Digital Academy Thailand

Software Testing Foundation Level

Lecture 2 – Testing Throughout the Software Development Lifecycle

Uwe Gühl

KASETSART UNIVERSITY

มหาวิทยาลัยเกษตรศาสตร์ วิทยาเขตศรีราชา

Kasetsart University Sriracha Campus

Contents

Contents

Software Development Lifecycle Models

- ***Software development lifecycle (SDLC)***:
  The activities performed at each stage in software development, and how they relate to one another logically and chronologically
- A software development lifecycle model describes
  - the types of activity performed at each stage in a software development project, and
  - how the activities relate to one another logically and chronologically.
- There are a number of different software development lifecycle models, each of which requires different approaches to testing.

Software Development and Software Testing

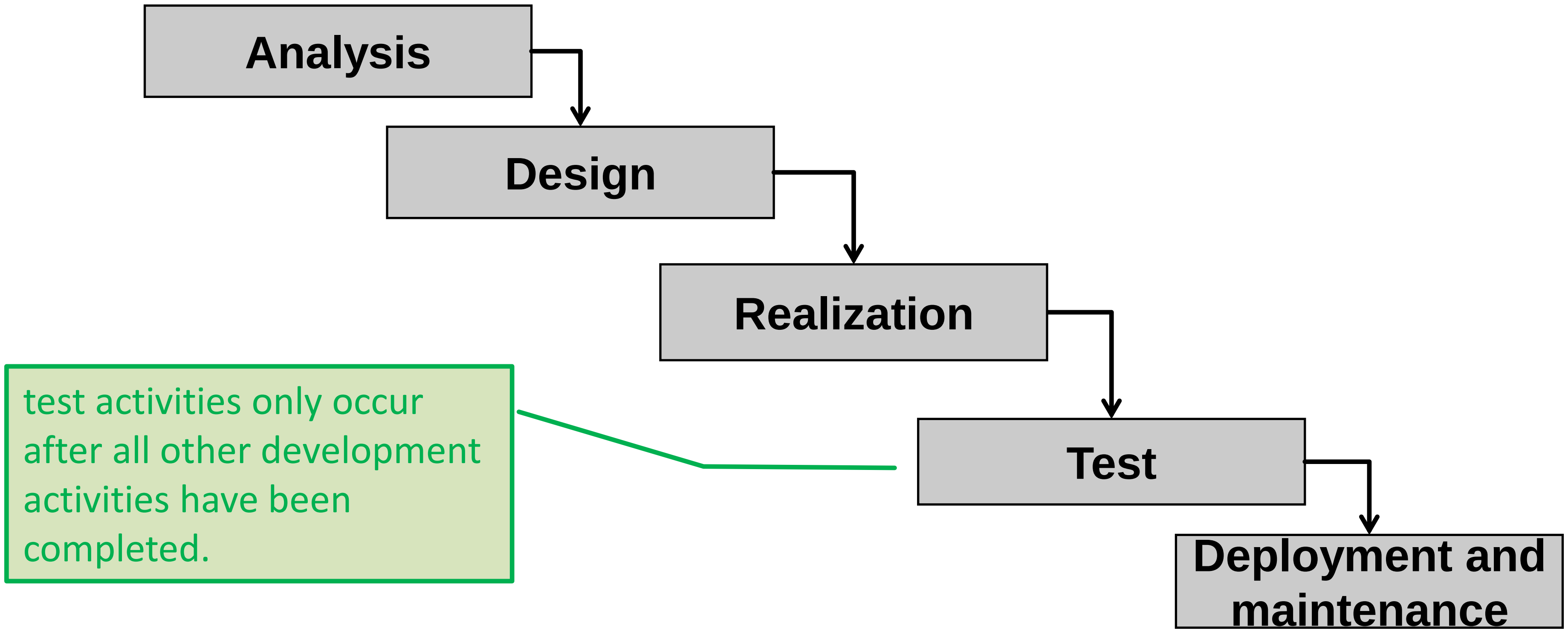
- In any software development lifecycle model, there are several characteristics of good testing:
  - For every development activity, there is a corresponding test activity
  - Each test level has test objectives specific to that level
  - Test analysis and design for a given test level begin during the corresponding development activity
  - Testers
    - ➢ participate in discussions to define and refine requirements and design
    - ➢ are involved in reviewing work products like requirements, design, and user stories
  - Test activities start in the early stages of the lifecycle

Software Development and Software Testing

- Common software development lifecycle models in this context are defined as follows:
  - ***Sequential development model***: A type of software development lifecycle model in which a complete system is developed in a linear way of several discrete and successive phases with no overlap between them.
  - Iterative and incremental development models
    - ➢ ***Iterative development model***: A type of software development lifecycle model in which the component or system is developed through a series of **repeated cycles.**
    - ➢ ***Incremental development model***: A type of software development lifecycle model in which the component or system is developed through a series of **increments.**

Software Development and Software Testing

- Sequential development models
  - Waterfall model


Analysis
Design
Realization
Test
Deployment and maintenance
test activities only occur after all other development activities have been completed.

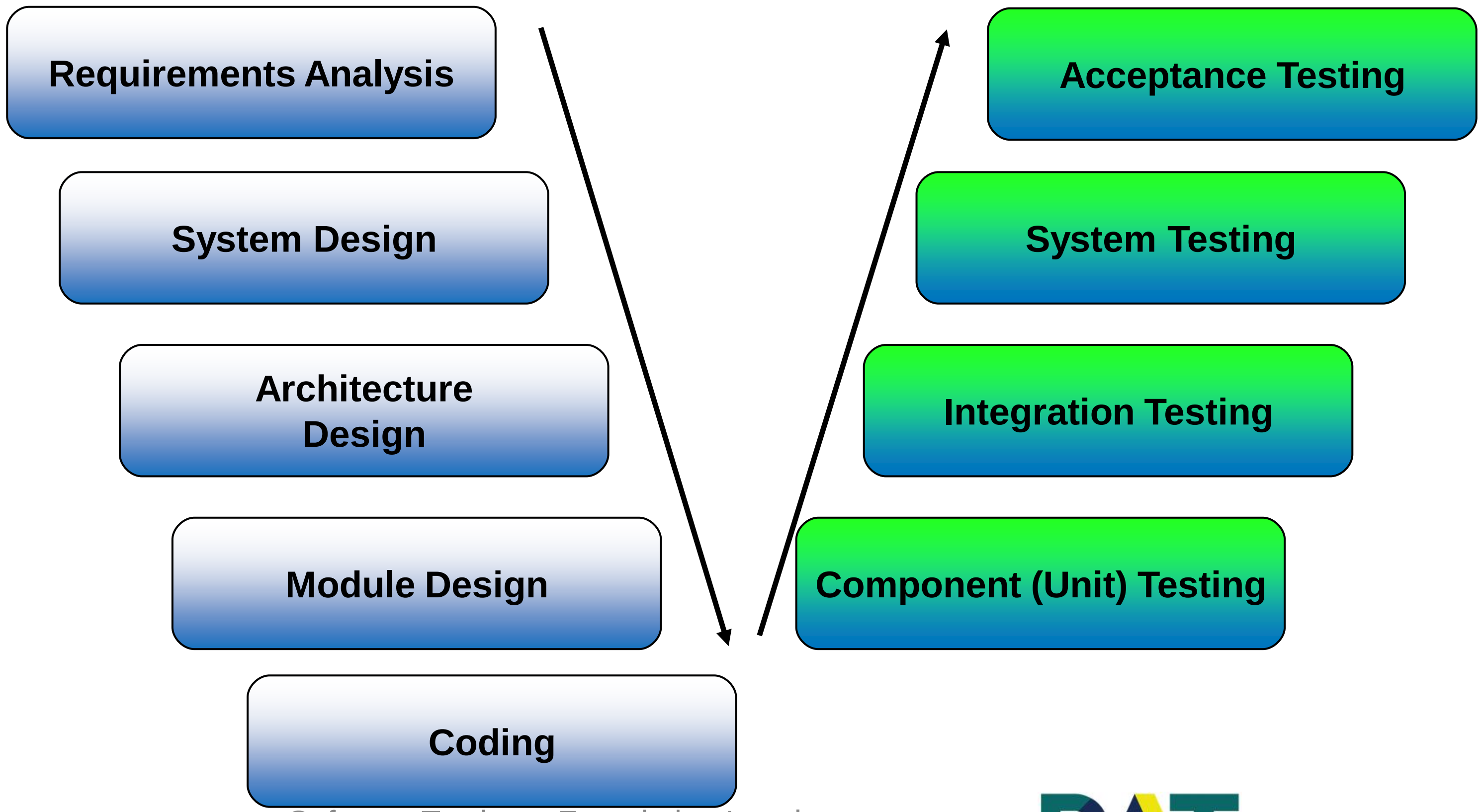
Software Development and Software Testing

- Sequential development models
  - ***V-model***: A sequential development lifecycle model describing a one-for-one relationship between major phases of software development from business requirements specification to delivery, and corresponding test levels from acceptance testing to component testing.

2.2

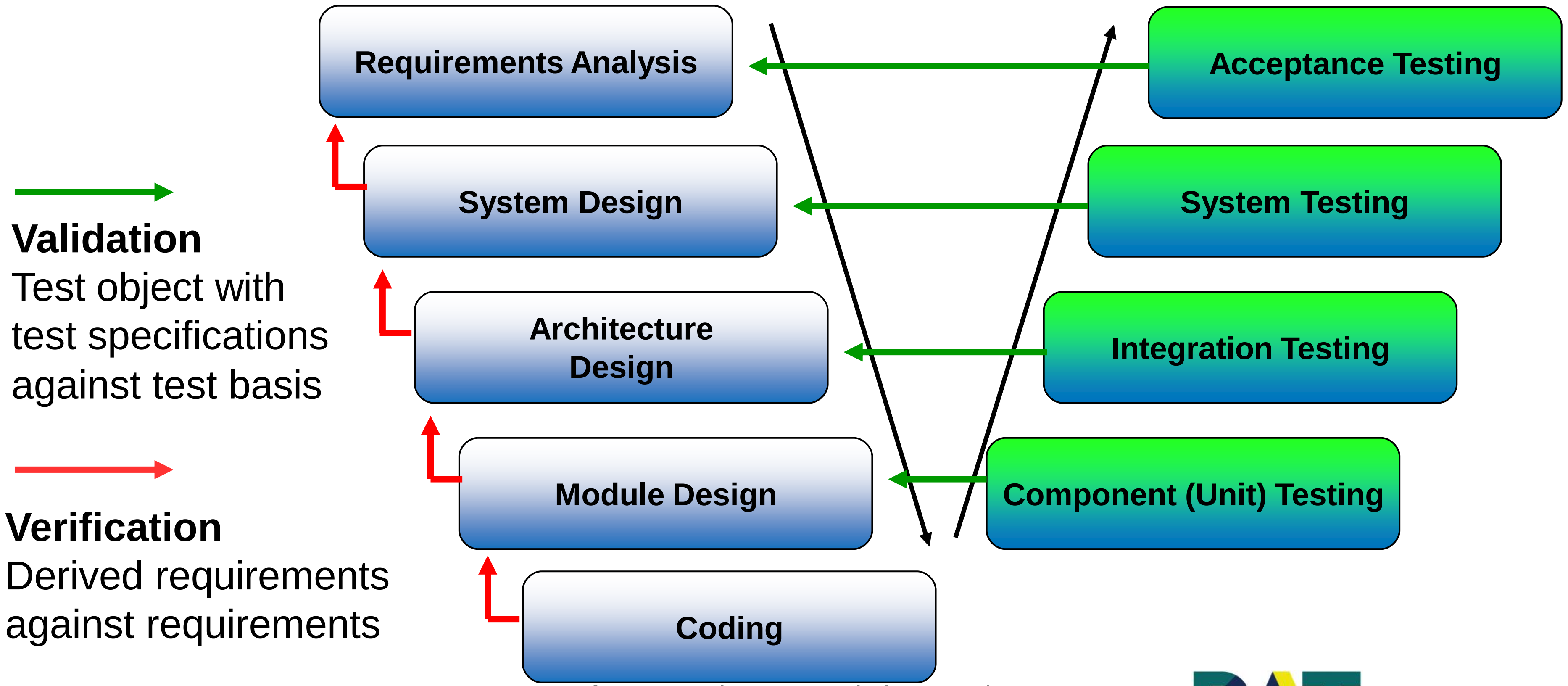
Software Development and Software Testing

- Sequential development models
  - V-model



Software Development and Software Testing

- Sequential development models
  - V-model



Software Development and Software Testing

- Iterative and incremental development models
  - Iterative development:
    - ➢ Groups of features are specified, designed, built, and tested together in a series of cycles
    - ➢ Cycles often have a fixed duration.
    - ➢ Instead of features, changes to features developed in earlier iterations or changes in project scope could be considered in a cycle
    - ➢ Each iteration delivers working software which is a growing subset of the overall set of features

Big plus: usable software in weeks or even days.

