

Software Engineering

COMP1035

Lecture 12

*Release Testing &
Acceptance Testing*

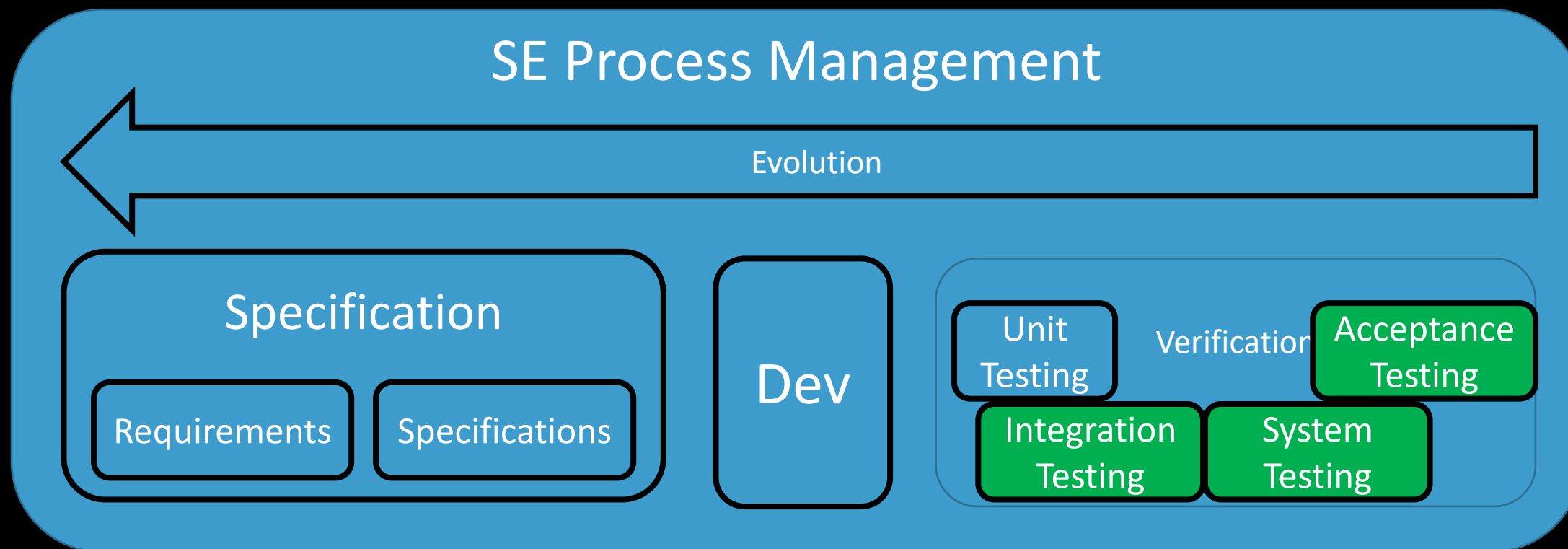


- Integration Testing
- System Testing
- Acceptance Testing

Where We Are In the Process

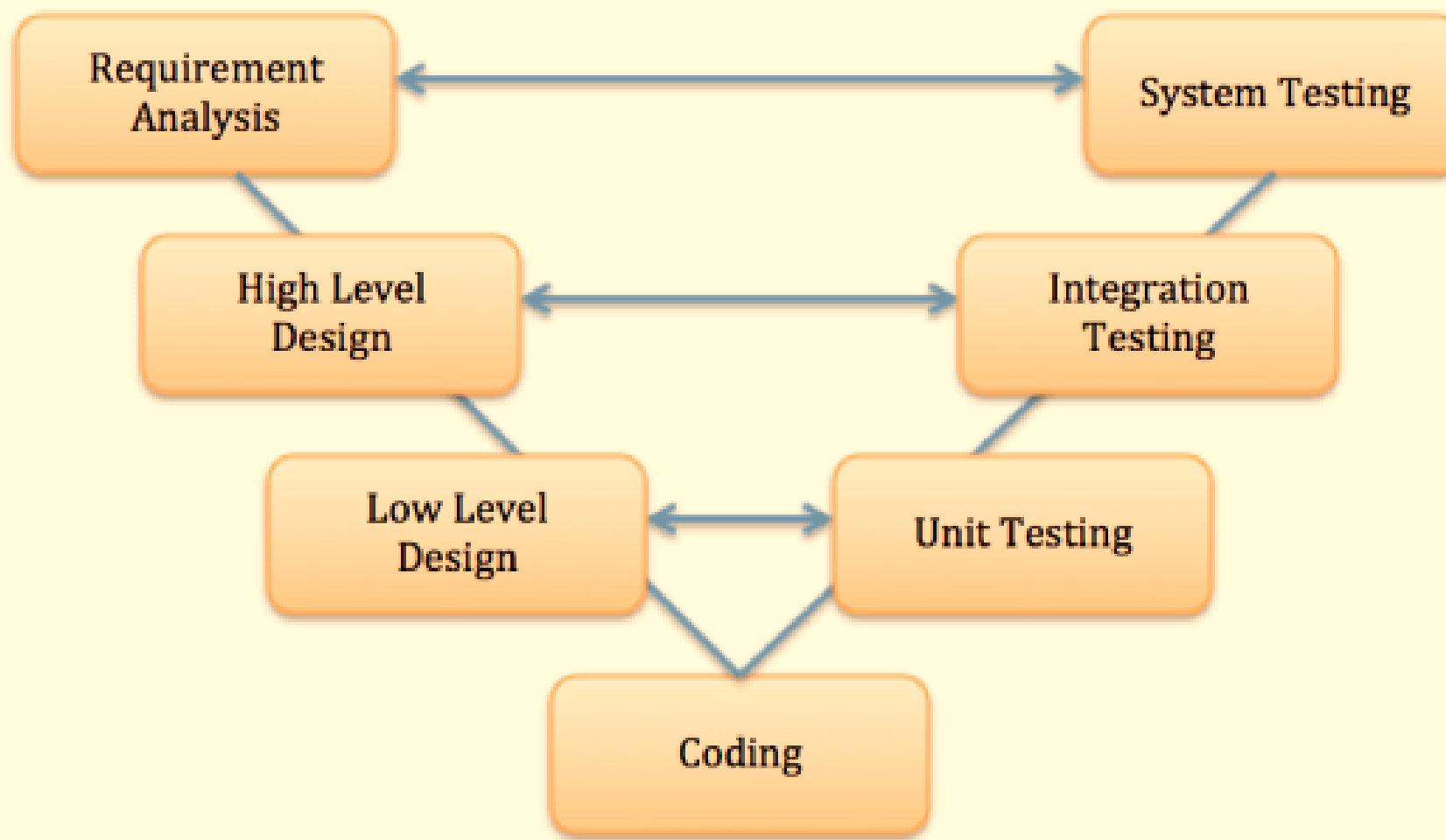


Keeping Track of SE Module





Keeping Track of SE Module



Integration Testing

- Combining: combine all of the units within a program and test them as a group.
- Designed to find interface **defects** between the **modules/functions**.
 - Whether the individual units can be integrated into a sub-system correctly.
 - Check data flow from one module to other modules.



Types of Integration Testing

- **Big bang**

- All the modules are first required to be completed and then integrated.
- After integration, testing is carried out on the integrated unit as a whole.

