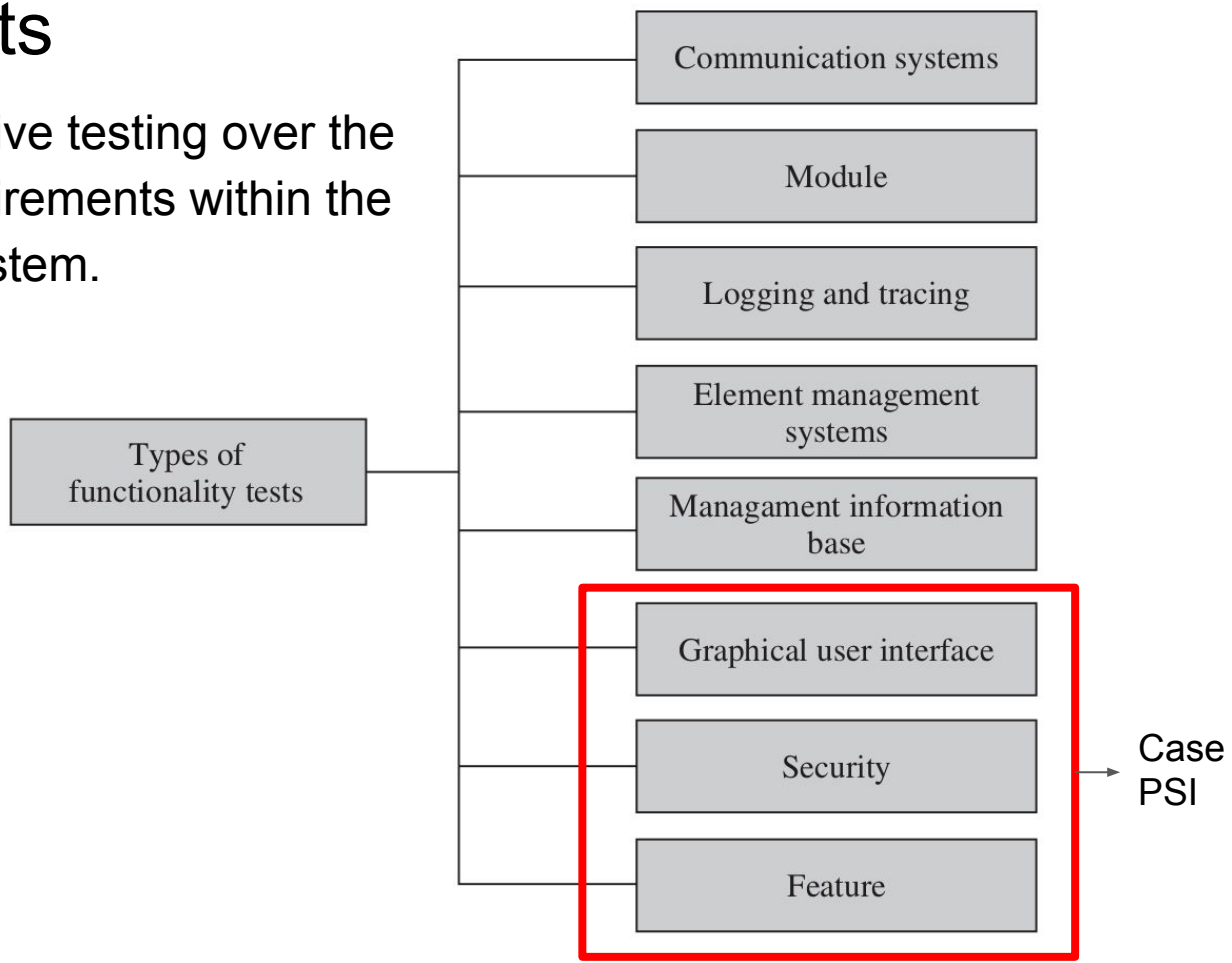
Functionality tes

- Provide comprehensive testing over the full range of the requirements within the capabilities of the system.



Functionality tests

- Provide comprehensive testing over the full range of the requirements within the capabilities of the system.