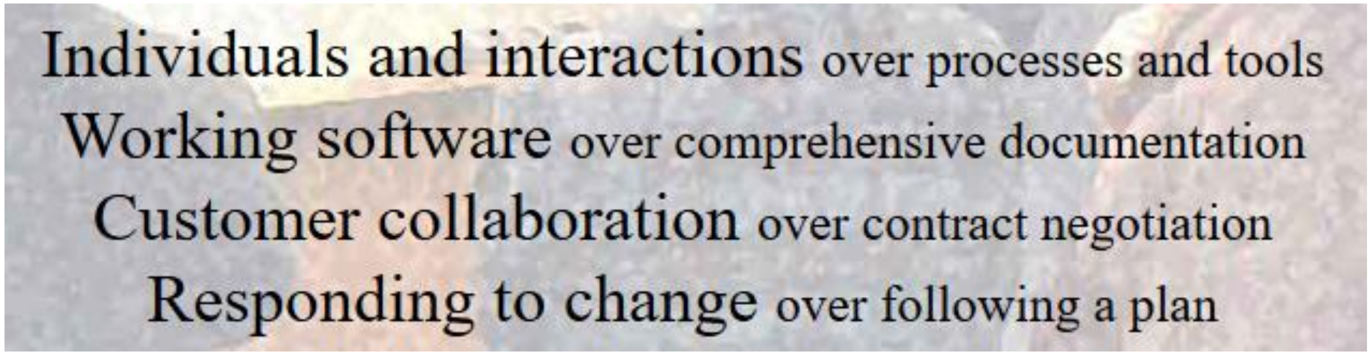
Software Development and Software Testing

- Iterative and incremental development models
  - Incremental development:
    - Establishing requirements, designing, building, and testing a system in pieces
    - Software's features grow incrementally.
    - The feature increments could also be changes like
      - a single change to a user interface screen
      - a new query option.

Software Development and Software Testing

Excurses

- Iterative and incremental development models
  - **Agile software development**: A group of software development methodologies based on iterative incremental development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.
  - **Agile manifesto**



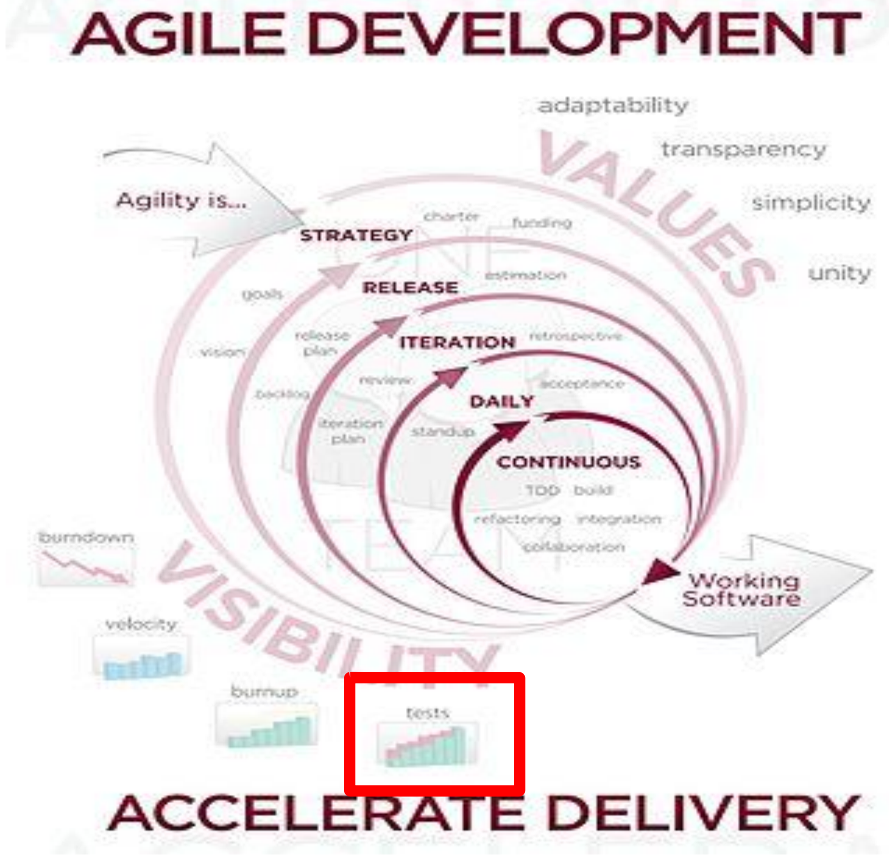
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

Source: https://agilemanifesto.org/

Software Development and Software Testing

Excurses

- Iterative and incremental development models



Software Development and Software Testing

- Iterative and incremental development models
  - Rational Unified Process (RUP)

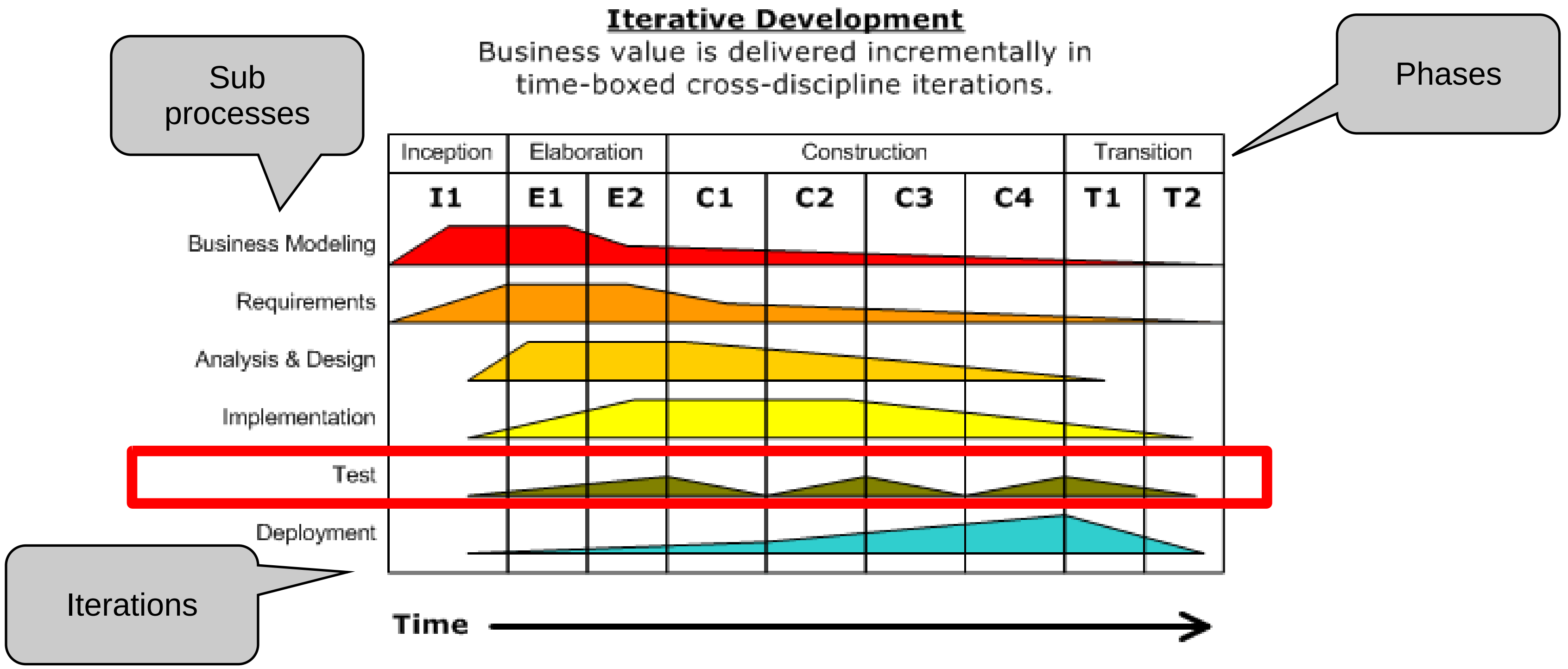


Image source: https://upload.wikimedia.org/wikipedia/commons/1/19/Development-iterative.png