- **Top-down**

- Testing flow starts from top-level modules that are higher in the hierarchy towards the lower-level modules
- If the lower-level modules have not been developed, using stubs/dummy modules



Types of Integration Testing

- **Bottom-up**

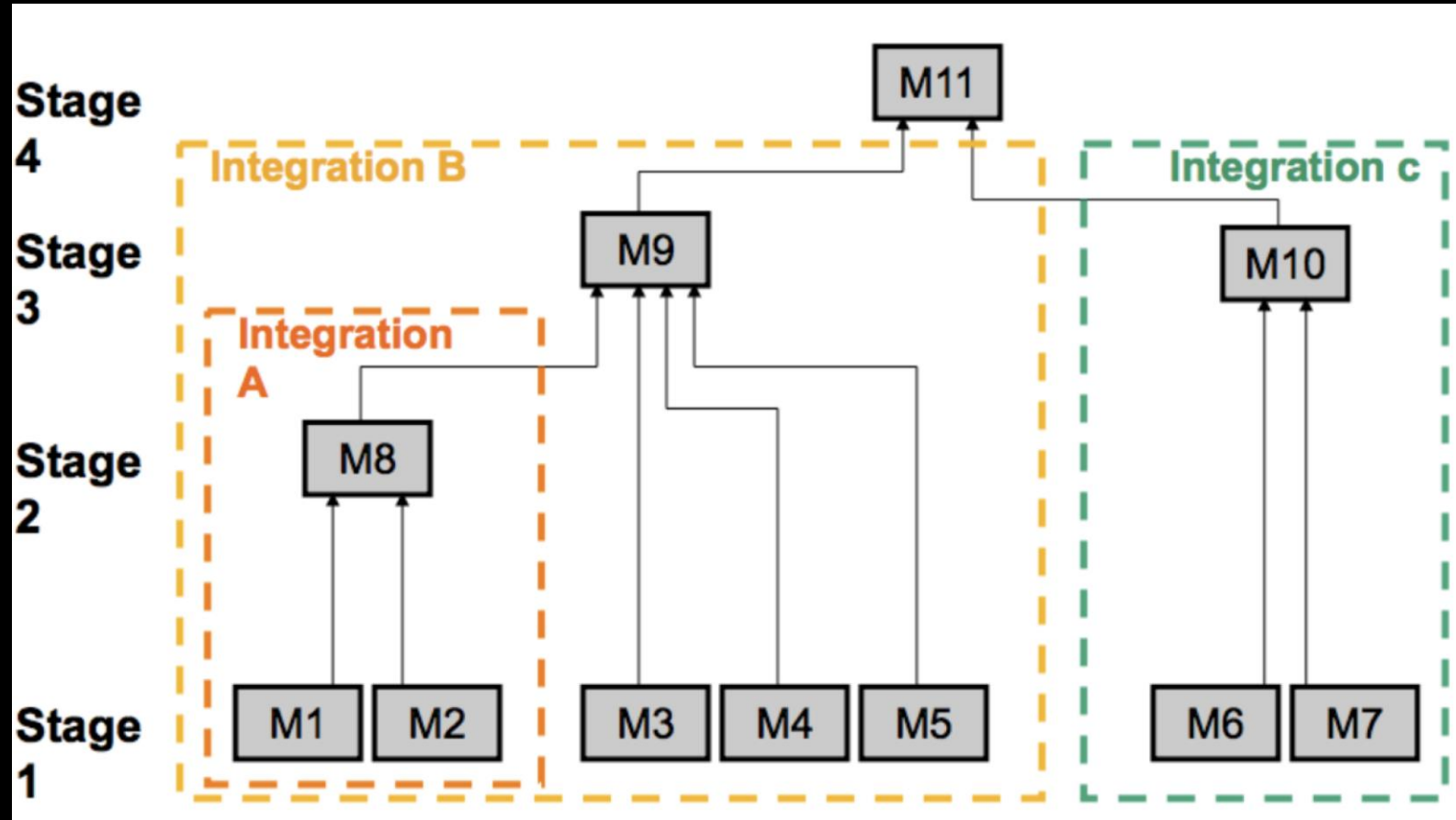
- Starts from lower-level modules, moving upwards to the higher-level modules.
- If the high-level modules have not been developed, using drivers.