Reliability, Robustness and Interoperability tests

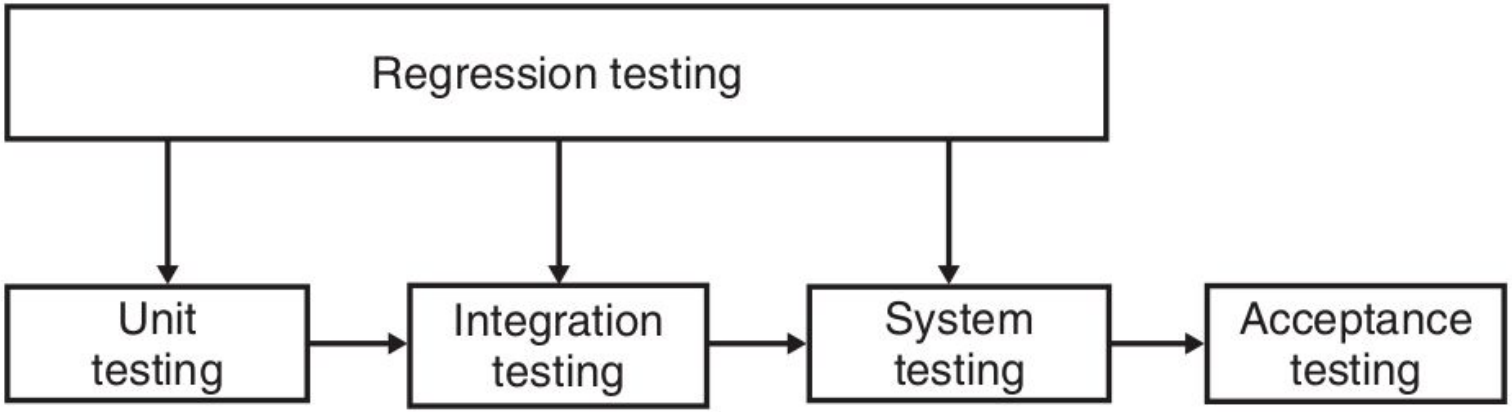
- **Reliability tests** are designed to measure the ability of the system to remain operational for long periods of time.
- **Robustness tests** determine how well the system recovers from various input errors and other failure situations.
- **Interoperability tests** determine whether the system can interoperate with other third-party products.

Performance, Scalability, Stress, Load and Stability tests

- **Performance tests**: measure the performance characteristics of the system, for example, throughput and response time, under various conditions.
- **Scalability tests**: determine the scaling limits of the system in terms of user scaling, geographic scaling, and resource scaling.
- **Stress tests**: put a system under stress in order to determine the limitations of a system and, when it fails, to determine the manner in which the failure occurs.
  - Memory leak
- **Load and stability tests**: provide evidence that the system remains stable for a long period of time under full load.

Regression tests

- determine that the system remains stable as it cycles through the integration of other subsystems and through maintenance tasks.



Documentation and Regulatory tests

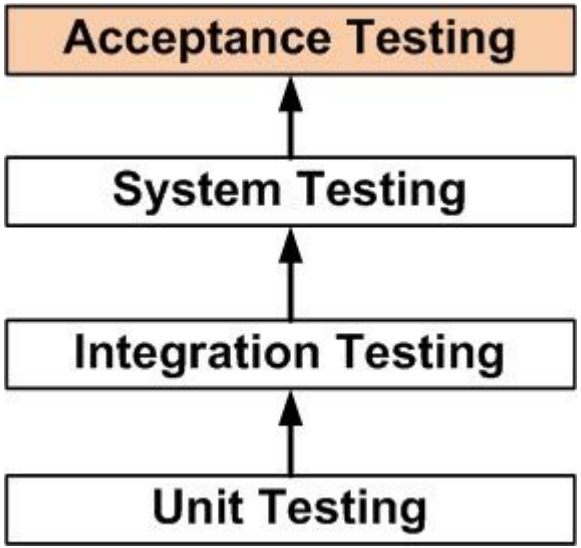
- **Documentation tests** ensure that the system's **user guides** are accurate and usable.
- **Regulatory tests** ensure that the system meets the requirements of **government regulatory** bodies in the countries where it will be deployed.

Acceptance Testing

Acceptance testing is a formal testing conducted to determine whether a system **satisfies its acceptance criteria** — the criteria the system **must satisfy** to be accepted by the **customer**.

There are two categories of acceptance testing:

- User acceptance testing
- Business acceptance testing



Acceptance Testing

- Usually, **Black Box Testing** method is used in Acceptance Testing.
- Testing does not normally follow a strict procedure and is not scripted but is rather ad-hoc.
- Tasks
  - Acceptance Test Plan
  - Acceptance Test Cases/Checklist
  - Acceptance Test Execution

Acceptance Criteria

- Core of any **contractual** agreement is a set of **acceptance criteria**
- **what criteria must the system meet in order to be acceptable?**
- Preferably, criteria must be **measurable**
- “One must understand the meaning of the quality of a system, which is a complex concept. It means different things to different people, and it is highly context dependent.”
- **Finally, Customer’s opinion that prevails.**

Agile Testing

- Adapts to changes very fast
- Not plan-driven, but lives with the project
- Testing is done from the beginning of the project, not just in the end of it
- Emphasizes the team, humanity/human-centeredness and cooperation
- Tests iteratively piece by piece
- Customer is involved even in all testing phases
- Continuous integration
- Automation of testing at least in unit-testing