Software Development and Software Testing

- Iterative and incremental development models
  - Scrum

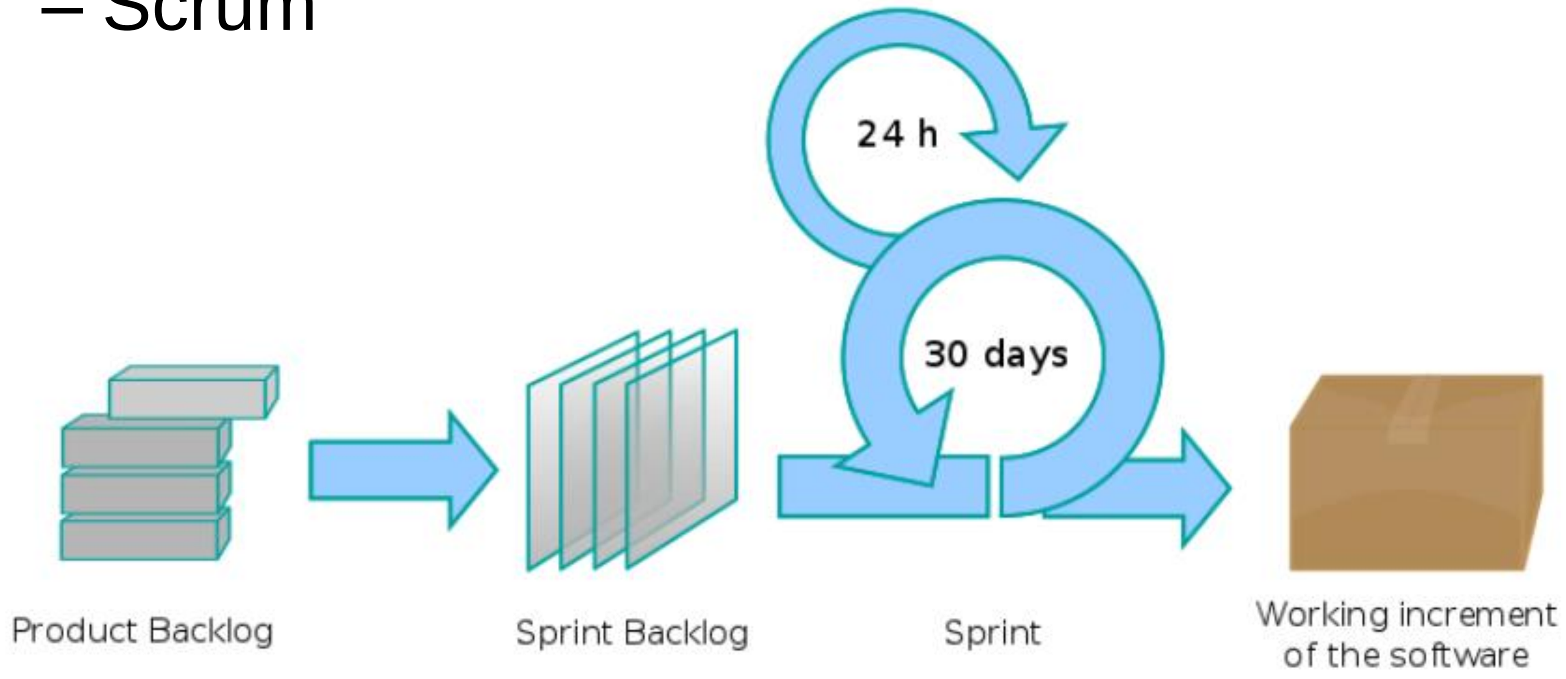


Image source: https://en.wikipedia.org/wiki/File:Scrum_process.svg

Software Development and Software Testing

- Iterative and incremental development models
  - Kanban

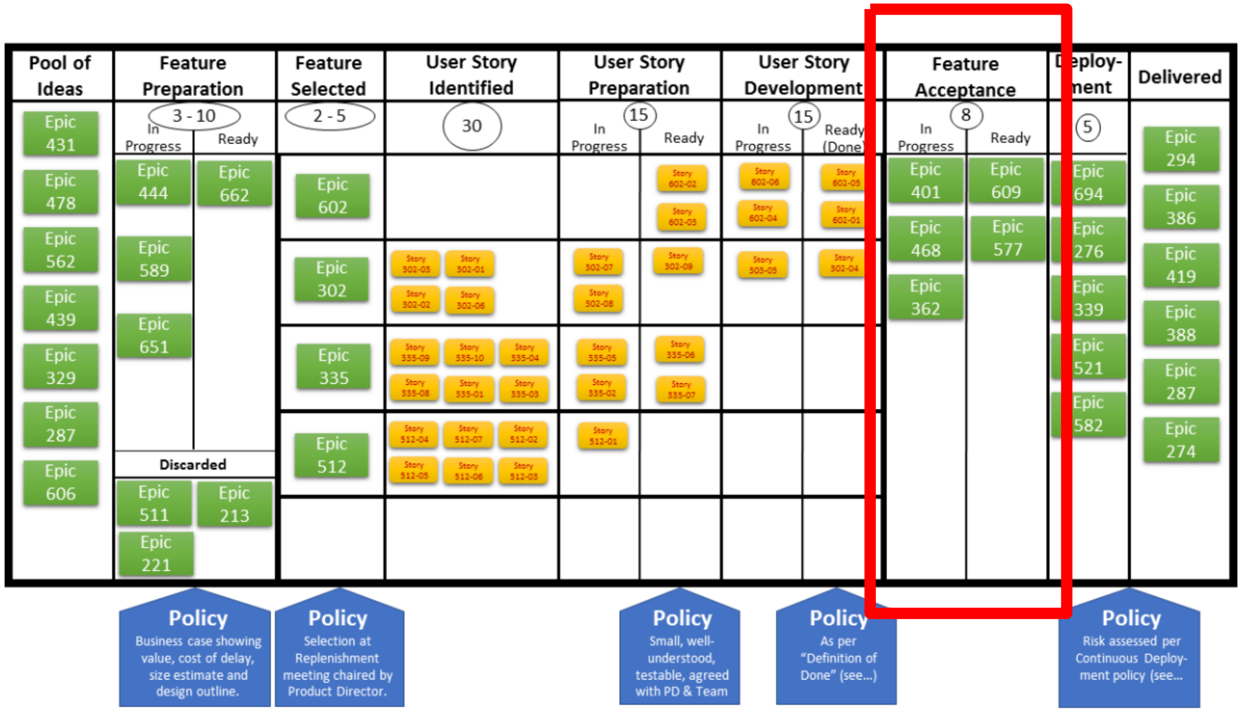


Image source: https://commons.wikimedia.org/wiki/File:Sample_Kanban_Board.png

Software Development and Software Testing

- Iterative and incremental development models
  - Spiral

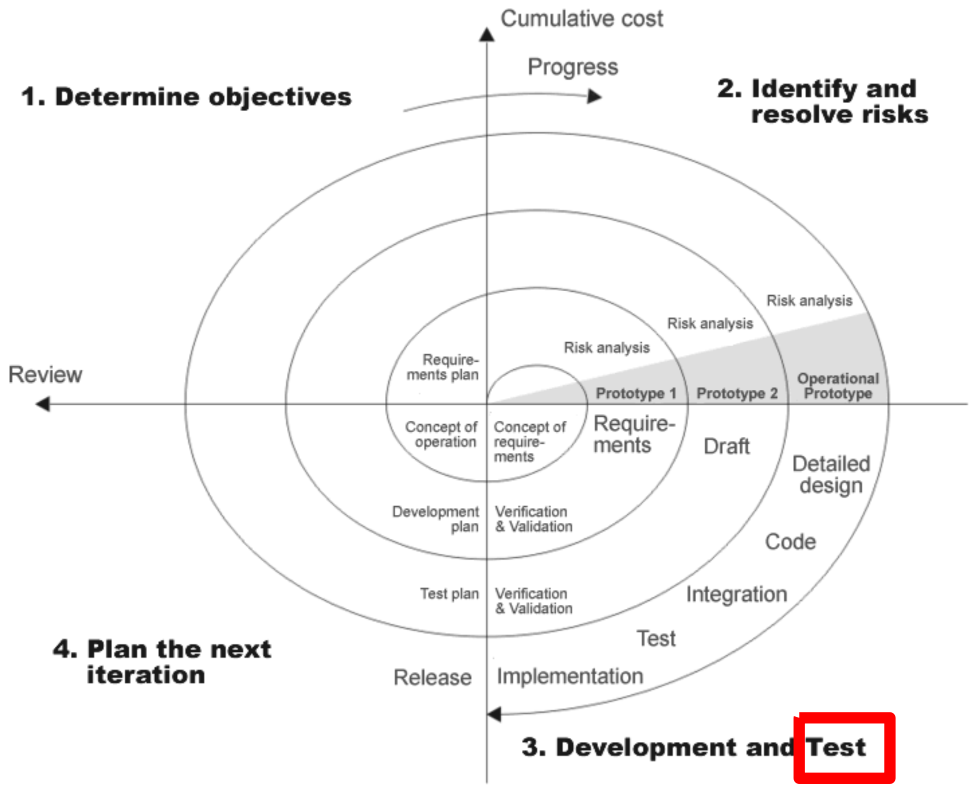


Image source: https://commons.wikimedia.org/w/index.php?curid=9000950

Software Development and Software Testing

- Iterative and incremental development models
  - Often overlapping and iterating test levels throughout development
  - Each feature to be tested at several test levels as it moves towards delivery
  - Continuous delivery
    - ➢ Iteration by iteration or major/minor releases
    - ➢ Typically multiple automated tests required
    - ➢ Importance of regression test increases as a system is growing
  - Test organization flexible within self-organizing teams

Software Development and Software Testing

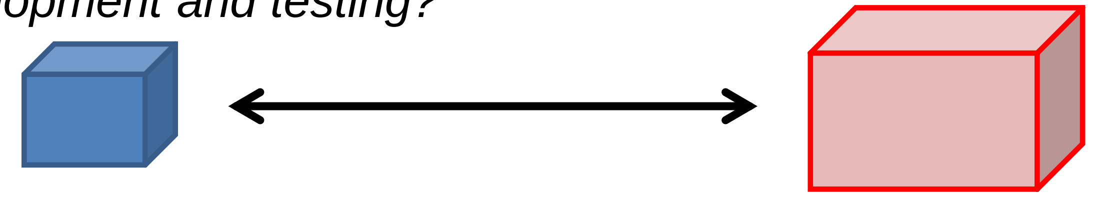
Excurses

- Iterative and incremental development models
  - Agile testing involves testing from the customer point of view as early as possible – depending on availability and stability of code.
  - Test automation plays a central role.
    Typical test execution proceeding after delivery:
    - ➢ (Automated) smoke test/sanity check
    - ➢ Execution of automated regression test suite
    - ➢ Execution of manual tests concerning new implemented user stories/retest of bug fixes
    - ➢ Extending automated test suite

Software Development Lifecycle Models in Context

- Selecting and adapting a software development lifecycle model should consider the context of project and product characteristics
  - project goal,
  - type of product being developed,
  - business priorities (e.g., time-to-market),
  - identified product and project risks,
  - possible organizational and cultural issues

Same development and testing?



Internal administrative system

Safety critical system

Software Development Lifecycle Models in Context

Commercial off-the-shelf (COTS)
***(Synonym:** off-the-shelf software):*
A type of product developed in an identical format for a large number of customers in the general market.

- Example: Test organization for a COTS software product into a larger system.
  – Purchaser performs interoperability testing at the system integration test level (integration to customer system infrastructure) 2.2
  – Purchaser supports at the acceptance test level (functional and non-functional, user acceptance and operational acceptance) 2.3

Software Development Lifecycle Models in Context

- Software development models to be adapted to the context of project and product characteristics; possible reasons:
  - Difference in product risks of systems (complex or simple project)
  - Many business units are part of a project or program (combination of sequential and agile development)
  - Short time to deliver a product to the market (merge of test levels and/or integration of test types in test levels)
- Internet of Things (IoT) systems
  - consist of different objects like devices, products, and services,
  - apply separate software development lifecycle models for each object.

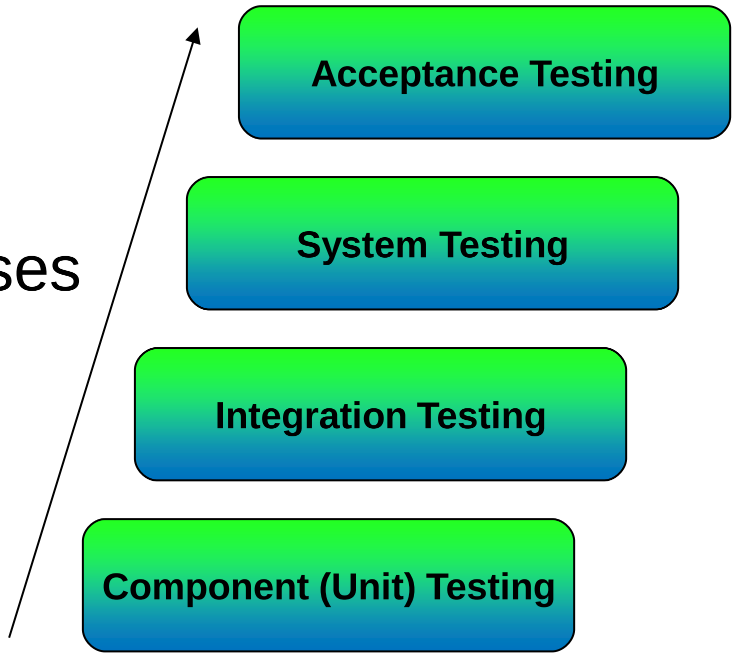
Summary

- Software development lifecycle models
  - Sequential development models
    Complete system is developed, all phases with no overlap between them
    Examples: Waterfall model, V-model
  - Iterative and incremental development models
    Growing system in fixed cycles, after each iteration there is working software
    Examples: RUP, Scrum, Kanban, Spiral
- Good ideas for testing:
  - Development activity ⇔ Test activity
  - Each test level has test objectives