- **Hybrid integration**

- Combination of both top-down and bottom-up
- Starts from the middle layer, testing is carried out in both directions
- Making use of both stubs and drivers, whenever necessary



A Sample of Integration Testing





Unit Testing V.S Integration Testing

	Unit Testing	Integration Testing
What to test	Individual pieces (e.g. classes)	Combinations of pieces (subsystems)
Input	Functional specification of unit (unit test case)	Functional specification of subsystem
Output	Pass/Fail	Bug reports
By who	Developer	Development Team
Frequency	Many per day	Periodically e.g., end of a sprint
Difficulty	Easy	Difficult
Speed	Very fast (few seconds or less)	Still consider fast (few minutes or more)



Testing In Google – The Testing Pyramid

“Google often suggests a 70/20/10 split: 70% unit tests, 20% integration tests, and 10% end-to-end tests.”

The key points are:

Unit tests > Integration tests > E2E tests

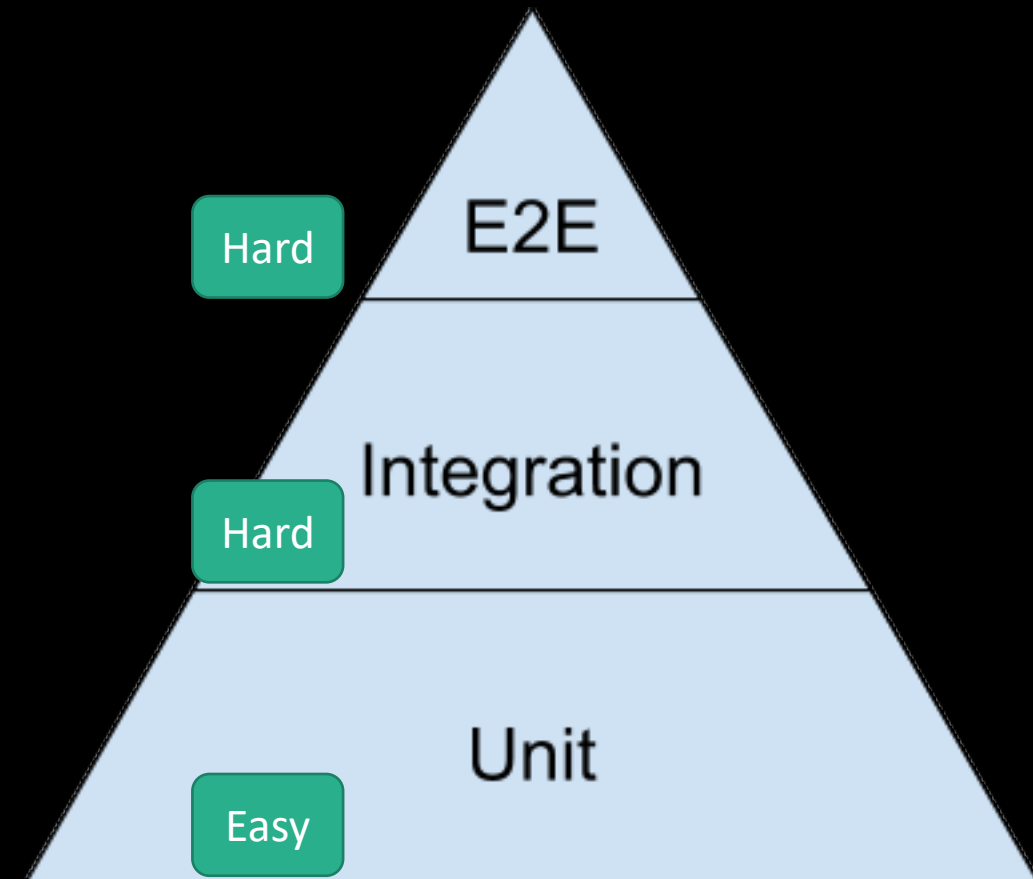


Figure source: https://docs.google.com/presentation/d/15gNk21rjer3xo-b1ZqyQVGebOp_aPvHU3YH7YnOMxtE/edit#slide=id.g437663ce1_53_98

