Introduction to TMMi and Testing Basics

Organized by

Supported by

Funded by SME Development Fund

工業貿易署
Trade and Industry Department

Topics

9:30AM-12:30AM

Test Process Improvement

Overview of The TMMi

Break

Process areas of the TMMi model

2:00PM – 5:00PM

Testing Documentation

Testing Strategy & Testing Plan

Break

Testing Case Optimization

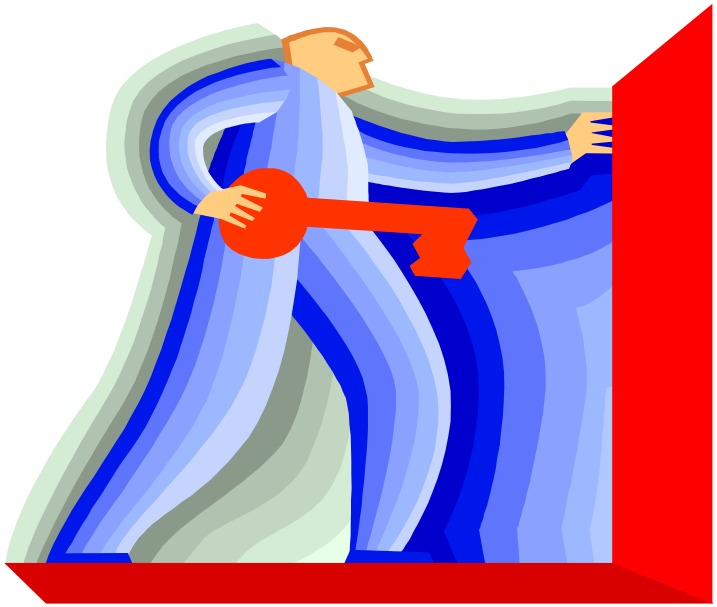
Measurement and Defect

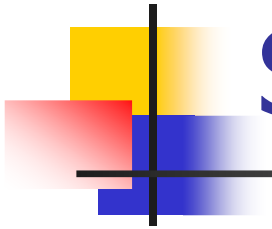
Government Initiative

1 Test Process Improvement

Software Testing

- Process of searching for software errors
- How and when do we start?



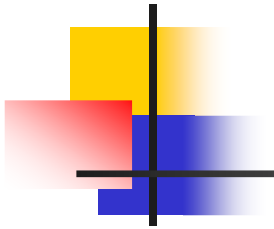


Software Testing

- Testing is often considered as an expensive and uncontrollable process.
- Testing takes too much time, costs a lot more than planned, and offers insufficient conception of the quality of the test process.
- Therefore, the quality of the information system and the risks for the business can be difficult to determine

The Defective Code (Fault)

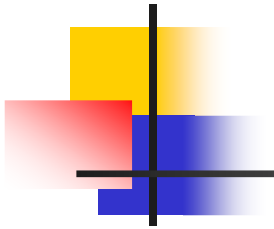
```
1   void zunebug(int days) {
2     int year = 1980;
3     while (days > 365) {
4       if (isLeapYear(year)){
5         if (days > 366) {
6           days -= 366;
7           year += 1;
8         }
9         else {
10        }
11      }
12      else {
13        days -= 365;
14        year += 1;
15      }
16    }
17    printf("current year is %d\n", year);
18 }
```



The Correction

```
1   void zunebug(int days) {
2     int year = 1980;
3     while (days > 365) {
4       if (isLeapYear(year)){
5         if (days > 366) {
6           days -= 366;    // delete this
7           year += 1;
8         }
9         else {
10      }
11        days -=366;        // insert this
11        }  else {
13          days -= 365;
14          year += 1;
15      }
16    }
17    printf("current year is %d\n", year);
18 }
```

A small fault!



Errors, Faults, Failures

Error: a mistake in the design or programming; made by a human

Fault: a mistake in the code; the result of an error

Failure: the occurrence of a software fault

Users don't observe errors or faults. They observe execution failures.

H. Mills

How to prevent failure?

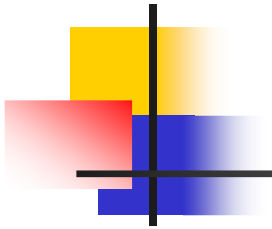
- Don’t make mistake
- Ask someone do checking for us
- Use more tools (automation)

Bomb Making and Process Improvement

Two worst issues we do not want to see:

It should blast but it doesn't.

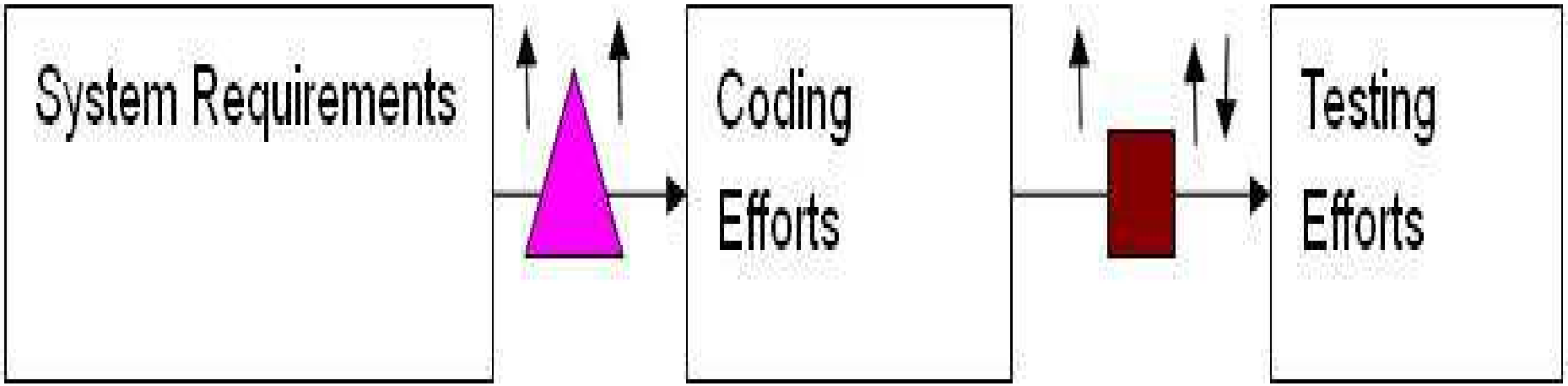
It blasts but it shouldn't.