Contents

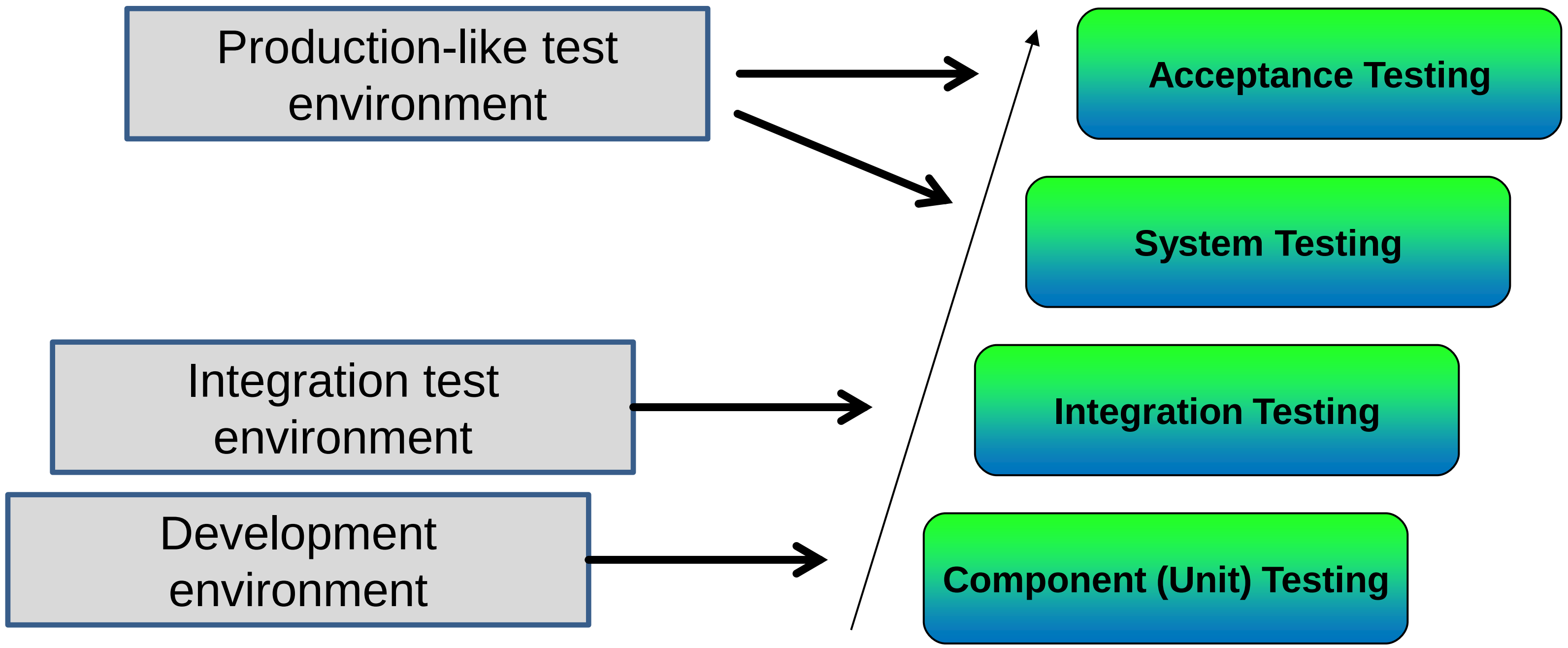
Test Levels

- ***Test levels***:
  A specific instantiation of a test process.
  - are characterized by the following attributes:
    - ➢ Specific objectives
    - ➢ Test basis, referenced to derive test cases
    - ➢ Test object
    - ➢ Typical defects and failures
    - ➢ Specific approaches and responsibilities

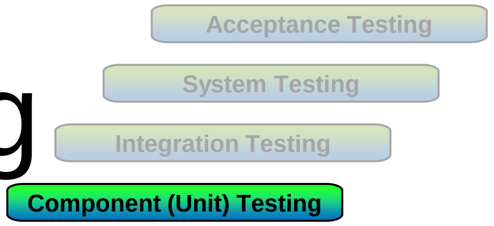


Test Levels

- For every Test level a fitting test environment is required, e.g.



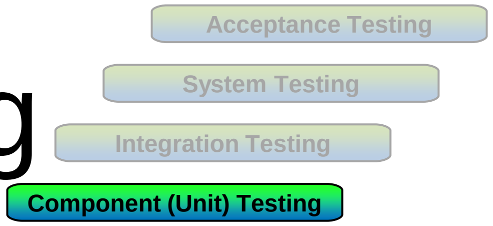
Component Testing



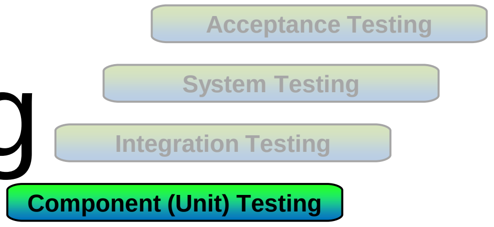
Component testing
*(**Synonyms:** module testing, unit testing)*: A test level that focuses on individual hardware or software components.

- Objectives
  - Verifying if the component works as designed and specified
    - ➢ Functionality (e.g., correctness of calculations),
    - ➢ Non-functional characteristics (e.g., searching for memory leaks), and
    - ➢ Structural properties (e.g., decision testing).
  - Finding defects in the component
  - Preventing defects from escaping to higher test levels
  - Reducing risk
  - Building confidence in the component’s quality

Component Testing



- Objectives
  - In agile projects:
    - ➢ Automated component regression tests play a key role
    - ➢ Ensure changes have not broken existing components
  - Required, depending on project:
    - ➢ mock objects,
    - ➢ ***service virtualization***:
      A technique to enable virtual delivery of services which are deployed, accessed and managed remotely.
    - ➢ harnesses,
    - ➢ stubs, and
    - ➢ drivers.



Component Testing

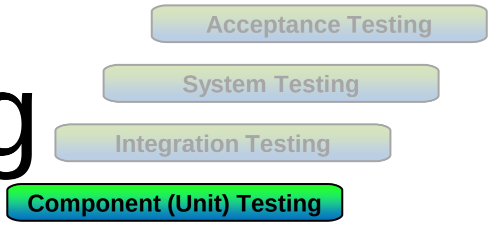
- Test basis
  - Detailed design
  - Code
  - Data model
  - Component specifications
- Test objects
  - Components, units or modules
  - Code and data structures
  - Classes
  - Database modules

Component Testing

- Typical defects and failures
  - Incorrect functionality (e.g., not as described in design specifications)
  - Data flow problems
  - Incorrect code and logic
- Specific approaches and responsibilities
  - Tests executed by
    - ➢ developer who wrote the code,
    - ➢ other developers in the project.

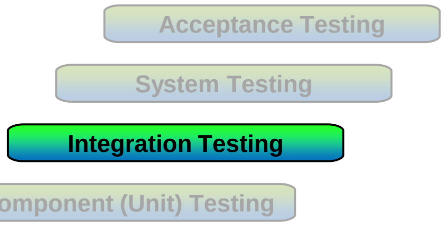
Often with no formal defect management

Component Testing



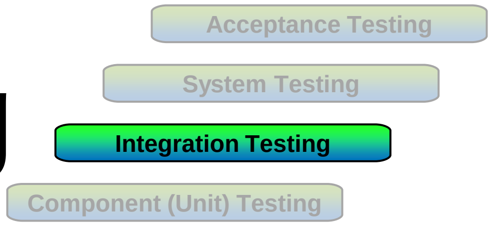
- Test driven development
  - Prepare and automate test cases before coding
  - Based on very short development cycles
  - Proceeding
    - ➢ First write an (initially failing) automated test case that defines a desired improvement or new function.
    - ➢ Second produce the minimum amount of code to pass that test.
    - ➢ Finally refactor the new code to acceptable standards.
- Related:
  - Acceptance test–driven development (ATDD)
  - Behavior-driven development (BDD)

Integration Testing

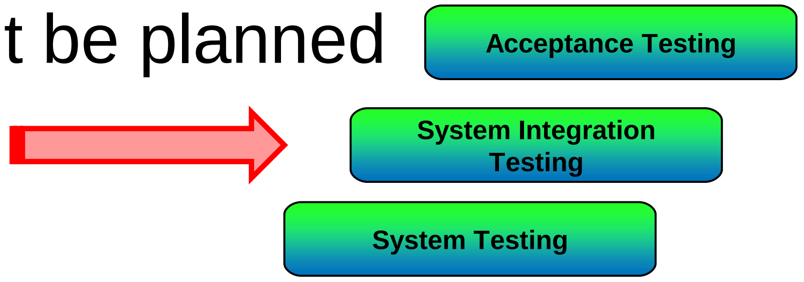


- ***Integration***: The process of combining components or systems into larger assemblies.
- ***Integration testing***: A test level that focuses on interactions between components or systems.
  - ***Component integration testing*** (**Synonym:** *link testing*): Testing in which the test items are interfaces and interactions between integrated components.
  - ***System integration testing***: A test level that focuses on interactions between systems.
- Integration and Integration test have different objectives
- An integration strategy should consider efficient testing

Integration Testing

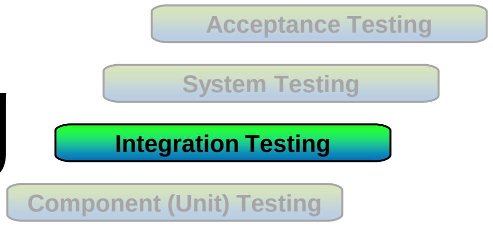


- System integration testing:
  - Additional test level might be planned



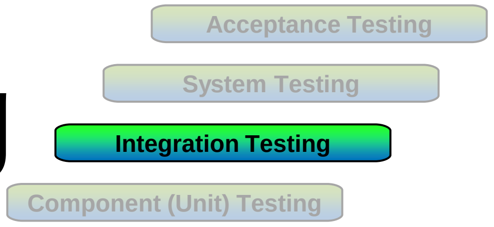
  - Challenging: test of external interfaces
    - ➢ How to ensure that test-blocking defects in the external organization’s code are resolved?
    - ➢ Arranging for test environments
    - ➢ Support for negative test, e.g.,
external system is not working, to be simulated

Integration Testing



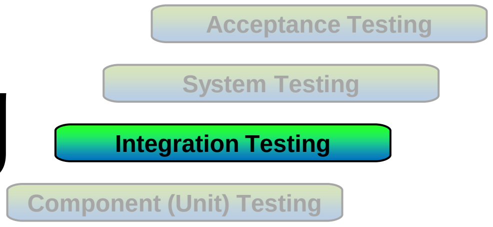
- Objectives
  - Verifying whether the functional and non-functional behaviors of the interfaces are as designed and specified
  - Finding defects (which may be in the interfaces themselves or within the components or systems)
  - Preventing defects from escaping to higher test levels
  - Building confidence in the quality of the interfaces
  - Reducing risk

Integration Testing



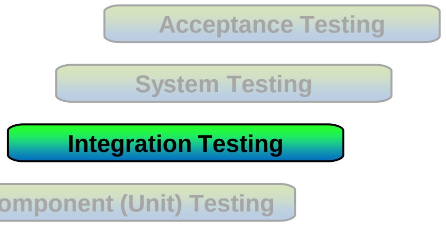
- Objectives
  - In case: Supporting ***continuous integration***:
    A software development procedure merging, integrating and testing all changes as soon as they are committed within an automated process.
  - Automated integration regression tests as part of continuous integration to ensure that existing interfaces, components, or systems are not broken