System Testing

Release Testing



System Testing

- After Unit Testing and Integration Testing
 - Once the complete system is ready and can be tested as a whole
- This is a test of the full system
 - Is it 'done'? Is it ready?
- Check the whole system
 - To meet the full specifications (business requirements)
 - Including non-functional ones!
- “This process is concerned with finding errors that result from unanticipated interactions between components”



System Testing

- Primary goal: convince the company that the software is good enough to give to the customer
 - The end is a sign-off approval that it is ready
- Team Role: not let the software go to acceptance test until it is ready
- Way to do this: show that it meets all of the specifications
 - Functional and non-functional
 - Usually the final test to verify that the system meets the specification



Types of System Testing

- **Usability Testing:** measure how easy and user-friendly a software application is.
- **Load Testing:** non-functional software testing process in which the performance of software application is tested under a specific expected load (real-life loads).
- **Regression Testing:** ensure none of the changes made over the course of the development process can affect existing features.



Types of System Testing

- **Recovery Testing:** demonstrate a software solution is reliable (can recovery from possible crashes)
- **Migration Testing:** ensure the software can be moved from older system infrastructures to current system infrastructures without any issues.
- **Functional Completeness Testing:** think of any possible missing functions.
- **Hardware/Software Testing:** focus on the interactions between the hardware and software



System Testing

- “Therefore, System Testing should focus on testing the interactions between different components [...] that make up the system [...] Because of its focus on interactions, **use case-based testing** is an effective approach to system testing” – Sommerville
- It is nearly impossible to test all interactions comprehensively - especially because it is hard to automate system testing
- Need a strategy for this



- Metrics:

- **Response Time**: the amount of time it takes the application to respond to requests.
- **Resource Utilization**: database, memory, I/O, and CPU, etc.
- **Workload**: the number of users and concurrent tasks an application can manage.
- **Throughput**: amount of transactions the software can implement in a pre-specified period of time.



- **Methods:**

- **Load Testing:** test the application against increasing loads until it reaches its maximum potential.
- **Stress Testing:** check the stability of the application regarding to hardware resources (disk space, CPU, memory, etc.).
- **Scalability Testing:** determine the capability of the software to scale its operations regarding to data volumes, number of transactions, and user load levels, etc.
- **Endurance Testing:** check the application with expected amounts of user load – but over a long period of time.
- **Volume Testing:** check the application holds up against a large amount of data.



- A testing approach in which **test cases**, **conditions** and **data** are **derived from requirements**.
 - Defining Test Completion Criteria
 - Design Test Cases
 - Execute Tests
 - Verify Test Results
 - Verify Test Coverage (both functional and non-functional aspects of the requirement)
 - Track and Manage Defects