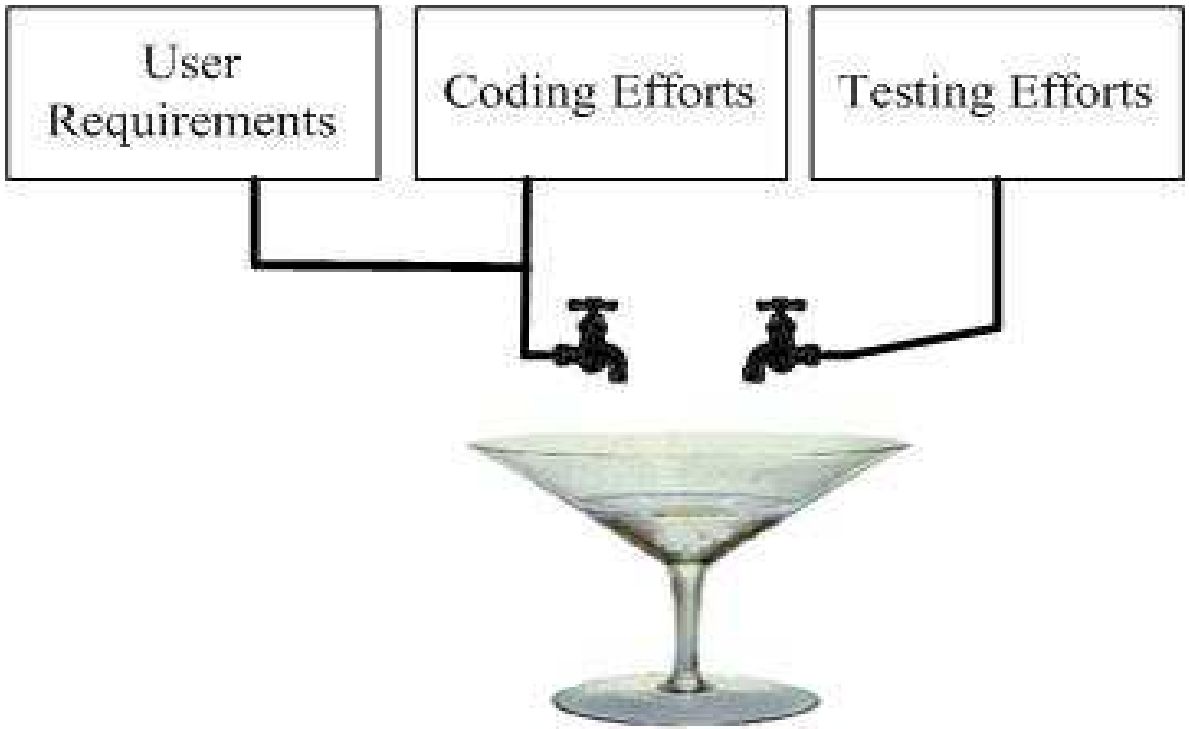
Quality

- Quality means "conformance to requirements"
  - The best testers can only catch defects that are contrary to specification.
  - Testing does not make the software perfect.
  - If an organization does not have good requirements engineering practices then it will be very hard to deliver software that fills the users' needs, because the product team does not really know what those needs are.

Testing Efforts and Coding Efforts

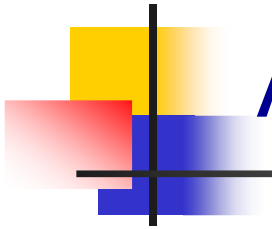


Testing decoupled



Why Invest in Process Improvement?

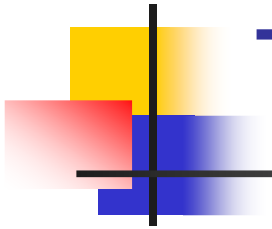
- Reduce overhead
- Increase efficiency & effectiveness
- Allow test to embrace change (key to agile testing) Focus is on delivering results
- Improve Test's influence in order to deliver better quality



Avalable Model ?

- There are many models that a test process improvement effort can follow:
- TPI®
- TMMi
- etc

2 Overview of The TMMi

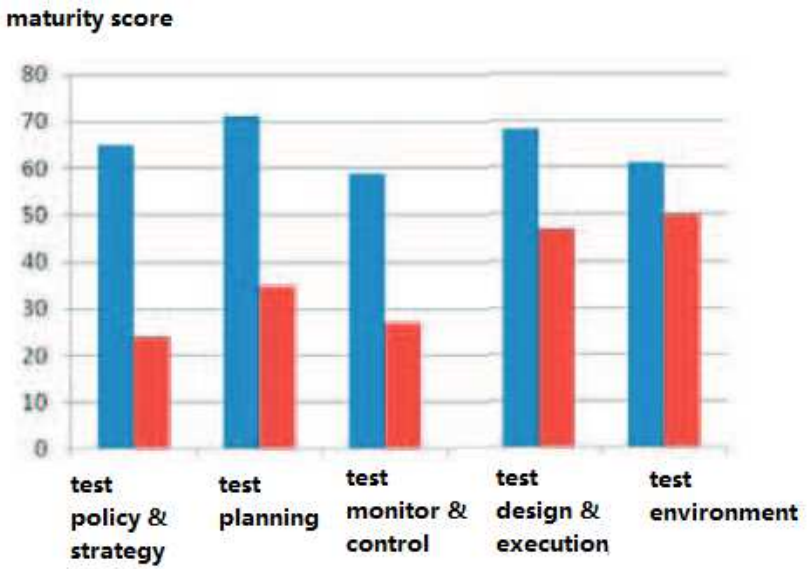


TMMi

- Introduced in 2007
- TMMi is based on TMM (partially published in 1996)
- New CMMI replacing the old SW-CMM
- Need for independent test maturity assessment
- Need for standard certification
- Focus on testing and integration with CMMI
- Practical experiences have shown that TMMi can also be applied success fully in organizations which are not at all familiar with CMMI.

TMMi Company Adoption

- Analyzing the assessment data, a significantly higher maturity score was observed on especially the managerial TMMi
- A significantly higher maturity score was observed on especially the managerial TMMi process areas for organizations that are also using CMMI (in blue) compared with those that are not also using CMMI (in red)