Integration Testing



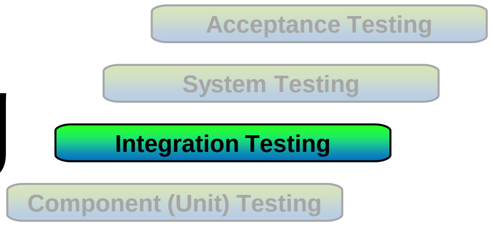
- Test basis
  - Use cases
  - Workflows
  - Architecture at component or system level
  - Software and system design
  - Interface and communication protocol specifications
  - External interface definitions
  - Sequence diagrams
- Test objects
  - Subsystems
  - Databases
  - Infrastructure
  - Interfaces
  - APIs
  - Microservices

Integration Testing



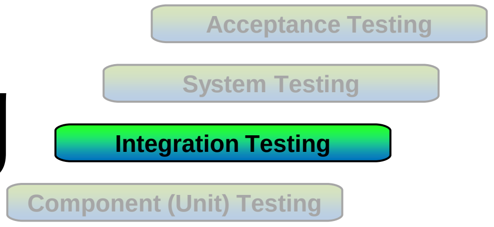
- Typical defects and failures
  - for component integration testing
    - ➢ Incorrect data, missing data, or incorrect data encoding
    - ➢ Incorrect sequencing or timing of interface calls
    - ➢ Interface mismatch
    - ➢ Failures in communication between components
    - ➢ Unhandled or improperly handled communication failures between components
    - ➢ Incorrect assumptions about the meaning, units, or boundaries of the data being passed between components

Integration Testing

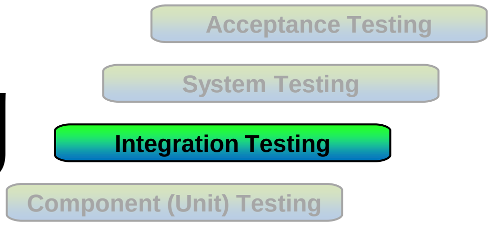


- Typical defects and failures
  - for system integration testing
    - ➢ Inconsistent message structures between systems
    - ➢ Incorrect data, missing data, or incorrect data encoding
    - ➢ Interface mismatch
    - ➢ Failures in communication between systems
    - ➢ Unhandled or improperly handled communication failures between systems
    - ➢ Incorrect assumptions about the meaning, units, or boundaries of the data being passed between systems
    - ➢ Failure to comply with mandatory security regulations

Integration Testing

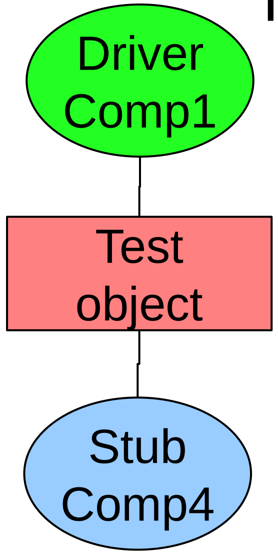


- Proceeding
  - Testing is depending from integration strategy
    - ➢ Incremental
      A small number of additional components or systems at a time
    - ➢ “big bang”
      Integrating all components or systems in one single step
  - Following the incremental approach, a solution is required for components/systems not in place yet

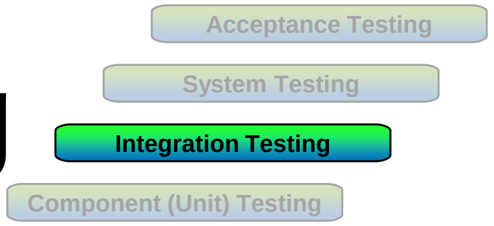


Integration Testing

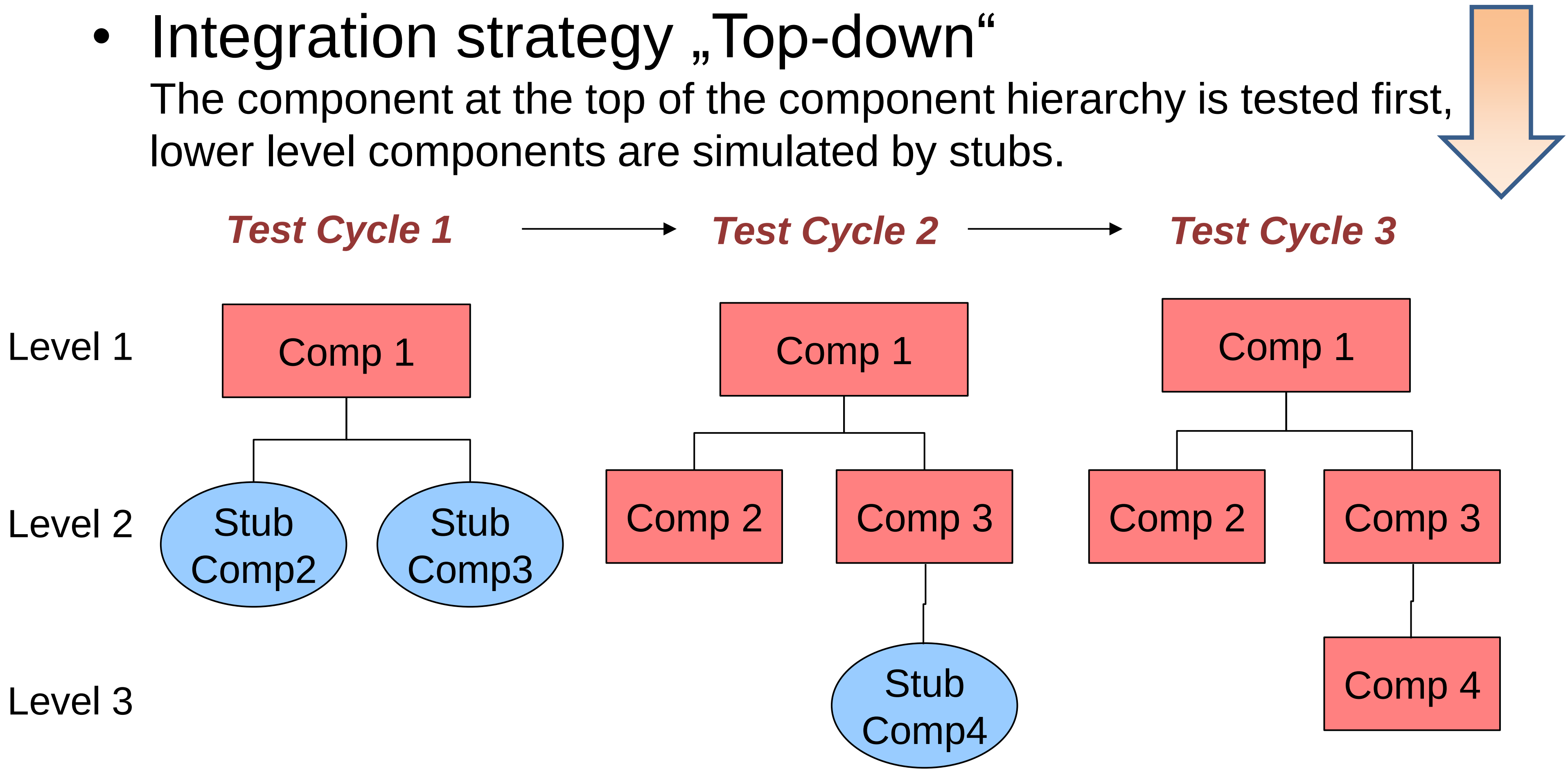
- Proceeding
  - Required: ***Test harness***:
    A test environment comprised of stubs and drivers needed to execute a test suite.
    - ➢ ***Driver*** (***Synonym:*** *test driver)*: A temporary component or tool that replaces another component and controls or calls a test item in isolation.
    - ➢ ***Stub***: A skeletal or special-purpose implementation of a software component, used to develop or test a component that calls or is otherwise dependent on it. It replaces a called component.



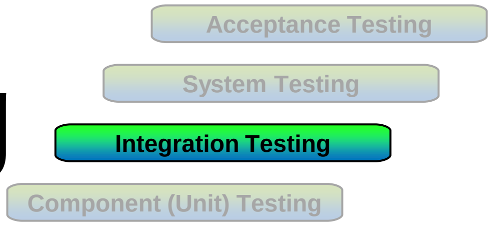
Integration Testing



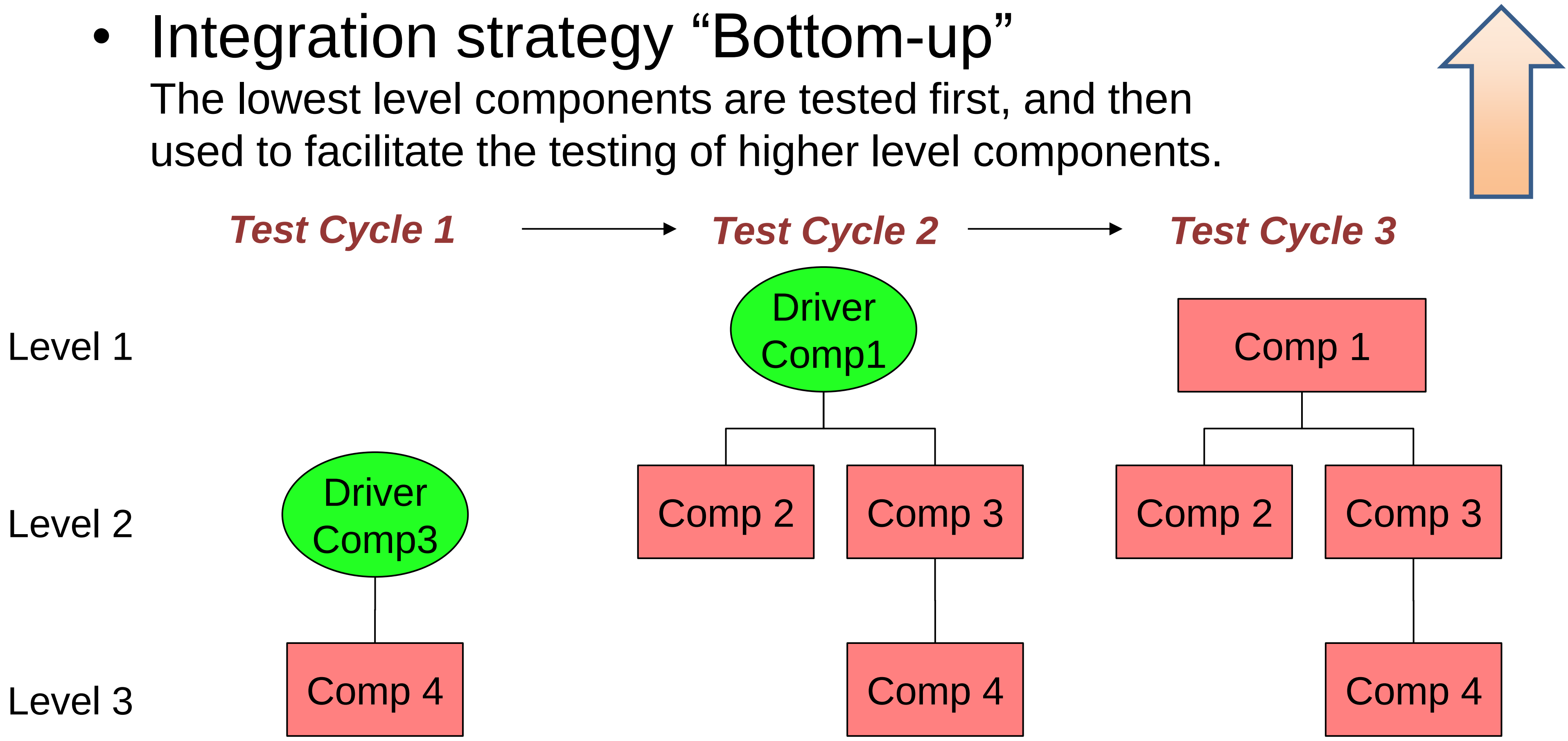
- Integration strategy „Top-down“
  The component at the top of the component hierarchy is tested first, lower level components are simulated by stubs.