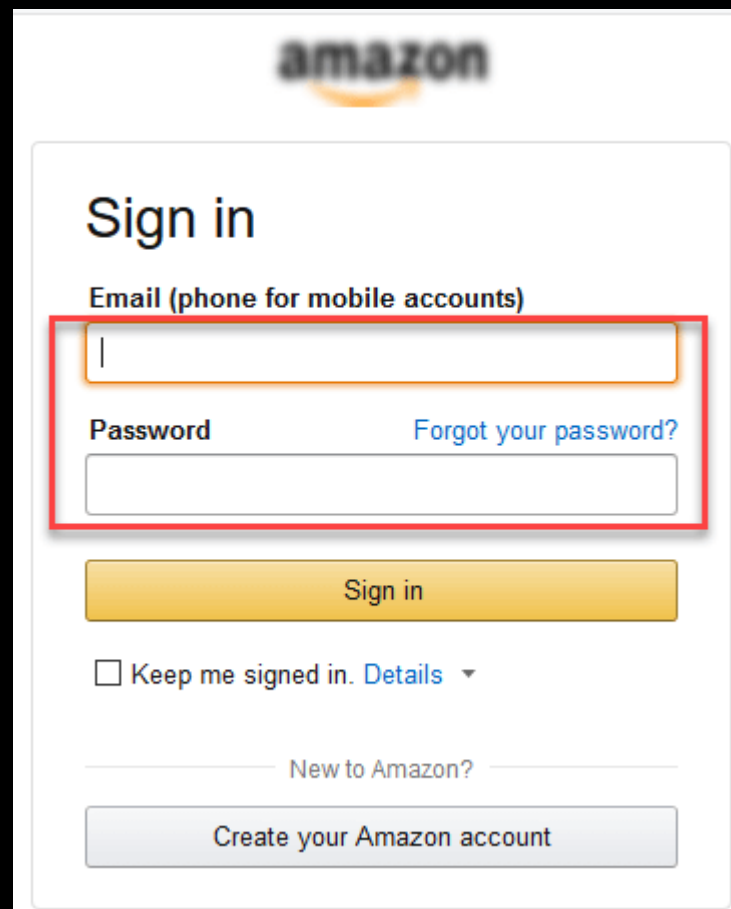


Scenario Driven Strategy

- The actual scenarios are used for testing the software application instead of test cases.
 - To test end to end scenarios for a specific complex problem of the software.
 - To verify that the user can perform the desired actions completely **from beginning to end**.
- As a release tester
 - You play the role of the scenario character
 - Make both **intended mistakes** and the **right actions**
- A scenario test activity may test **several individual requirements**



1. Check system behavior when valid email id and password is entered.
2. Check system behavior when invalid email id and valid password is entered.
3. Check system behavior when valid email id and invalid password is entered.
4. Check system behavior when invalid email id and invalid password is entered.
5. Check system behavior when email id and password are left blank and Sign in entered.
6. Check Forgot your password is working as expected
7. Check system behavior when valid/invalid phone number and password is entered.
8. Check system behavior when “Keep me signed” is checked



The image shows the Amazon sign-in page. At the top is the Amazon logo. Below it is the heading "Sign in". Under the heading is the label "Email (phone for mobile accounts)" followed by a text input field. Below this is the label "Password" followed by a text input field. To the right of the password field is a link "Forgot your password?". Below these fields is a yellow "Sign in" button. Under the button is a checkbox labeled "Keep me signed in." followed by a link "Details" and a dropdown arrow. At the bottom is a link "New to Amazon?" followed by a button "Create your Amazon account". A red rectangular box highlights the email and password input fields.



Compared with Integration Testing

- Three major differences
 - A separate team that has not been involved in the system development should be responsible for system testing. (**Developer Team** vs. **QA Team**)
 - Rather than finding integration bugs, the objective of system testing is to check that the system **meets its specifications**, and is good enough for external use.
 - This is validation testing - rather than defect testing (black box or white box?)



Comparisons of Different Testing

	Unit Testing	Integration Testing	System Testing
What to test	Individual pieces (e.g. classes)	Combinations of pieces (subsystems)	Full system
Input	Functional specification of unit (unit test case)	Functional specification of subsystem	Full functional and non-functional specifications
Output	Pass/Fail	Bug reports	(1) Validation reports; (2) Sign off for acceptance tests
By who	Developer	Development Team	QA Team (Testing Team)
Frequency	Many per day	Periodically e.g., end of a sprint	Prior to showing client
Difficulty	Easy	Difficult	Difficult
Speed	Very fast (few seconds or less)	Relatively slow (few minutes or more)	Slow