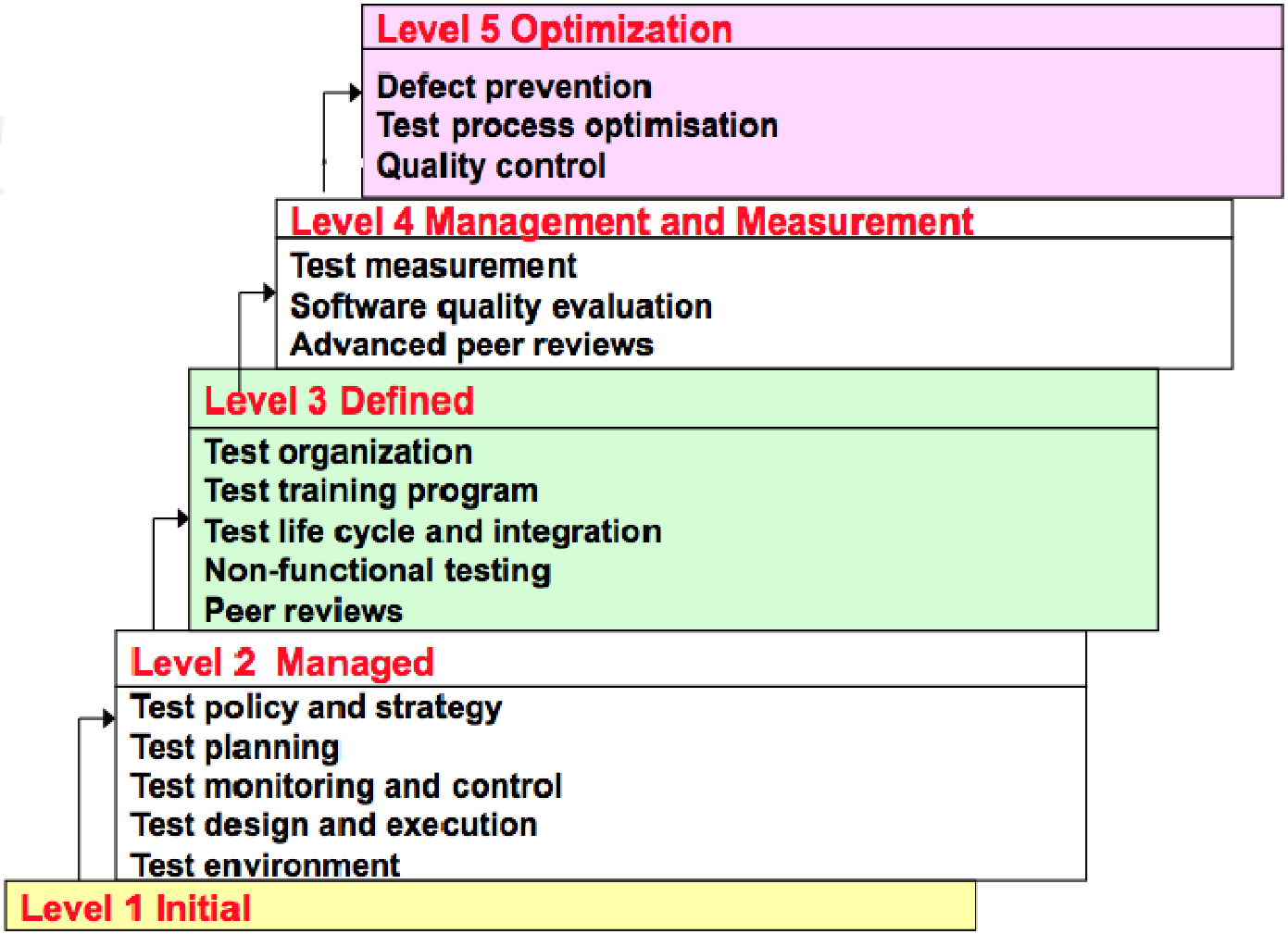
TMMi Foundation

- A non-profit making organization
- non-commercial test improvement model
- Dedicated to improving test processes and practice.
- Focus: development of a common, robust model of test process assessment and improvement in IT organizations
- As of May 2012, 800 members representing 330 unique companies and 55 unique countries

TMMi Foundation

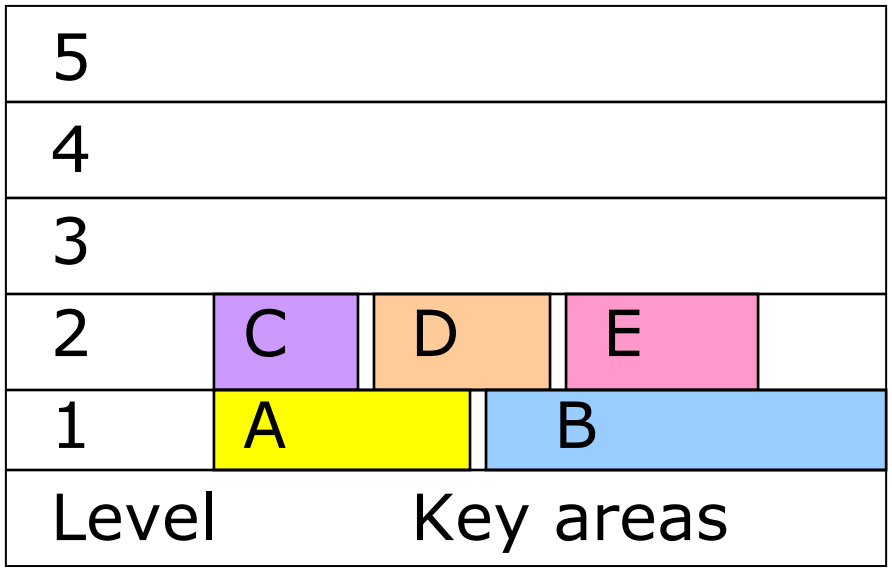
- A standard staged TMMI model that can be used in isolation or in support of other Software Process Improvement models
- An independently managed data repository to support TMMI assessment method accreditation, assessor and assessment certification/validation and validated assessment data and certificates
- Certification and training/examination process, procedures and standards for formal, public accreditation of assessors and lead assessors and the on-going management.

TMMi Levels

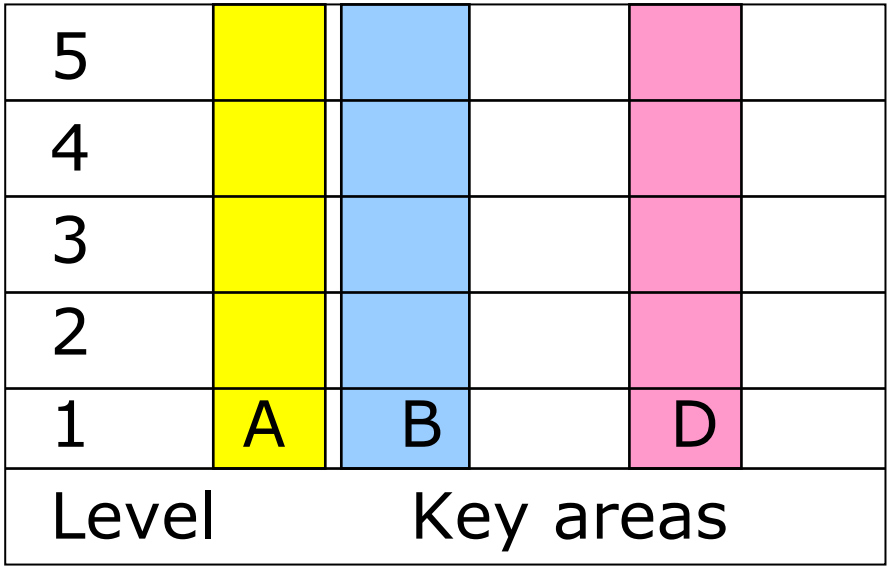

Level 5 Optimization
Defect prevention
Test process optimisation
Quality control
Level 4 Management and Measurement
Test measurement
Software quality evaluation
Advanced peer reviews
Level 3 Defined
Test organization
Test training program
Test life cycle and integration
Non-functional testing
Peer reviews
Level 2 Managed
Test policy and strategy
Test planning
Test monitoring and control
Test design and execution
Test environment
Level 1 Initial

Models

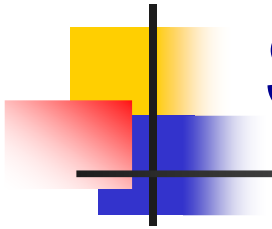
Stage model



Continuous model



Note: It is ground floor in GB and first floor in the US.