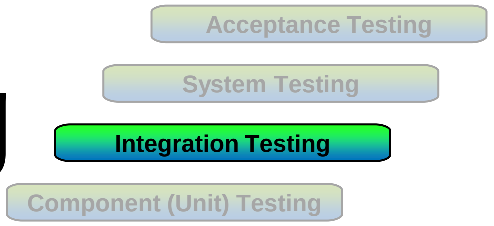
Integration Testing



- Integration strategy “Bottom-up”
  The lowest level components are tested first, and then used to facilitate the testing of higher level components.

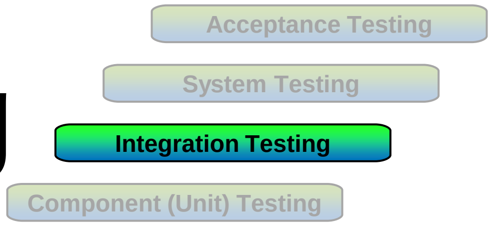


Integration Testing



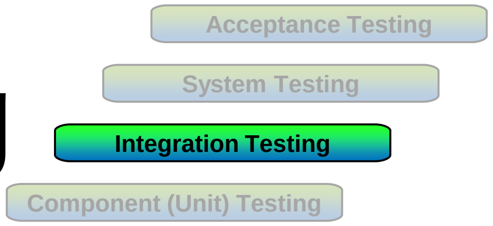
- Integration strategy “Hardest first”
  - The most critical components are developed and tested first
  - Drivers and stubs are required, means effort
  - Feasibility is checked at an early stage reducing the risk that the project fails

Integration Testing



- Integration strategy
  “Function oriented”/”Transaction oriented”
  - Integrate the components that realize a common functionality/transaction
  - Drivers and stubs are required, means effort
  - User/Data oriented approach
    reducing the risk that the project fails

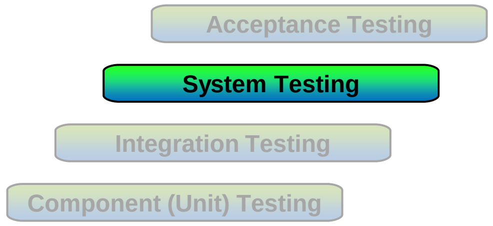
Integration Testing



- Integration strategy “Ad hoc”
  - Integrate the components that are available
  - Drivers and stubs are required, means effort
  - Integration and integration testing could start early

Integration Testing

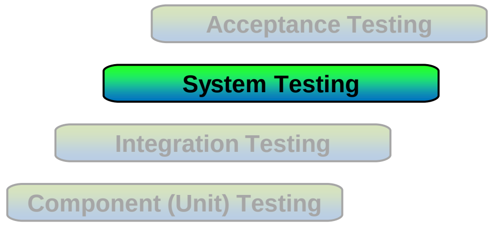
- Integration strategy “Big bang”
  – Integrate the components all at once
  – No drivers and stubs are required
  – Integration and integration testing could start only at a later stage
  – Localization of defects very difficult
  – Risky strategy



System Testing

System testing:
A test level that focuses on verifying that a system as a whole meets specified requirements.

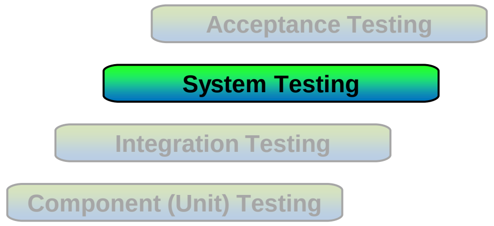
- Objectives
  - Validating that the system is complete and will work as expected
    - ➢ Consider end-to-end tasks
  - Verifying whether the functional and non-functional behaviors of the system are as designed and specified
  - Finding defects
  - Preventing defects from escaping to higher test levels or production
  - Building confidence in the quality of the system as a whole
  - Reducing risk



System Testing

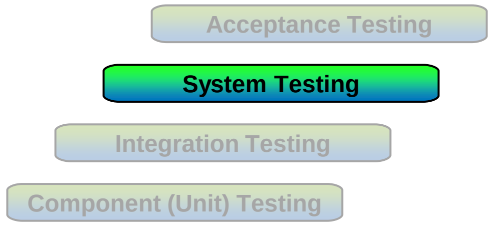
- Objectives – depending on context
  - verifying data quality
  - automated system regression tests to ensure that changes have not broken existing features or end-to-end capabilities.
- Additional hints
  - System testing
    - ➢ often produces information that is used by stakeholders to make release decisions.
    - ➢ may also satisfy legal or regulatory requirements or standards.
  - The test environment should ideally correspond to the final target or production environment

System Testing



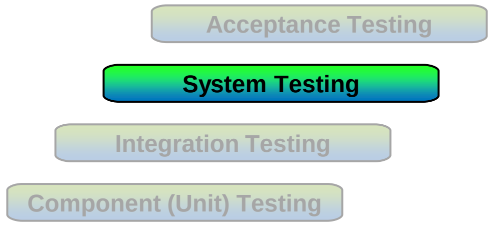
- Test basis
  - System and software requirement specifications (functional and non-functional)
  - Risk analysis reports
  - Use cases
  - Epics and user stories
  - Models of system behavior
  - State diagrams
  - System and user manuals
- Test objects
  - Applications
  - Hardware/software systems
  - Operating systems
  - ***System under test (SUT)***:
    A type of test object that is a system.
  - System configuration and configuration data

System Testing



- Typical defects and failures
  - Incorrect calculations
  - Incorrect or unexpected system functional or non-functional behavior
  - Incorrect control and/or data flows within the system
  - Failure to properly and completely carry out end-to-end functional tasks
  - Failure of the system to work
    - ➢ properly in the system environment(s)
    - ➢ as described in system and user manuals

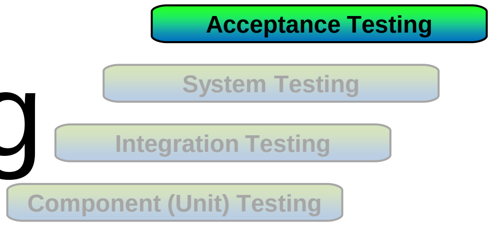
System Testing



- Specific approaches and responsibilities
  - focus on the overall, end-to-end behavior of the system as a whole,
    both functional and non-functional.
  - typically carried out by independent testers who rely heavily on specifications

System Testing

- Specific approaches and responsibilities
  - Risk: Communication issues
    - ➢ Defects in specifications like
      - ❖ missing user stories,
      - ❖ incorrectly stated business requirements.
    - ➢ Impact:
      - ❖ Lack of understanding of system behavior
      - ❖ Disagreements
      - ❖ False positives
      - ❖ False negatives
    - ➢ Mitigation: Early involvement of testers in user story refinement/review



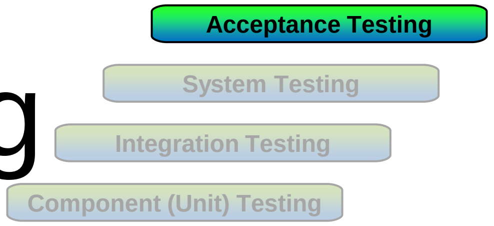
Acceptance Testing

Acceptance testing: A test level that focuses on determining whether to accept the system.

- Objectives:
  - Establishing confidence in the quality of the system as a whole
  - Validating that the system is complete and will work as expected
  - Verifying that functional and non-functional behaviors of the system are as specified
- Finding defects is often not an objective
- Finding a significant number of defects to be considered a major project risk.
- Different forms of acceptance testing are known

Acceptance Testing

- ***User acceptance testing***:
  A type of acceptance testing performed to determine if intended users accept the system.
  - Objective: building confidence that the users can use the system to meet their needs, fulfill requirements, and perform business processes with minimum difficulty, cost, and risk.
  - Real or simulated operational environment.
  - Done by business users (Customers, system users)



Acceptance Testing

- ***Operational acceptance testing***
  (**Synonym:** *production acceptance testing*):
  A type of acceptance testing performed to determine if operations and/or systems administration staff can accept a system.
  - Objective: confidence that the operators or system administrators can keep the system working properly for the users in the operational environment, even under exceptional or difficult conditions.
  - Testing in a (simulated) production environment by
    - ➢ operations,
    - ➢ systems administration staff.

Acceptance Testing

- Operational acceptance testing
  - Scope:
    - ➢ Testing of backup and restore
    - ➢ Installing, uninstalling and upgrading
    - ➢ Disaster recovery
    - ➢ User management
    - ➢ Maintenance tasks
    - ➢ Data load and migration tasks
    - ➢ Periodic checks for security vulnerabilities
    - ➢ Performance testing

Acceptance Testing

- ***Contractual acceptance testing***:
  A type of acceptance testing performed to verify whether a system satisfies its contractual requirements.
  - Objective: compliance has been achieved
  - Basics: Acceptance criteria as defined in the contract for custom-developed software
  - Performed by users or independent testers

Acceptance Testing

- ***Regulatory acceptance testing***:
  A type of acceptance testing performed to verify whether a system conforms to relevant laws, policies and regulations.
  - Objective: compliance has been achieved
  - Basics: any regulations that must be adhered to, like
    - ➢ government regulations,
    - ➢ legal regulations,
    - ➢ safety regulations.
  - performed by users or by independent testers, sometimes audited by regulatory agencies.

Acceptance Testing

- Alpha and beta testing
  - ***Alpha testing***: A type of acceptance testing performed **in the developer's test environment** by roles outside the development organization.
  - ***Beta testing***: A type of acceptance testing performed **at an external site** to the developer's test environment by roles outside the development organization.
  - typically used by developers of commercial off-the-shelf (COTS) software to get feedback from potential or existing
    - ➢ users,
    - ➢ customers, and/or
    - ➢ operators.

Acceptance Testing

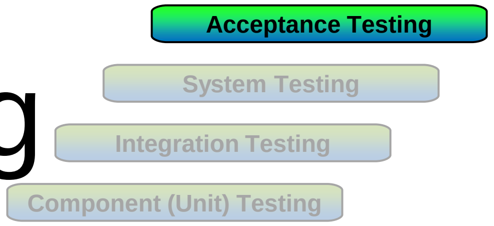
System Testing

Integration Testing

Component (Unit) Testing

Acceptance Testing

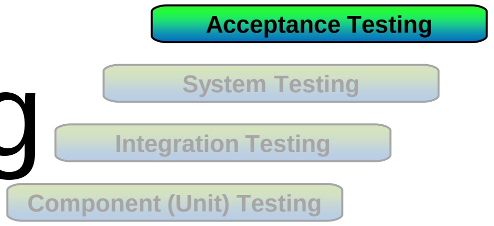
- Alpha and beta testing
  - Objectives
    - ➢ Confidence that the system could be used under normal, everyday conditions
    - ➢ Detection of defects related to the conditions and environment(s) in which the system will be used, especially when those conditions and environment(s) are difficult to replicate by the development team.