Acceptance Testing

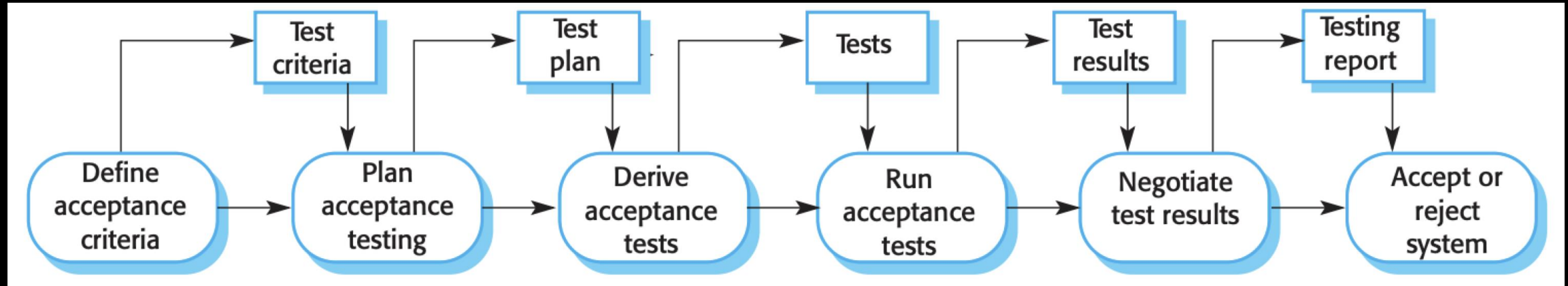


Acceptance Testing

- “Formal testing with respect to user needs, requirements, and business processes conducted to determine whether a system satisfies the acceptance criteria and to enable the **user, customers** or other **authorized entity** to determine whether to **accept the system**.”
 - It takes place after system testing
 - Final testing stage before the software is accepted for active use
 - Conducted by the **customers**, using real data
- Acceptance implies that payment should be made for the product



Acceptance Testing Process



© Pearson Education Limited 2016



Best Practices for Acceptance Testing

1. Prepare AT plan early in the project life cycle
2. Prepare Checklist before the UAT starts
3. Conduct Pre-AT session during System Testing phase itself
4. Set the expectation and define the scope of UAT clearly
5. Test End to End business flow and avoid system tests
6. Test the system or application with real-world scenarios and data
7. Think as an Unknown user to the system
8. Perform Usability Testing
9. Conduct Feedback session and meeting before moving to production