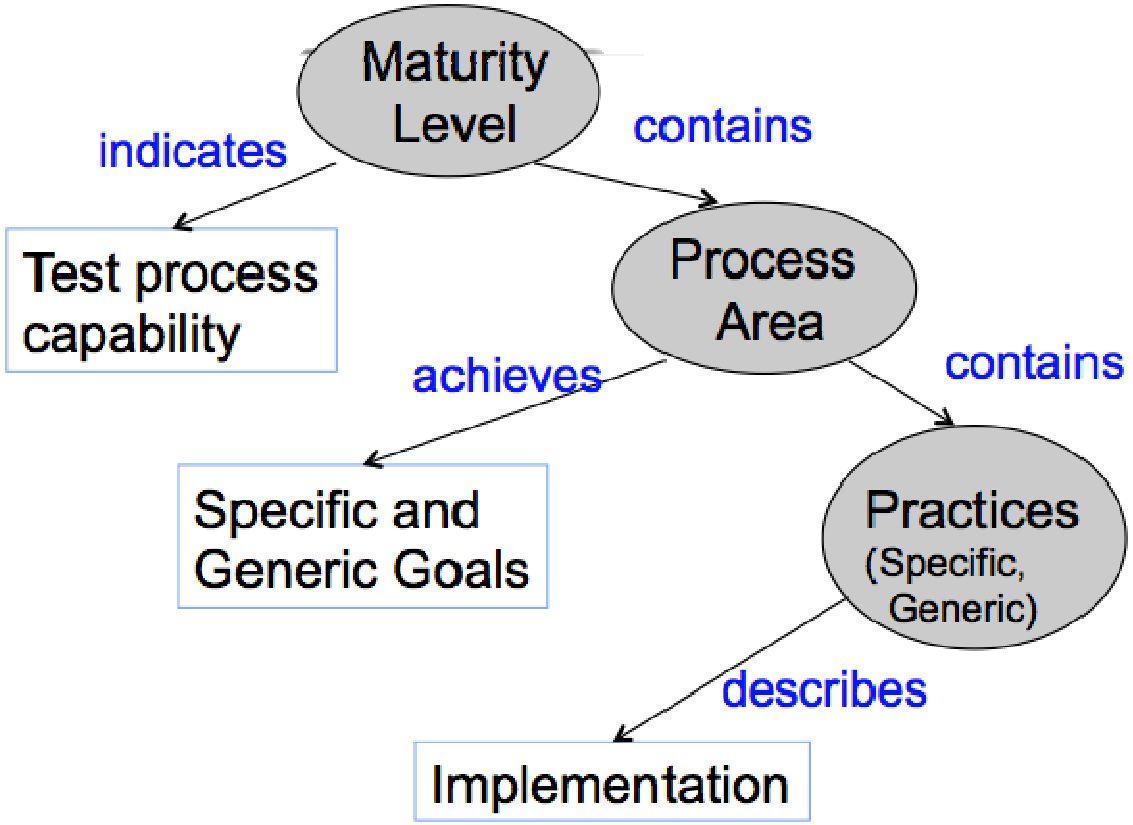


Some Statistics

- According to a study done by Experimentus of UK
  - Survey 100 companies
  - 72.5% were at TMMi level 1 (they are working in a chaotic, hero-based way but starting to build project based processes)
  - 27.5% were at TMMi level 2 (they have some established project based process and are moving towards implementing process at an organizational level)
  - More than 70% of respondents do not have metrics in place to monitor or manage testing goals.

3 Process areas of the TMMi model

Components within each Maturity Level



Level 1 Organization

- Testing is a chaotic and is often considered a part of debugging.
- The organization usually does not provide a stable environment to support testing.
- Success in these organizations depends on the competence and heroics of the people in the organization.
- Tests are developed in an ad-hoc way after coding is completed.
- The objective of testing is to show that the software runs without major failures.

Level 1 Organization

- The delivered product often does not fulfill its needs, is not stable, or is too slow to work with.
- Within testing there is a lack of resources, tools and well-educated staff.
- Organizations are characterized by a tendency to over commit, abandonment of processes in a time of crises, and an inability to repeat their successes.
- Products tend not to be released on time, budgets are overrun and quality is not according to expectations.
- No defined process areas.

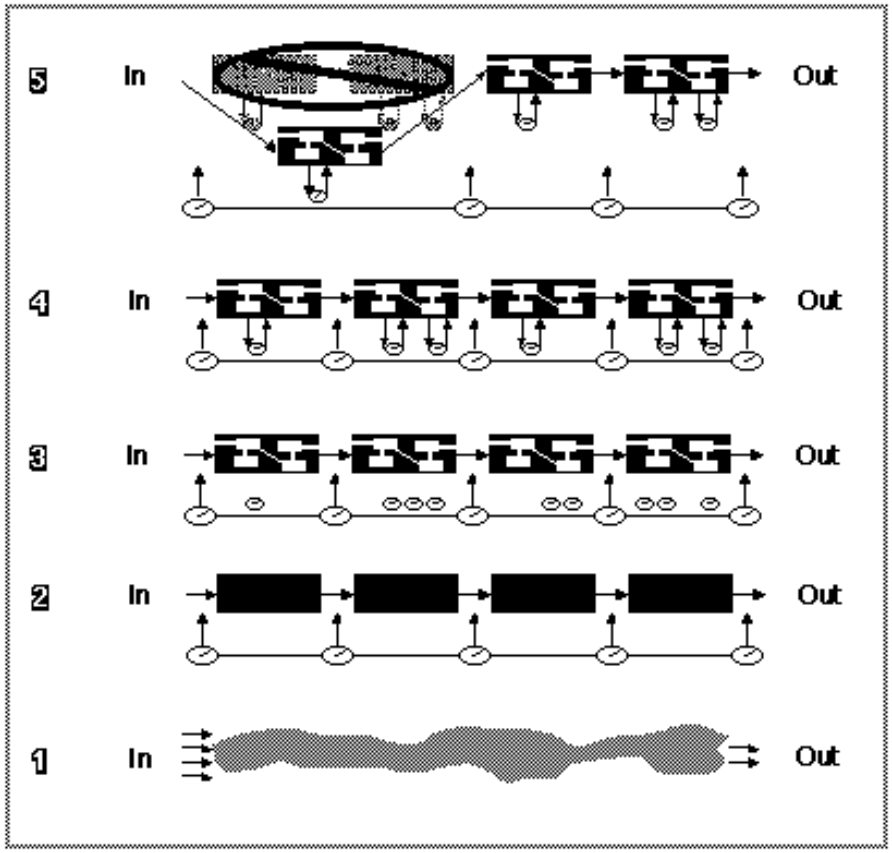
Level 2 Organization

- Testing becomes a managed process and is clearly separated from debugging.
- Testing is still perceived as being a project phase that follows coding.
- A company-wide or programme-wide test strategy is established.
- Test plans are also being developed. The test plan defines test approach and what testing is required, when, how and by whom.
- Risk management techniques are used to identify the product risks based on documented requirements.
- Testing is monitored and controlled to ensure it is going according to plan.

Level 2 Organization

- The status of the work products and the delivery of testing services are visible to management.
- Testing may still start relatively late in the development life cycle, e.g. during design or during the coding phase.
- Testing is multileveled: there are unit, integration, system and acceptance test levels.
- The main objective of testing is to verify that the product satisfies the specified requirements.
- Many quality problems occur because testing occurs late in the development life cycle. Defects are propagated from the requirements and design into code.
- No formal review programs.

Visibility



Generic Goal of Level 2

GG 2 Institutionalize a Managed Process

- GP 2.1 Establish an organizational policy
- GP 2.2 Plan the process
- GP 2.3 Provide resources
- GP 2.4 Assign responsibilities
- GP 2.5 Train people
- GP 2.6 Manage configurations
- GP 2.7 Identify and involve relevant stakeholders
- GP 2.8 Monitor and control the process
- GP 2.9 Objectively evaluate adherence
- GP 2.10 Review status with higher level management

Level 3 Organization

- Testing is fully integrated into the development life cycle.
- Test planning is done at an early project stage, e.g. during the requirements phase.
- The organization's set of standard processes is established and improved over time.
- A test organization and a specific **test training program** exist, and testing is perceived as a profession.
- Test cases are gathered, stored and managed in a central database for re-use and regression testing.
- Basic tools support key testing activities.
- A formal review program **is implemented.** Reviews take place across the life cycle.

Level 3 Organization

- Now do **non-functional testing**, e.g. on usability and/or reliability.
- Testing processes are tailored from the organization's set of standard processes to suit a particular project or organization unit and therefore are more consistent except for the differences allowed by the tailoring guidelines.

Generic Goal of Level 3

GG 2 Institutionalize a Defined Process

- GP 3.1 Establish a defined process
- GP 3.2 Collect improvement information

Organization, Training, Integration

- Establish a software test organization
  - Well trained and dedicated group in charge of test process
  - Oversees test planning, test execution & recording, defect tracking, test database, test reuse, test tracking & evaluation
- Establish a technical training program
  - Staff trained in test planning, methods, standards and tools
  - Staff prepared for review process and user participation
- Integrate testing into the software life cycle
  - Testing activities are carried out in all phases of life cycle
  - Test planning initiated early, user input solicited

Level 4 Organization

- Testing is a thoroughly defined, well-founded and measurable process.
- The organization and projects establish quantitative objectives for product quality and process performance and use them as criteria in managing them.
- Product quality and process performance is understood in statistical terms and is managed throughout the life cycle.
- Measures are incorporated into the organization's measurement repository to support fact-based decision making.