Acceptance Testing

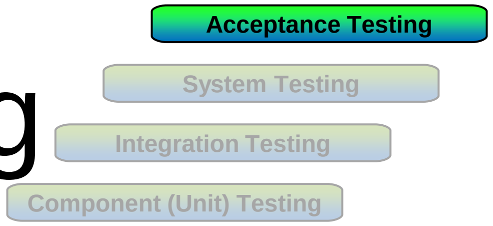
- Test basis
  - Business processes
  - User or business requirements
  - Regulations, legal contracts and standards
  - Use cases and/or user stories
  - System requirements
  - System or user documentation
  - Installation procedures
  - Risk analysis reports

- Test basis for operational acceptance testing
  - Backup and restore procedures
  - Disaster recovery procedures
  - Non-functional requirements
  - Operations documentation
  - Deployment and installation instructions
  - Performance targets
  - Database packages
  - Security standards or regulations



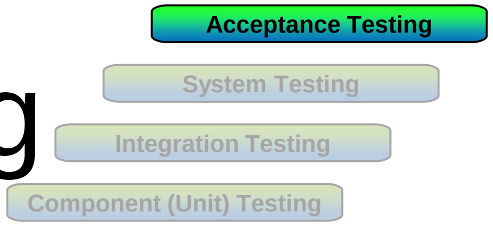
Acceptance Testing

- Test objects
  - System under test
  - System configuration and configuration data
  - Business processes for a fully integrated system
  - Recovery systems and hot sites (for business continuity and disaster recovery testing)
  - Operational and maintenance processes
  - Forms
  - Reports
  - Existing and converted production data



Acceptance Testing

- Typical defects and failures
  - System workflows
    do not meet business or user requirements
  - Business rules are not implemented correctly
  - System does not satisfy
    contractual or regulatory requirements
  - Non-functional failures such as
    - ➢ security vulnerabilities,
    - ➢ inadequate performance efficiency under high loads,
    - ➢ improper operation on a supported platform.



Acceptance Testing

- Specific approaches and responsibilities
  - Acceptance testing is often the responsibility of
    - ➢ customers,
    - ➢ business users,
    - ➢ product owners,
    - ➢ operators of a system,
    - ➢ other stakeholders.
  - often last test level; possible exceptions:
    - ➢ When a COTS software product is installed or integrated
    - ➢ A new functional enhancement before system testing
    - ➢ At the end of an iteration in agile projects

Summary

Test Levels

- Component testing
- Integration testing
  - Component integration testing
  - System integration testing
  - Integration strategies:
    - ➢ Top-down
    - ➢ Bottom-up
    - ➢ Hardest first
    - ➢ “Function oriented”/ ”Transaction oriented”
    - ➢ Ad hoc
    - ➢ Big bang
- System testing
- Acceptance testing
  - User acceptance testing
  - Operational acceptance testing
  - Contractual acceptance testing
  - Regulatory acceptance testing
  - Alpha and beta testing.

Contents

Test types

- ***Test type***:
  A group of test activities based on specific test objectives aimed at specific characteristics of a component or system.
  - Evaluating **functional** quality characteristics, such as completeness, correctness, and appropriateness
  - Evaluating **non-functional** quality characteristics, such as reliability, performance efficiency, security, compatibility, and usability
  - Evaluating whether the **structure** or architecture of the component or system is correct, complete, and as specified
  - Evaluating the **effects of changes**, such as confirming that defects have been fixed (confirmation testing) and looking for unintended changes in behavior resulting from software or environment changes (regression testing)
- To measure testing specific characteristics of a software system, coverage is used.

Test types

- 8 product quality characteristics as defined by ISO 25010 to evaluate a system and software
  - ➢ functional suitability,
  - ➢ performance efficiency
  - ➢ compatibility,
  - ➢ usability,
  - ➢ reliability,
  - ➢ security,
  - ➢ maintainability,
  - ➢ portability.

Functional characteristic

Non-functional characteristics

Source: https://en.wikipedia.org/wiki/ISO/IEC_9126

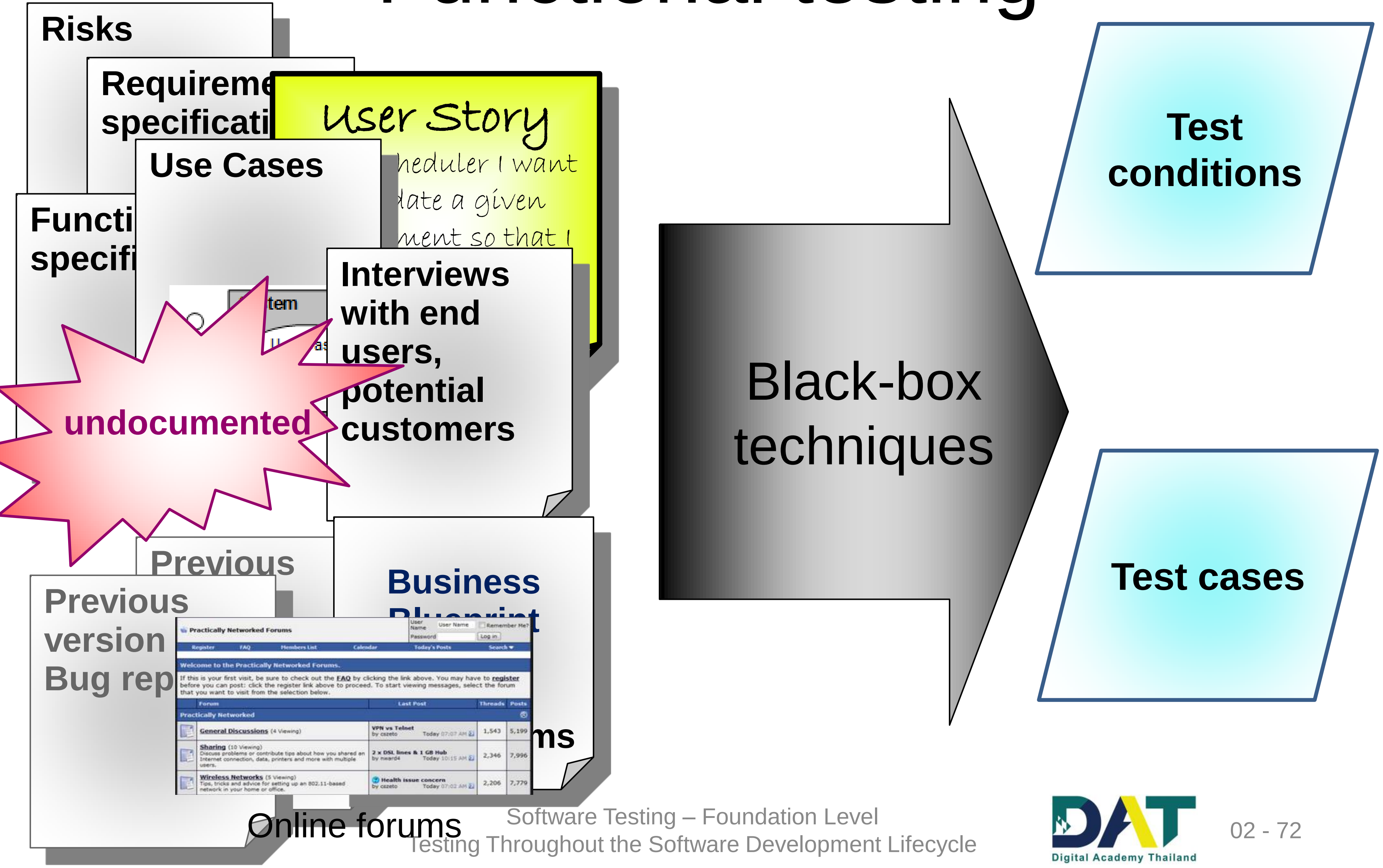
Functional testing

- ***Functional testing***:
  Testing performed to evaluate if a component or system satisfies functional requirements.
  - is the testing of
    “what” the system should do.

Functional testing

- Basics: Functional requirements, such as
  - Business requirements specifications,
  - functional specifications,
  - use cases,
  - ***epics***: A large user story that cannot be delivered as defined within a single iteration or is large enough that it can be split into smaller user stories.
  - ***User stories***: A user or business requirement consisting of one sentence expressed in the everyday or business language which is capturing the functionality a user needs, the reason behind it, any non-functional criteria, and also including acceptance criteria.
- Functional requirements could be undocumented (implicit requirements)

Functional testing



Practically Networked Forums

User Name	User Name	Remember Me?	
Password		Log in	

Register FAQ Members List Calendar Today's Posts Search

Welcome to the Practically Networked Forums.

If this is your first visit, be sure to check out the FAQ by clicking the link above. You may have to register before you can post: click the register link above to proceed. To start viewing messages, select the forum that you want to visit from the selection below.

Forum	Last Post	Threads	Posts
Practically Networked			
General Discussions (4 Viewing)	VPN vs Telnet by cezeto Today 07:07 AM	1,543	5,199
Sharing (10 Viewing) Discuss problems or contribute tips about how you shared an Internet connection, data, printers and more with multiple users.	2 x DSL lines & 1 GB Hub by nward4 Today 10:15 AM	2,346	7,996
Wireless Networks (5 Viewing) Tips, tricks and advice for setting up an 802.11-based network in your home or office.	Health issue concern by cezeto Today 07:02 AM	2,206	7,779

Functional testing

- To be performed at all test levels – with different focus, e.g., tests for components may be based on a component specification

2.2

- Black-box techniques are used to derive
  - test conditions and
  - test cases

4.2

- Functional coverage
  - to measure which functionality has been exercised by tests
  - expressed as a percentage of the type(s) of element being covered
- Expert knowledge
  - Special skills or knowledge of the business area
  - Understanding of different roles

Non-functional testing

- ***Non-functional testing***:
  Testing performed to evaluate that a component or system complies with non-functional requirements.
  – is the testing of
    “how well” the system behaves.

reliability

performance efficiency

compatibility

security

portability

usability

maintainability

Non-functional testing

- To be performed at all test levels as early as possible
- Black-box techniques are used to derive
  - test conditions and
  - test cases.

  Example: boundary value analysis to define the stress conditions for performance tests.

4.2

- Non-Functional coverage
  - to measure the thoroughness of non-functional testing
  - is the extent to which some type of non-functional element has been exercised by tests
  - expressed as a percentage of the type(s) of element being covered

Non-functional testing

- Expert knowledge required, e.g., knowledge of
  - inherent weaknesses of a design or technology
    Example:
    - ➢ Security vulnerabilities associated with particular programming languages
  - particular user base
    Example:
    - ➢ User profiles of
      healthcare facility management systems

White-box Testing

- ***White-box testing*** (***Synonyms:*** *clear-box testing, code-based testing, glass-box testing, logic-coverage testing, logic-driven testing, structural testing, structure-based testing):* Testing based on an analysis of the internal structure of the component or system.
- System's internal structure or implementation may include
  - code,
  - architecture,
  - work flows,
  - data flows within the system.

4.3

White-box Testing

- ***Structural coverage***:
  Coverage measures based on the internal structure of a component or system.
  – extent to which some type of structural element has been exercised by tests,
  – expressed as a percentage of the type of element being covered.
  – At the component testing level we talk about ***Code coverage***: The coverage of code.
  … with different aspects like statement coverage or decision coverage

White-box Testing

- Required skills/knowledge:
  - coding related to used programming language(s),
  - how data is stored (e.g., to evaluate possible database queries),
  - how to use coverage tools and to correctly interpret their results.

Change-related Testing

Change-related testing: A type of testing initiated by modification to a component or system.

- Changes are made to a system
  - to correct a defect ,
  - because of new or changing functionality.
- Change-related testing should confirm that the changes have
  - corrected the defect or
  - implemented the functionality correctly, and
  - not caused any unforeseen negative consequences.

Change-related Testing

- ***Confirmation testing*** *(**Synonym**: re-testing):* A type of change-related testing performed after fixing a defect to confirm that a failure caused by that defect does not reoccur.
  - Depending on the context conduction of
    - ➢ steps to reproduce the failure(s) caused by the defect,
    - ➢ all test cases that failed due to the defect,
    - ➢ new tests to cover changes needed to fix the defect.