It is more important to find bugs than to write comprehensive documents

Test Levels and Agile

- Unit testing: Test Driven Development
- Integration testing: continuous integration
- System testing: mainly roles
- Acceptance testing:
  - Customer tests single functionalities as soon as these are available
  - All functionalities included in sprint are tested during the last few days of the sprint
  - Acceptance in sprint review
- Not the means for working phase-by-phase, but giving responsibility

Integrating Testing into Agile Development

- There’s not one single prescription
- Agile methodologies promote short release cycles
- Classic testing model does not work on continuous integration context
- A new approach to testing
- Testers have to reinvent themselves and their craft

Integrating Testing into Agile Development

- Iterate on planning
  - Keep pace with development
- Iterate on testing
  - Manage risk so that the most important features and actions are well-tested
- Iterate on quality
  - Measure your quality
  - Improve it in the next iteration
- Early involvement by testers
  - Testers need to work directly with product owners
  - Must understand the user needs
  - Get deep understanding of the user stories

Integrating Testing into Agile Development

- Focus testing on features that make a difference to users
  - No requirements to analyze
  - No time for comprehensive test plans
- Learn what the users need
  - Make sure those features work
- Ensure that testers take the lead
  - Take responsibility for application quality
  - Become an interface to the user community
  - Work with developers as equal partners
- Enable testers to work side-by-side with developers and customers
  - Testers need to know about tactical development decisions
  - This is especially important in Agile projects without formal requirements

Integrating Testing into Agile Development

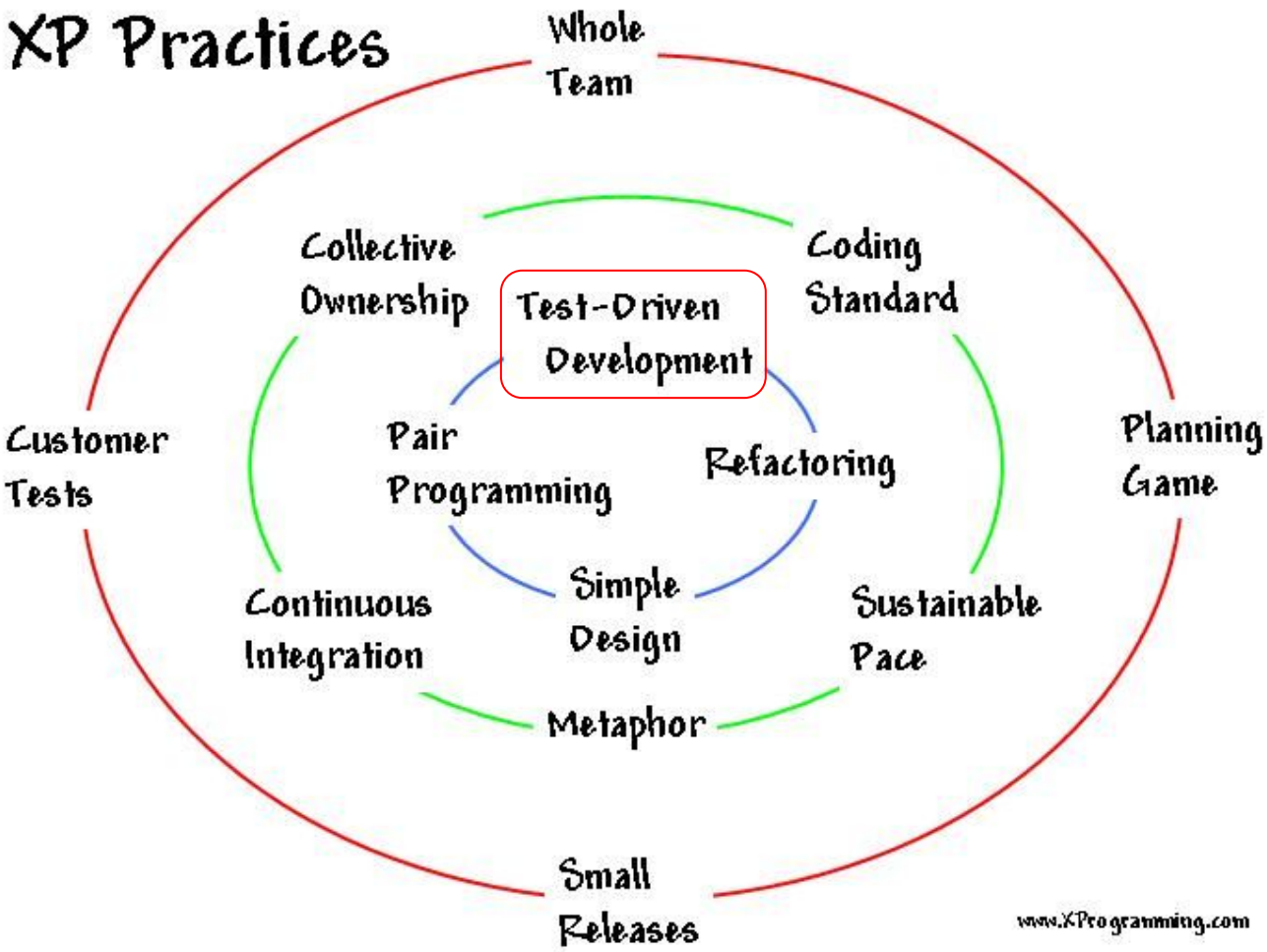
- Ensure that testers take the lead on the building and execution of functional, regression, and acceptance test cases
- Independence from development helps ensure objective evaluation
- Enable testers and developers to make coding decisions together
- **Helps prioritize testing**
- Guide development in decision-making