The Accept/Reject Stage

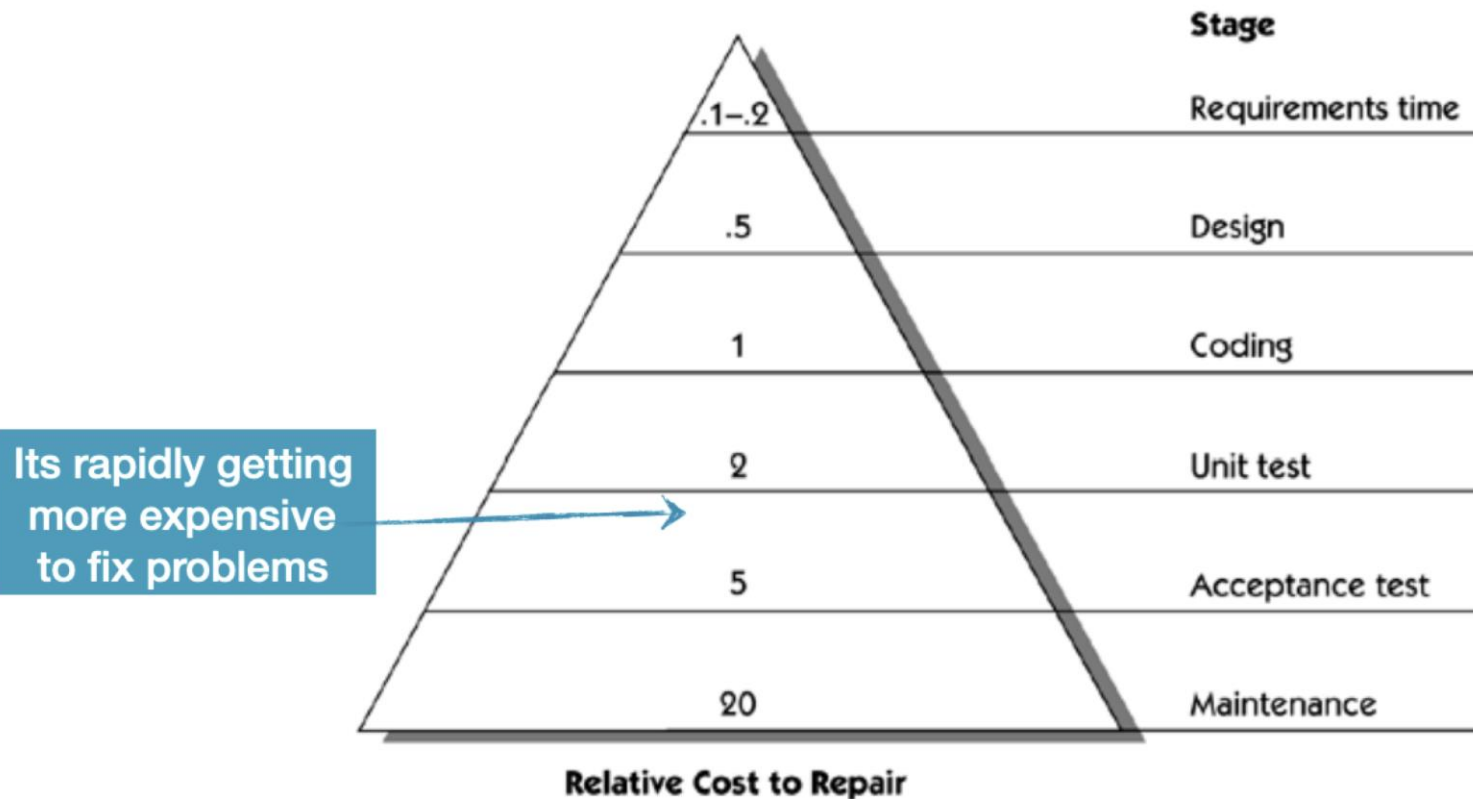
- Although it sounds clear cut - its not always so
- Customers might need the new software ASAP
 - It might be on their critical path schedule
- “They might be willing to accept the software, irrespective of problems, because the costs of not using the software are greater than the costs of working around the problems.”
- The agreement might be - provide an updated version later

Comparisons of Different Testing

	Unit Testing	(Subsystems) Integration Testing	(Full System) Release Testing	(Customer) Acceptance Testing
What to test	Individual pieces (e.g. classes)	Combinations of pieces (subsystems)	Full system	Full system
Input	Functional specification of unit (unit test case)	Functional specification of subsystem	Full functional and non-functional specifications	User requirements
Output	Pass/Fail	Bug reports,	1) Validation reports 2) Sign off for acceptance tests	1) Acceptance reports 2) Sign off for end of contract
By who	Developer	Development team	QA team (Testing team)	QA team (Testing team) and customer
Frequency	Many per day	Preiodically e.g., end of a sprint	Prior to showing client	Prior to use
Difficulty	Easy	Difficult	Difficult	Usually Difficult
Speed	Very fast (few seconds or less)	Relatively slow (few minutes or more)	Slow	Usually Slow

Costs to Repair a Defect in Different Lifecycles

Figure 1-2. Relative cost to repair a defect at different lifecycle phases. (Data derived from [Davis \[1993\]](#).)





- Integration Testing
 - Test subsystems, conduct by development team
 - Find bugs/defects between modules/functions
- System Testing Processes
 - Test full system, conduct by QA team
 - Check if the system meet the full specifications
- Acceptance Testing
 - Test full system, conduct by customer (user)
 - Check if the software can be deployed in practice

THANK
YOU