Change-related Testing

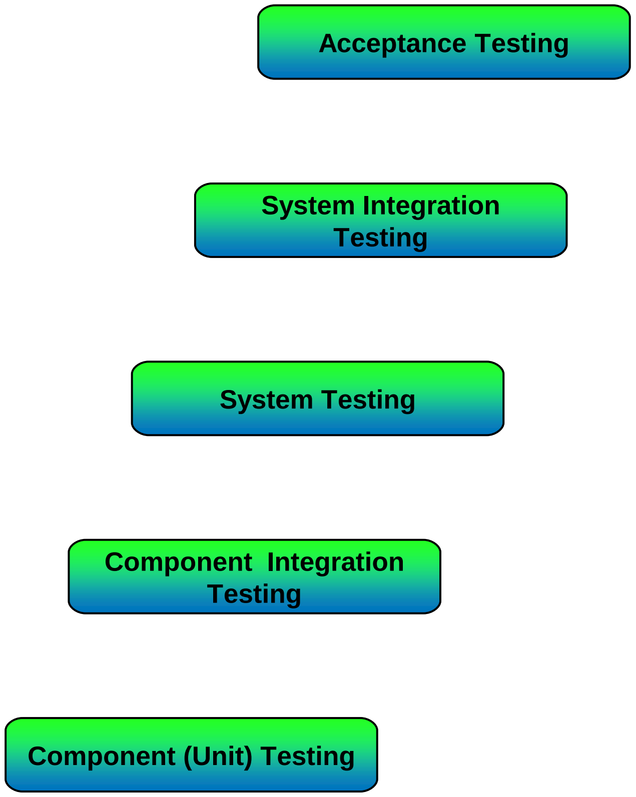
- ***Regression testing***:
  A type of change-related testing to detect whether defects have been introduced or uncovered in unchanged areas of the software.
- Possible changes
  - Fix of a defect
  - Introducing new/changed functionality/features; especially in agile software development and IoT
  - Changes to the environment, such as a new version of an operating system or database management system
  - New version of a COTS software product

Change-related Testing

- Especially for evolving systems very important.
  A change made in one part of the code, could accidentally affect the behavior
  - of other parts of the code,
  - within the same component,
  - in other components of the same system, or even
  - in other systems.
- Regression test suites
  - run many times and generally evolve slowly,
  - are a strong candidate for automation.
    => should start as soon as possible in the project

Test types and test levels

Example: banking application – functional tests



- Banker approves a credit application
- Banker declines a credit application

- Check an account holder’s credit score with an external microservice

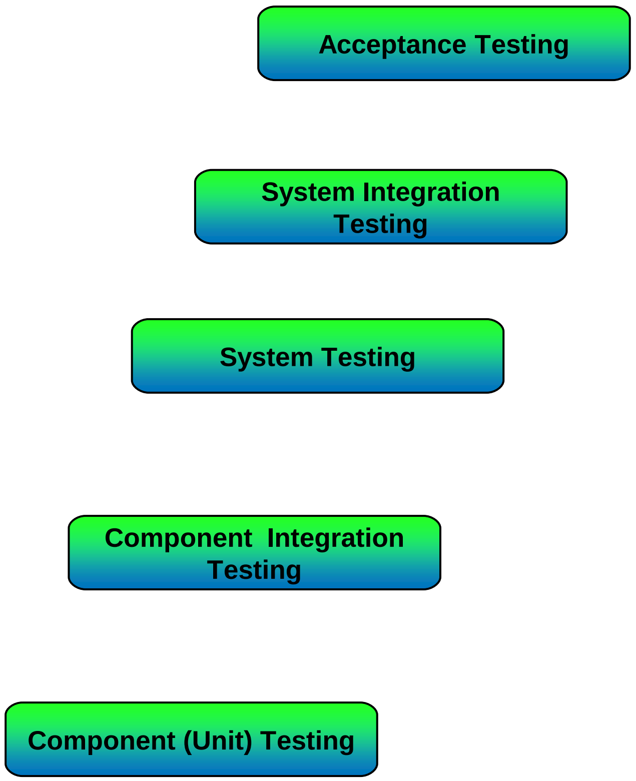
- Account holder applies for a line of credit on his checking account

- Account information captured at the user interface is passed to the business logic.

- Calculate compound interest.

Test types and test levels

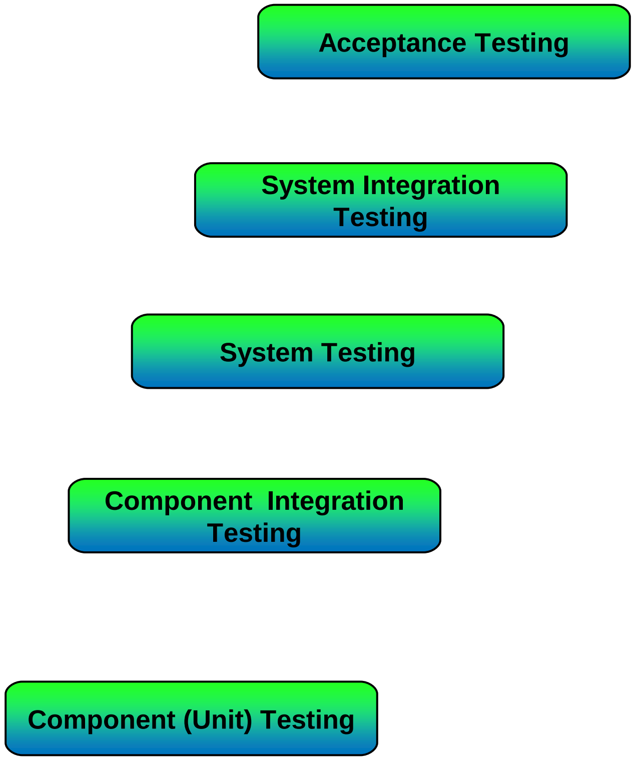
Example: banking application – non-functional tests



- Usability tests: Evaluate the accessibility of the banker’s credit processing interface for people with disabilities.
- Reliability tests: Evaluate system robustness if the credit score microservice fails to respond.
- Portability tests: Check if the presentation layer works on all supported browsers and mobile devices.
- Security tests: Check buffer overflow vulnerabilities due to data passed from the user interface to the business logic.
- Performance tests: Evaluate the number of CPU cycles required to perform a complex total interest calculation..

Test types and test levels

Example: banking application – white-box tests

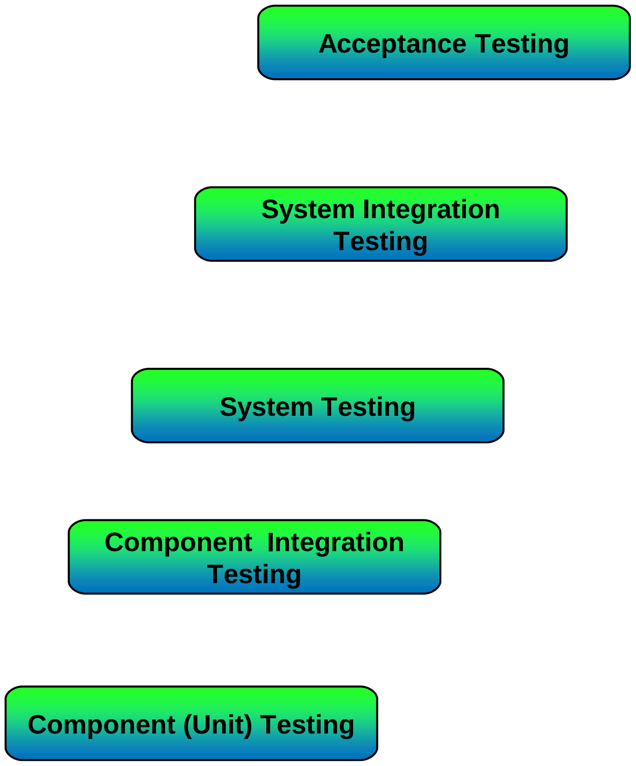


- Cover all supported financial data file structures and value ranges for bank-to-bank transfers.
- Exercise all possible inquiry types sent to the credit score microservice.
- Cover sequences of web pages that can occur during a credit line application.
- Exercise how each screen in the browser interface passes data to the next screen and to the business logic.
- Achieve complete statement and decision coverage for all components that perform financial calculations.

4.3

Test types and test levels

Example: banking application – change-related tests



- All previously-failed tests are re-executed after a defect found in acceptance testing is fixed.
- Tests of the application interacting with the credit scoring microservice are re-executed daily as part of continuous deployment of that microservice.
- All tests for a given workflow are re-executed if any screen on that workflow changes.
- Confirm fixes to interface-related defects as the fixes are checked into the code repository.
- Automated regression tests are built for each component and included within the continuous integration framework.

Summary

- Test types
  - functional
  - non-functional
  - white-box
  - change-related, differing between
    - ➢ confirmation testing
    - ➢ regression testing
- All test types could be performed at any test level

Contents

Maintenance Testing

- Once deployed to production environments, software and systems need to be maintained, e.g.,
  - Changes to fix defects
  - New functionality
  - Alter already-delivered functionality
  - Preserve or improve non-functional quality characteristics,
    - ➢ especially

performance efficiency

compatibility

security

reliability

portability

    - ➢ less important

maintainability

usability

Maintenance Testing

- ***Maintenance***:
  The process of modifying a component or system after delivery to correct defects, improve quality characteristics, or adapt to a changed environment.
- ***Maintenance testing***:
  Testing the changes to an operational system or the impact of a changed environment to an operational system.

Maintenance Testing

- A maintenance release may require maintenance testing
  - at multiple test levels,
  - using various test types.
- The scope of maintenance testing depends on:
  - The degree of risk of the change, for example, the degree to which the changed area of software communicates with other components or systems
  - The size of the existing system
  - The size of the change

Triggers for Maintenance

- Reasons for software maintenance, and thus maintenance testing
  - Modifications
    - ➢ Planned enhancements (e.g., release-based),
    - ➢ Corrective and emergency changes,
    - ➢ Changes of the operational environment (operational system and/or database upgrades),
    - ➢ Upgrades of COTS software,
    - ➢ Patches for defects and vulnerabilities.

Triggers for Maintenance

- Reasons for software maintenance, and thus maintenance testing
  - Migration
    - ➢ Migration from one platform to another ⇨ operational tests
      - ❖ of the new environment
      - ❖ of the changed software
    - ➢ Data migration ⇨ tests of data conversion
      Data from another application are migrated into the system being maintained
    - ➢ Retirement of a system could require
      - ❖ testing of data migration,
      - ❖ testing archiving if long data retention periods are required,
      - ❖ testing of restore/retrieve procedures,
      - ❖ regression testing of remaining functionality.

Impact Analysis for Maintenance

Impact analysis:
The identification of all work products affected by a change, including an estimate of the resources needed to accomplish the change.

- evaluates the changes that were made for a maintenance release to identify the intended consequences
- evaluates expected and possible side effects of a change
- identifies the areas in the system that will be affected by the change
- helps to identify the impact of a change on existing tests.

Impact Analysis for Maintenance

- Impact analysis challenges
  - Specifications (e.g., business requirements, user stories, architecture) are out of date or missing
  - Test cases are not documented or are out of date
  - Bi-directional traceability between tests and the test basis has not been maintained
  - Tool support is weak or non-existent
  - The people involved do not have domain and/or system knowledge
  - No sufficient attention has been paid to the software's maintainability during development

Summary

- Maintenance testing is required for productive systems because of changes related to
  - Modifications
    - ➢ Planned enhancements with new/changed functionality
    - ➢ Corrective and emergency changes like hot fixes of defects
    - ➢ Patches for defects and vulnerabilities.
  - Migration
    - ➢ Migration from one platform to another
    - ➢ Data migration
    - ➢ Retirement of a system