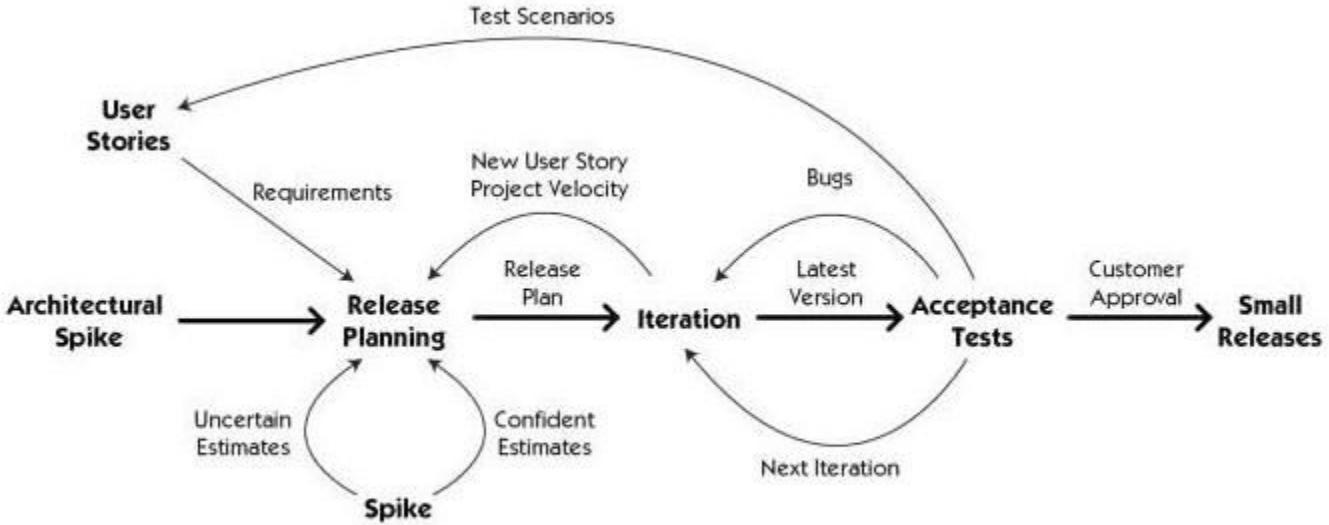
Integrating Testing into Agile Development

- Make automation an integral part of testing
  - Agility and automation work hand in hand
  - Accelerate testing to show agility
- Data collection and analysis essential
  - Feature backlog
  - Failure/Issue management
  - Technical Debt management
  - Logging
  - Metrics
  - Commit comments
  - Time trackers, ...