Level 4 Organization

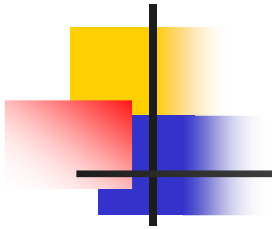
- Reviews and inspections are considered to be part of testing and used to measure document quality.
- The static and dynamic testing approach are integrated into one.
- Reviews are formally used as means to control quality gates.
- Products are evaluated using quantitative criteria for quality attributes such as reliability, usability and maintainability.
- An organization wide test measurement program provides information and visibility regarding the test process.
- Testing is perceived as evaluation; it consists of all life cycle activities concerned with checking products and related work products.

Level 5 Organization

- Testing is now a completely defined process and one is capable of controlling the costs and the testing effectiveness.
- Organization continually improves its processes based on a quantitative understanding of the common cause of variation inherent in processes.
- Improving test process performance is carried out through incremental and innovative process and technological improvements.
- The methods and techniques are optimized and there is a continuous focus on fine-tuning and test process improvement.

Level 5 Organization

- Defect prevention and quality control are practiced.
- Statistical sampling, measurements of confidence levels, trustworthiness, and reliability drive the test process. The test process is characterized by sampling based quality measurements.
- A detailed procedure exists for selecting and evaluating test tools. Tools support the test process as much as possible during test design, test execution, regression testing, test case management, etc.
- Process reuse is also practiced by a process asset library.
- Testing is a process with the objective to prevent defects.



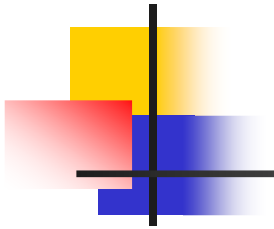
TMMi Summary

Level 1

- Testing is a chaotic process - ill defined and not distinguished from debugging.
- Tests are developed ad hoc after coding is complete.
- Objective of testing is to show that the system and software works.
- Lacks a trained professional testing staff and testing tools.

Level 2

- Testing as a separate function from debugging.
- Testing becomes a defined phase following coding.
- Primary goal is to show that the system and software meets specifications.
- The process is standardized to the point where basic testing techniques and methods are in place.



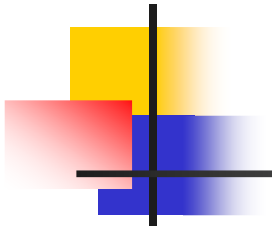
TMMi Summary

Level 3

- Testing is integrated into the entire life cycle.
- Test objectives are now based on the system requirements.
- A formal testing organization exists.
- Formal testing technical training, controls and monitors the testing process, and begins to consider using automated test tools.
- Management recognizes testing as professional activity.

Level 4

- Testing is a measured & quantified process.
- Products are also tested for quality attributes such as reliability, usability, and maintainability.
- Test cases are collected and recorded in a database for reuse and regression testing.
- Defects are logged, given a severity level, and a priority for correction.



TMMi Summary

Level 5

- Testing is well defined and managed
- Testing costs and effectiveness are monitored.
- Automated tools are a primary part of the testing process
- There is an established procedure for selecting and evaluating testing tools.

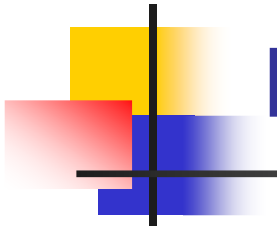
Exercise (10 mins)

- You are outsourcing hotel booking web and mobile app test tasks to a third party solution.
- There are three potential software outsourcing companies
- Can you provide five solid points how you would you like to evaluate the capabilities of three candidates.

4 Testing Document

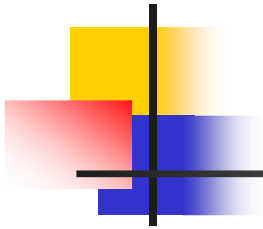
Testing Documentation

- PRD (Product Requirement Document)
- FS (Functional Specification)
- UI Spec (User Interface Specification)
- Test Plan
- Test Case
- Test Suite
- Traceability matrix
- Risk Analysis



PRD (Product Requirement Document)

- What: Set of software requirements
- Who: **Product Marketing, Sales, Technical Support**
- When: **Planning stage**
- Why: **We need to know what the product is supposed to do**
- QA role:
  - Participate in reviews
  - Analyze for completeness
  - Spot ambiguities
  - Highlight contradictions
  - Provide feedback on features/usability

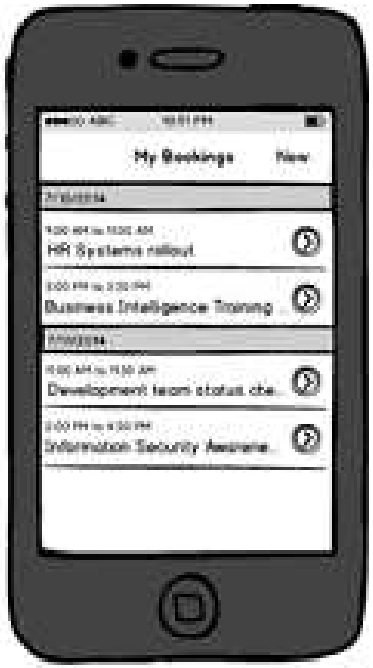


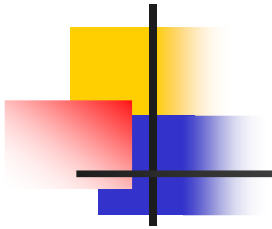
FS (Functional Specification)

- What: software design document;
- Who: Engineering, Architects;
- When: (planning)/design/(coding) stage(s);
- Why**: we need to know how the product will be designed**;
- QA role:
  - Participate in reviews;
  - Analyze for completeness;
  - Spot ambiguities;
  - Highlight contradictions.

Example: Room Booking App

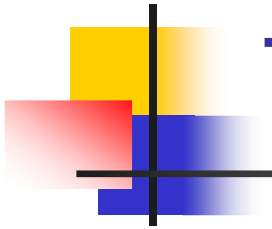
Requirement ID	Requirement	Criticality	Test Priority	Test ID
12.02	**The app** should have **system** generated, time-stamped audit trail to record the date and time of **booking entries and** actions that create, modify, or delete electronic record	H	H	6.1
12.03	**The app** should record **login users** who are responsible for creating, modifying and deleting records	H	H	6.2
12.03	It should be possible to view and print audit trail information	H	M	N/A





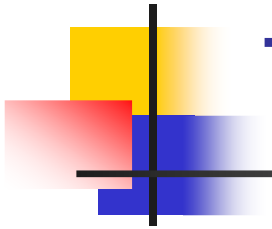
Test Plans

- The goal of test planning is to establish the list of tasks which, if performed, will identify all of the requirements that have not been met in the software. The main work product is the *test plan*.
  - **The test plan documents the overall approach to the test. In many ways, the test plan serves as a summary of the test activities that will be performed.**
  - It shows how the tests will be organized, and outlines all of the testers' needs which must be met in order to properly carry out the test.
  - The test plan should be inspected by members of the engineering team and senior managers.



Test Plan (cont'd)

- What: a document describing the scope, approach, resources and schedule of intended testing activities; identifies test items, the features to be tested, the testing tasks, who will do each task and any risks requiring contingency planning;
- Who: QA;
- When: **(planning)/design/coding/testing stage(s);**



Test Plan (cont'd)

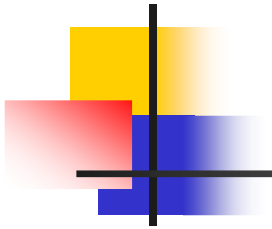
- Why:
  - **Divide responsibilities** between teams involved; if more than one QA team is involved (ie, manual / automation, or English / Localization) – responsibilities between QA teams ;
  - **Plan for test resources / timelines** ;
  - Plan for test coverage;
  - Plan for OS / DB / software deployment and configuration models coverage.
- - QA role:
  - Create and maintain the document;
  - Analyze for completeness;
  - Have it reviewed and signed by Project Team leads/managers.