Integrating Testing into Agile Development

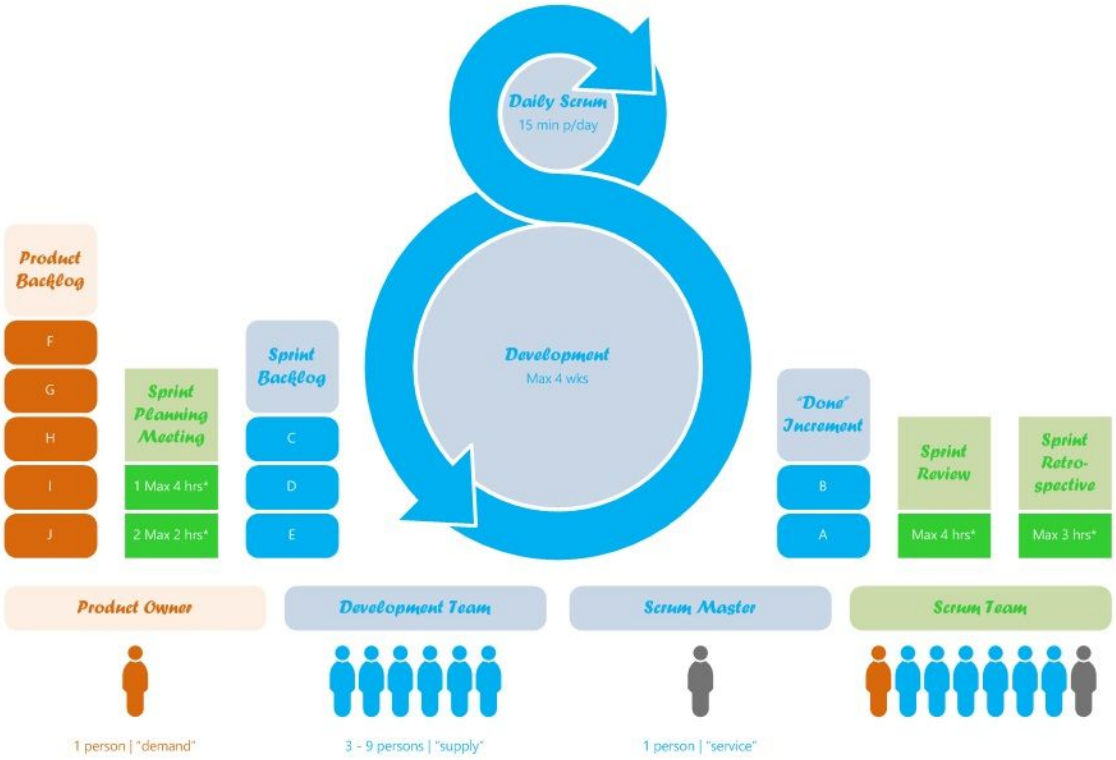
- Tools are essential for speed and flexibility
- Tools accelerate repetitive manual processes
- Tool overhead can’t outweigh benefits
  - Easy to learn and use
  - Provide easily digestible information
- Testing automation
  - Test management
  - Automated functional and regression tests
  - Automated load testing – valuable for assessing design and coding practices


XP Practices
Whole Team
Collective Ownership
Coding Standard
Test-Driven Development
Customer Tests
Pair Programming
Refactoring
Planning Game
Continuous Integration
Simple Design
Sustainable Pace
Metaphor
Small Releases
www.XProgramming.com



A visual process model for XP

Scrum Overview



* Duration of this event depends on the duration of the Sprint

Testers in Scrum

- Daily Scrum-meeting
- Keeping up-to-date in the progress of developers’ work
- Testing (includes parallel test design and test run)
- Taking care of test coverage
- Clearing up defects and reporting
- Maintaining tests and test automation
- Avoiding of test lack (quality debt)
    - Keep up with the pace of the developers
    - Avoid the attitude: : “rest of the tests can be done later”

TESTING PROCESS

Product backlog, sprint

Requirements, functionalities & user stories

List of test cases

Functional testing and exploratory testing

Testers

Developers

Day1 Start

Day 2-17 Implement functionality

Day 18-20 Fix defects

Planning of agile testing

- Start with a light overall test planning
- Choose the proper tools
- Create a testing environment
- Prioritize things to be tested in each iteration (sprint)
- Create test cases for the first iteration on the agreed (light) accuracy level
- Outline tests for next iterations
- Automate testing

What agile testing is not?

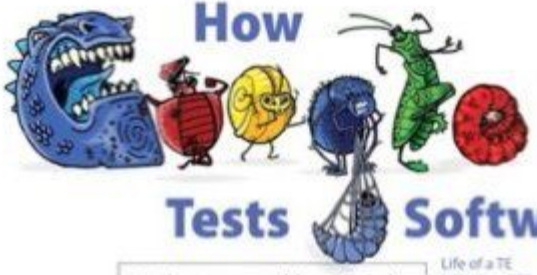
- A separate phase at the end of software development project
- Unsystematic
- Documentation free
- Random
- Uncontrolled
- Straightforward operation without feedback

Test for Value

- Deliver the right it is even more important than doing it right
- Fit customer true and implicit intention than fit a specification
- It is not because a customer said something that we must do
- It is not because a customer did not say something that we must not do

Testing is left mandatory but vague because its level and coverage are not instructed.

(Timperi, O. (2004). An Overview of Quality Assurance Practices in Agile Methodologies.)



How Tests Software

Help me test like Google

Life of a TE
Life of an SET
Interviews with Googlers
and more

James Whittaker · Jason Arbon · Jeff Carollo

How Google Test Software

- Software testing is part of a **centralized** organization called **Engineering Productivity**
- “The first piece of advice I give people when they ask for the keys to our success: **Don’t hire too many testers**.”
- **Quality is never “some tester’s” problem.** Everyone who writes code at Google is a tester, and quality is literally the **problem of this collective**.
- Put together a test practice in a **developer-centric** culture

Quality != Test

- 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.
- To merge development and testing so that you cannot do one without the other.
- The first point of escalation is the developer who created the problem, not the tester who didn’t catch it.
- This means that quality is more an act of prevention than it is detection.
- Quality is a development issue, not a testing issue.

Google’s SE Roles

- **Software engineer (SWE)**: write code to create and improve features
  - TDD - write unit tests
  - They must author of tests
- **Software engineer in test (SET)**: write code in service of quality rather then code features
  - focus is on testability and general test infrastructure
  - primary focus is on the developer
  - code quality and risk
  - refactor code to make it more testable
  - write unit testing frameworks and automation
- **Test engineer (TE)**: puts testing on behalf of the user (first) and developers
  - form of automation scripts and code that drives usage scenarios and even mimics the user
  - organize the testing work of SWEs and SETs
  - drive test execution

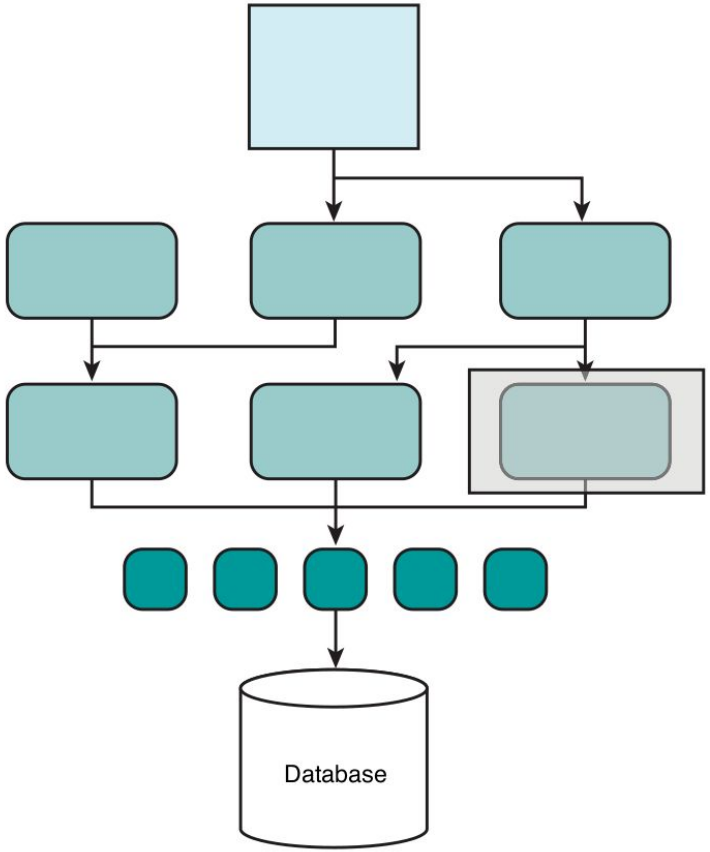
Approach “Crawl, Walk, Run”

- Rarely attempt to ship a large set of features at once
- Build the core of a product and release it the moment it is useful to as large a crowd as feasible, and then get their feedback and iterate.
  - Gmail
    - a product that kept its **beta** tag for **four years**
    - goal of 99.99 percent uptime for a real user’s email data
- Minimum useful product
- Channel sequence
  - Canary: only engineers (developer and testers) on the product
  - Dev: developers use for their day-to-day work.
  - Test Channel: best build of the month; internal dogfood users represents a candidate Beta
  - Beta Channel or Release Channel: survived internal usage; external exposure

Google’s Types of Tests

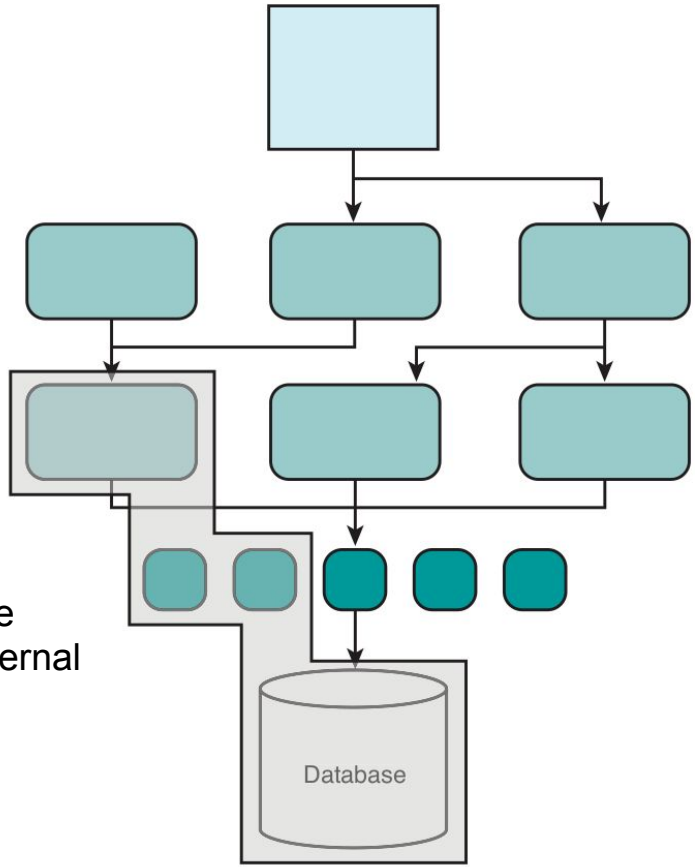
- Uses the language of **small , medium, and large** tests
- Non-distinguishing between code, integration, and system testing
- Emphasizing scope over form
- Mix between automated and manual testing favors the former for all three sizes of tests
- If a problem can be automated and it doesn’t require human cleverness and intuition, then it **should** be automated.
- Great deal of manual testing, both scripted and exploratory
    - under the watchful eye of automation
- Total test coverage should be at least 60% (Test Certified level 5)
- Test coverage from small tests alone should be at least 40% (Level 5)
- **Automation** is the key word!