Example

Task Name	Start	Finish	Effort	Comments
Test Planning				
Review Requirements documents			2 d	
Create initial test estimates			1 d	
Staff and train new test resources				
First deploy to QA test environment				
Functional testing – Iteration 1				
Iteration 2 deploy to QA test environment				
Functional testing – Iteration 2				
System testing				
Regression testing				
UAT				
Resolution of final defects and final build testing				
Deploy to Staging environment				
Performance testing				
Release to Production				

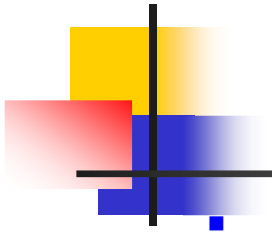
Example

Deliverable	For	Date / Milestone
Test Plan	Project Manager; QA Director; Test Team	
Traceability Matrix	Project Manager; QA Director	
Test Results	Project Manager	
Test Status report	QA Manager, QA Director	
Metrics	All team members	



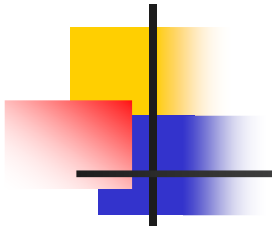
Test Case

- A *test case* is a description of a specific interaction that a tester will have in order to test a single behavior of the software.
- **Test cases are very similar to use cases**, in that they are step-by-step narratives which define a specific interaction between the user and the software.
- Test cases must be repeatable.
  - Good test cases are data-specific, and describe each interaction necessary to repeat the test exactly.



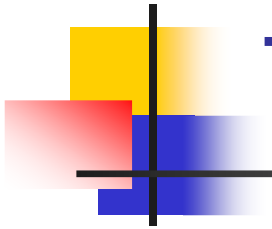
Test Case

- What: a set **of inputs, execution preconditions and expected outcomes** developed for a particular objective, such as exercising a particular program path or verifying compliance with a specific requirement;
- Who: QA;
- When: (planning)/(design)/coding/testing stage(s);
- Why:
  - Plan test effort / resources / timelines;
  - Plan / review test coverage;
  - **Track test execution progress;**
  - Track defects;
  - Track software quality criteria / quality metrics;
  - Unify Pass/Fail criteria across all testers;
  - Planned/systematic testing vs Ad-Hoc.



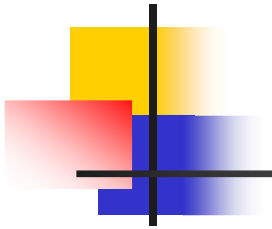
Test Case (cont'd)

- Five required elements of a Test Case:
  - **ID** – unique identifier of a test case;
  - **Features to be tested** / steps / input values – what you need to do;
  - **Expected result** / output values – what you are supposed to get from application;
  - **Actual result** – what you really get from application;
  - **Pass / Fail**.



Test Case (cont'd)

- Optional elements of a Test Case:
  - Title – verbal description indicative of testcase objective;
  - Goal / objective – primary verification point of the test case;
  - Project / application ID / title – for TC classification / better tracking;
  - Functional area – for better TC tracking;
  - **Bug numbers for Failed test cases** – for better error / failure tracking (ISO 9000);
  - Positive / Negative class – for test execution planning;
  - Manual / Automatable / Automated parameter etc – for planning purposes;
  - Test Environment.

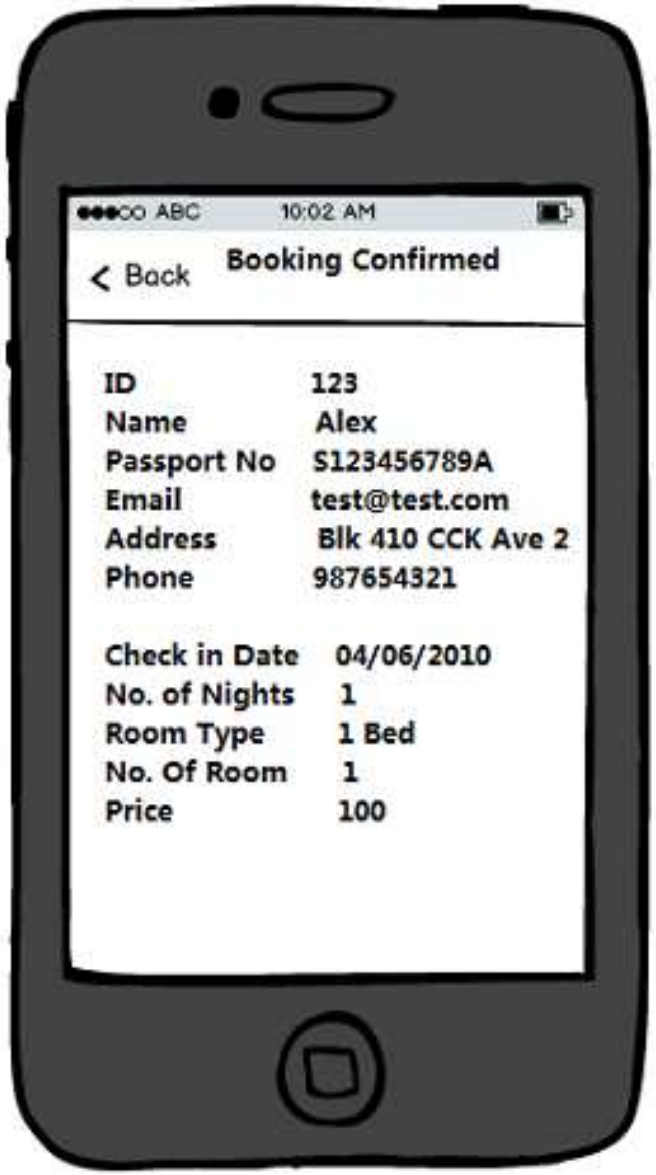


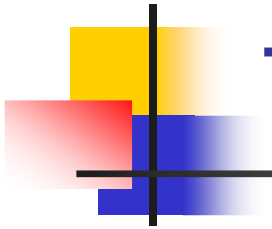
Test Case (cont'd)

- Inputs:
  - Through the UI;
  - From interfacing systems or devices;
  - Files;
  - Databases;
  - State;
  - Environment.
- Outputs:
  - To UI;
  - To interfacing systems or devices;
  - Files;
  - Databases;
  - State;
  - Response time.

Example

1	Book Room		
	Description	Allow users to add a room booking of their choice.	
	Test Inputs	1. Enter into the fields **Add Bookings** section of the **BookRoom**	
		Check-in Date: 04/06/2010 No. Of Nights: 1	Room Type: Room with ID 1 has 1 beds and costs 100 per night. No. Of Rooms: 1
		2. Enter into the fields Fill in **Personal Particulars** section of the **BookRoom**	
		Name: alex Passport No: S123456789A	Email: *test@test.com* *Contact No: 987654321* *Address:Blk 410 CCK Ave 2*
	Test Procedure	1. click on the "**Book Room**" to add a room booking.	
		2. Enter into the specific fields under the section of **Add Bookings** based on the test inputs listed and click "Add a Booking".	
		3. Enter into the specific fields under the section of "**Personal Particulars**" based on the test inputs listed and click "Submit".	
	Expected Results	1. After user clicks on the "Add a Booking" button, Bookings added by user will be displayed in a table form, showing the booking details of each bookings made. Booking details includes the check in date, number of nights, room type, number of rooms and the prices of each booking.	
		2. After filling up the input fields under the **Personal Particulars** Section, user clicks on the "submit" button. User will be directed to **Booking Comfirmed** Booking made by user has been confirmed. Details of user's personal particulars and booking details will be displayed.	
	Actual Results	Same as expected result.	