Small Test



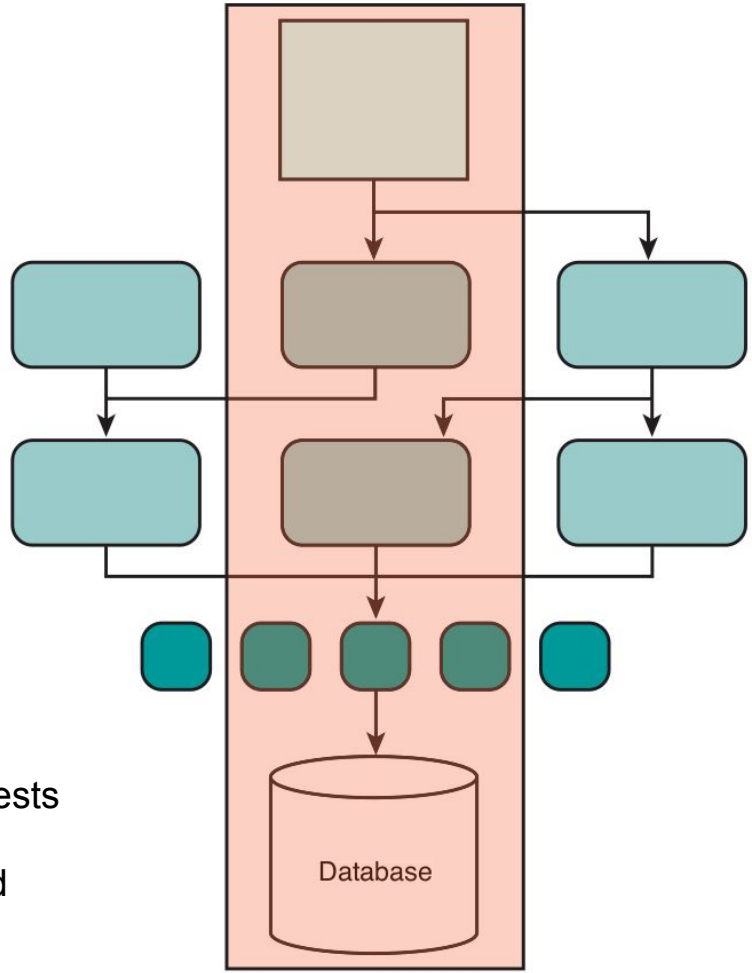
Small test where often only a single function.

Medium Test



Medium tests include multiple modules and can include external data sources.

Large Test



Large and enormous tests include modules that encompass end-to-end execution.

Goals and Limits of Test Execution Time by Test Size

	Small Tests	Medium Tests	Large Tests	Enormous Tests
Time Goals (per method)	Execute in less than 100 ms	Execute in less than 1 sec	Execute as quickly as possible	Execute as quickly as possible
Time Limits Enforced	Kill small test targets after 1 minute	Kill medium test targets after 5 minutes	Kill large test targets after 15 minutes	Kill enormous test targets after 1 hour

Resource Usage by Test Size

Resource	Large	Medium	Small
Network Services (Opens a Socket)	Yes	localhost only	Mocked
Database	Yes	Yes	Mocked
File System Access	Yes	Yes	Mocked
Access to User-Facing Systems	Yes	Discouraged	Mocked
Invoke Syscalls	Yes	Discouraged	No
Multiple Threads	Yes	Yes	Discouraged
Sleep Statements	Yes	Yes	No
System Properties	Yes	Yes	No

Test Engineer at Google

- Test planning and risk analysis
- Review specs, designs, code, and existing tests
- Exploratory testing
- User scenarios
- Test case creation
- Executing test cases
- Crowd sourcing
- Usage metrics
- User feedback

Test planning

- Test plans are the first testing artifact created and the first one to die of neglect
- After the software is released and is successful (or not), few people ask about testing artifacts
- Guiding principles:
  - Avoid prose and favor bulleted lists
  - Don’t bother selling
  - Bigger is not better
  - If it isn’t important and actionable, don’t put it in the plan
  - Make it flow
  - Guide a planner’s thinking
- Planning of tests should put the TE in a position to know what tests need to be written!

Who would you rather have testing your software:

highly paid, on-site exploratory testing **experts** doing their best to **anticipate actual usage** in the **hopes** they find important bugs **or** a **large army of actual users** who are incentivized to find and report **real bugs**?

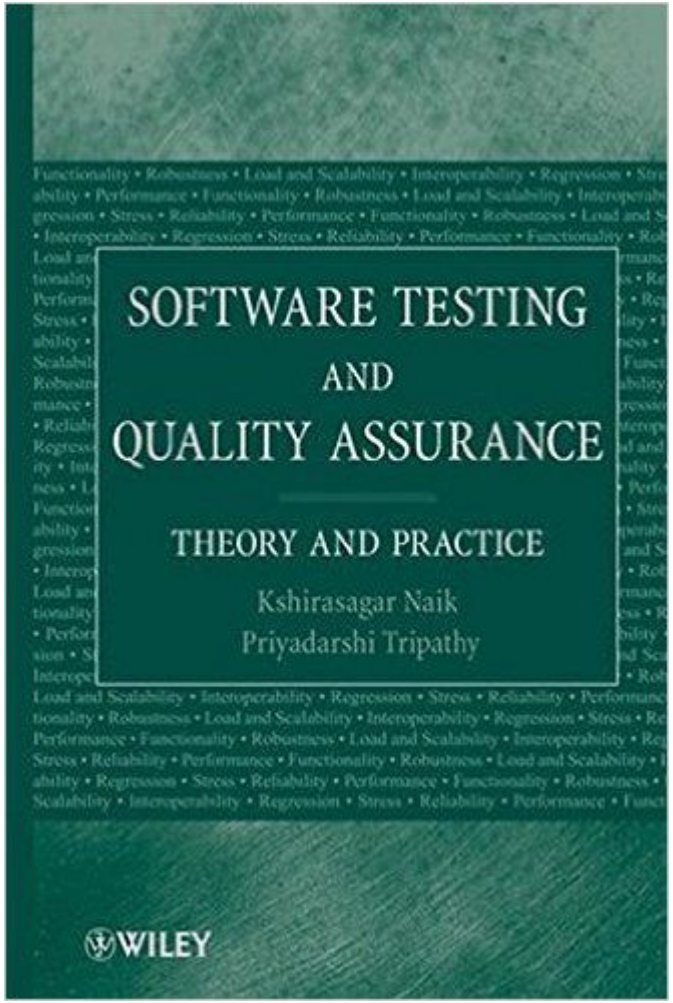
*“Access to **real users** willing to report issues in exchange for **early access** has never been **easier** and with daily or hourly updates, the **actual risks to these users is minimal**. In this new world order, the **TE role needs a complete overhaul**.”*

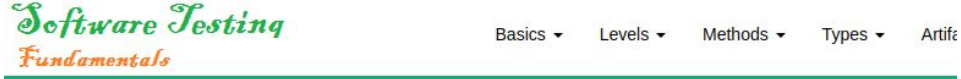
Google’s Message

Put quality in the workflow of every engineer.

“*As a rule, software systems do not work well until they* ***have been used****, and have* ***failed repeatedly****, in* ***real*** *applications.*”

— Dave Parnas


SOFTWARE TESTING
AND
QUALITY ASSURANCE
THEORY AND PRACTICE
Kshirasagar Naik
Priyadarshi Tripathy
WILEY



Do you wish to MASTER the basics of SOFTWARE TESTING? If yes, ...

Welcome to Software Testing Fundamentals (STF) !

Software Testing Fundamentals (STF) is a platform to gain (or refresh) basic knowledge in the field of Software Testing. If we are to 'cliche' it, the site is **of** the testers, **by** the testers, and **for** the testers. Our goal is to build a resourceful repository of *Quality Content* on *Quality*.

YES, you found it: the not-so-ultimate-but-fairly-comprehensive site for software testing enthusiasts. Since most of our articles are based on various resources, references and experiences, we do not claim any originality or authenticity. Browse the site at your own **RISK**.

Get started by going through **Software Testing Basics** like Quality Assurance, Quality Control, Software Development Life Cycle and Software Testing Life Cycle.

And, level up by understanding **Software Testing Levels** like Unit Testing, Integration Testing, System Testing and Acceptance Testing.

Then, check out the various **Software Testing Types** like Smoke Testing, Functional Testing, Usability Testing, Security Testing, Performance Testing, Regression Testing and Compliance Testing.

Also, learn about **Software Testing Artifacts, Defects** and **Software Testing Metrics**.

Finally, enjoy our collection of **Software Testing Resources** like Jokes, Quotes, Jobs, Exercises, Magazines, Blogs and Certifications.

http://softwaretestingfundamentals.com/

Next course

Unit Test and TDD