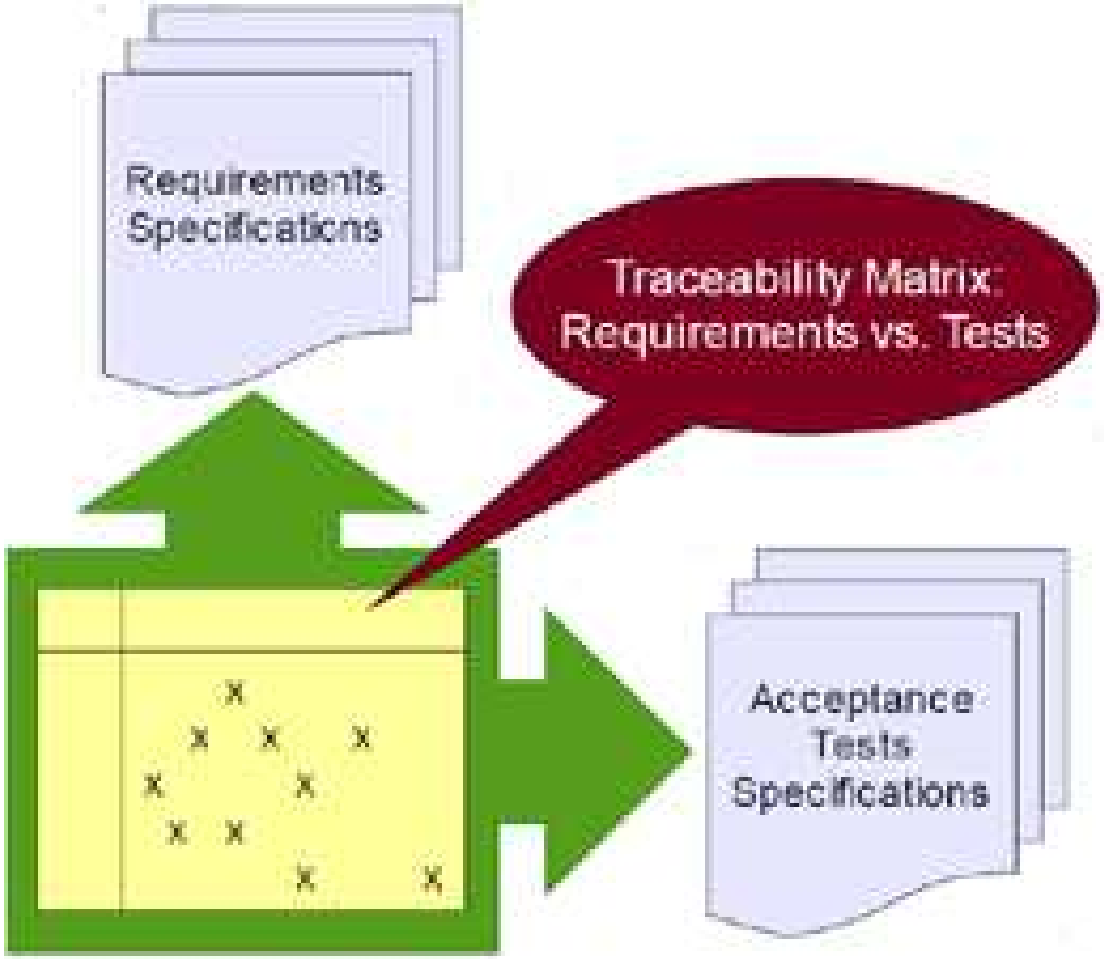
Test Suite

- A document specifying **a sequence of actions for the execution of multiple test cases;**
- Purpose: to put the test cases into **an executable order**, although individual test cases may have an internal set of steps or procedures;
- **Is typically manual**, if automated, typically referred to as test script (though manual procedures can also be a type of script);
- Multiple Test Suites need to be organized into some sequence – this defined the order in which the test cases or scripts are to be run, what timing considerations are, who should run them etc.

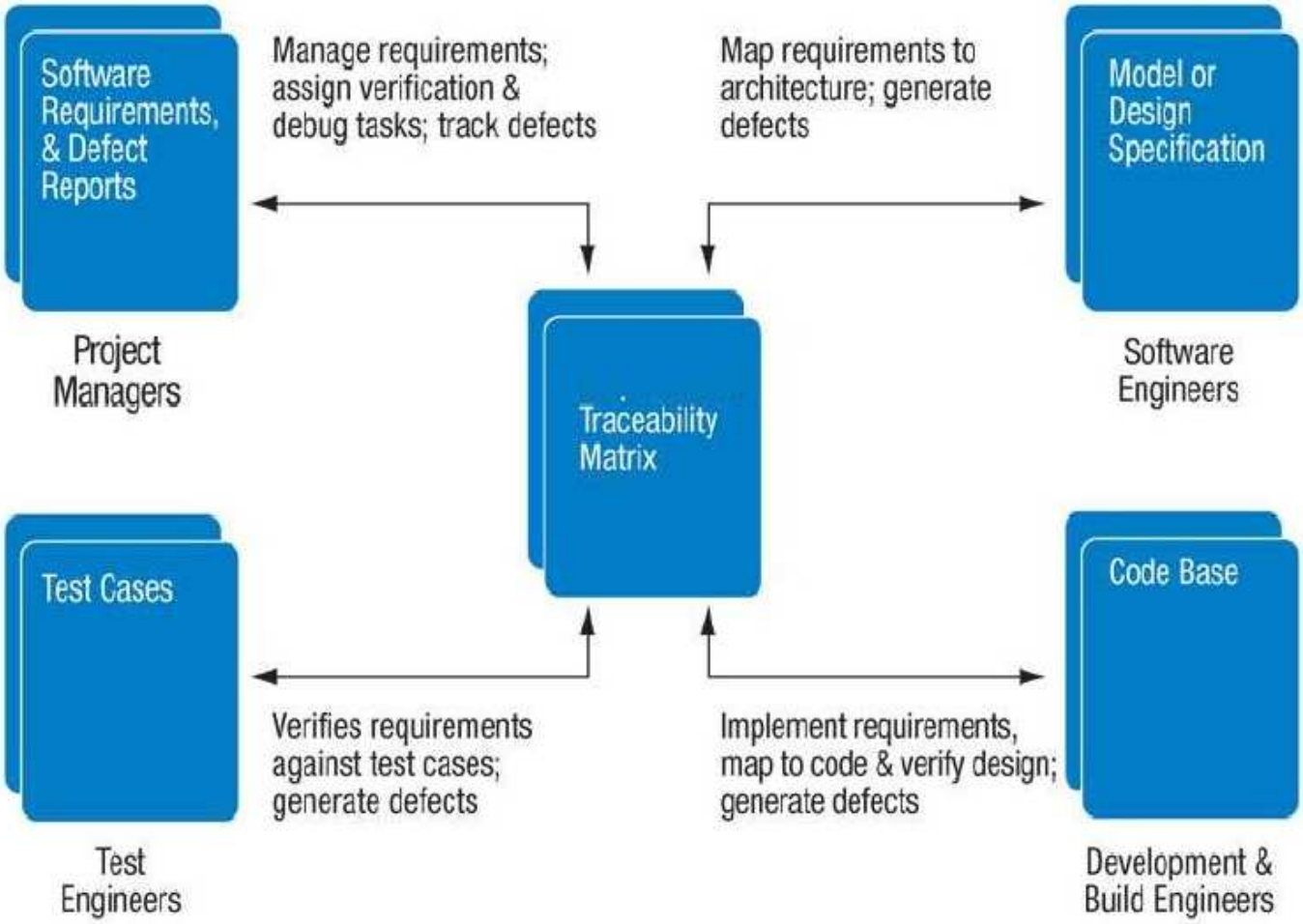
Traceability Matrix (cont'd)

- What: document tracking each software feature from PRD to FS to Test docs (Test cases, Test suites);
- Who: Engineers, QA;
- When: (design)/coding/testing stage(s);
- Why: we need to **make sure each requirement is covered in FS and Test cases**;
- QA role:
  - Analyze for completeness;
  - Make sure each feature is represented;
  - Highlight gaps.

Traceability Matrix (cont'd)

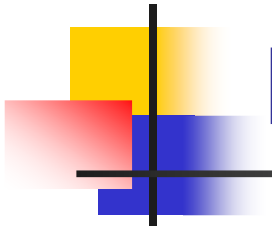


Traceability Matrix



Example

Reqs IDs	Reqs Tested	UC 1.1	UC 1.2	UC 1.3	UC 2.1	UC 2.2	UC 2.3.1	UC 2.3.2	UC 2.3.3	UC 2.4	UC 3.1	UC 3.2	TECH 1.1	TECH 1.2	TECH 1.3
Test Cases	231	3	2	3	1	1	2	1	1	1	2	3	1	1	1
1.1.1	1	x													
1.1.2	2		x	x											
1.1.3	2	x											x		
1.1.4	1			x											
1.1.5	2	x												x	
1.2.1	1		x												
1.2.2	1			x											
1.2.3	2				x		x								
1.2.4	2					x		x							
1.2.5	2								x	x					
etc...															

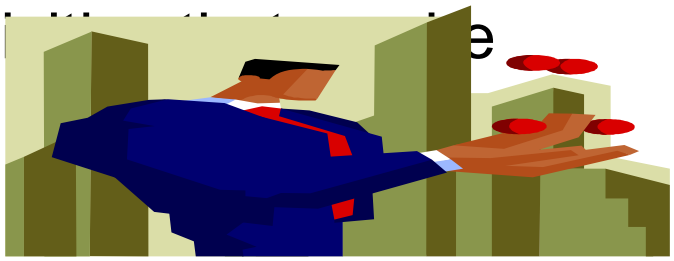


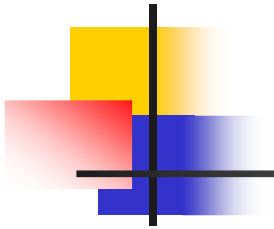
Risk Analysis

- **What:** The process of assessing identified risks to estimate their impact and probability of occurrence (likelihood).
  - **Likelihood** = The probability or chance of an event occurring (e.g., the likelihood that a user will make a mistake and, if a mistake is made, the likelihood that it will go undetected by the software)
  - **Impact** = The damage that results from a failure (e.g., the system crashing or corrupting data might be considered high impact)

Risk Analysis

- **Who:** PM, Tech Support, Sales, Engineers, QA;
- **When:** **(design)/coding/testing** stage(s);
- **Why:**
  - It helps us choose the best test techniques
  - It helps us define the extent of testing to be carried out
  - The higher the risk, the more focus given
  - It allows for the prioritization of the testing
  - Attempt to find the critical defects as early as possible
  - Are there any non-testing acti[illegible]e employed to reduce risk? e.g. inexperienced personnel





Example

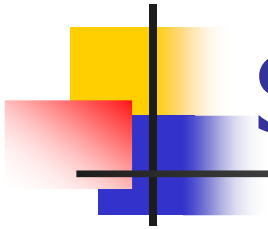
Scale 1-10

Test Case (Feature)	Likelihood of failure (Eng, QA, Tech Support)	Impact of failure (PM, Tech Support, Sales, QA)	Risk Factor
1. Login	2	10	20
2. Create database record	5	7	35
3. Modify database record	3	6	18

Exercise (20 mins in total)

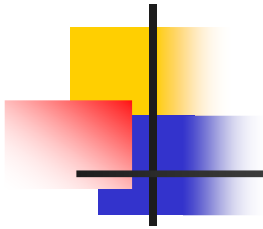
- Ten minutes to prepare the following:
- Can you work out your test plan on a recent or past project and share with other colleagues?
- The plan should include at least: testing scope, activates/items, features, testing resources/timelines, risk analysis
- *We will select two~three audiences to share their plan (10 mins)*

5 Testing Strategy and Testing Plan



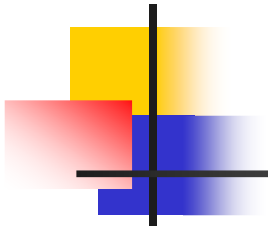
Strategic Approach to Testing - 1

- Testing begins at the component level and works outward toward the integration of the entire computer-based system.
- Different testing techniques are appropriate at different points in time.
- The developer of the software conducts testing and may be assisted by independent test groups for large projects.
- The role of the independent tester is to remove the conflict of interest inherent when the builder is testing his or her own product.



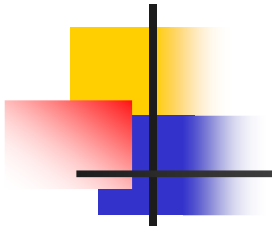
Strategic Approach to Testing (cont)

- Testing and debugging are different activities.
- Debugging must be accommodated in any testing strategy.
- Need to consider verification issues
  - are we building the product right?
- Need to Consider validation issues
  - are we building the right product?



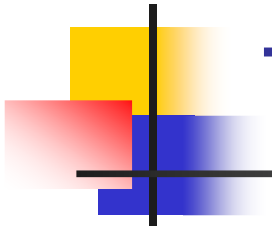
Strategic Testing Issues

- Specify product requirements in a quantifiable manner before testing starts.
- Specify testing objectives explicitly.
- Identify the user classes of the software and develop a profile for each.
- Develop a test plan that emphasizes rapid cycle testing.



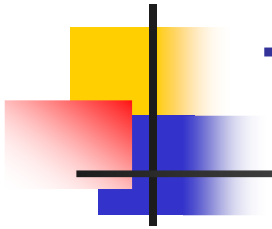
Strategic Testing Issues (cont)

- Build robust software that is designed to test itself (e.g. anti-bugging).
- Use effective formal reviews as a filter prior to testing.
- Conduct formal technical reviews to assess the test strategy and test cases.



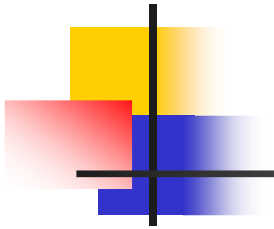
Testing Levels

- Various development models are there in the market
- Within each development model, there are corresponding levels/stages of testing
- There are four basic levels of testing that are commonly used within various models:
  - Component (unit) testing
  - Integration testing
  - System testing
  - Acceptance testing



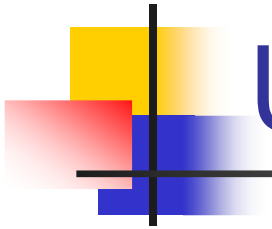
Testing Levels

- **Component testing:** The testing of individual software components.
- **Integration testing:** Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems.
- **System testing:** The process of testing an integrated system to verify that it meets specified requirements.
- **Acceptance testing:** Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.



Stages of Testing

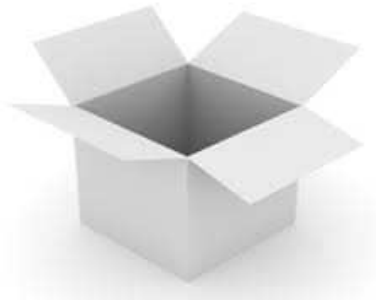
- Module or unit testing
- Integration testing
- Function testing
- Performance testing
- Acceptance testing
- Installation testing



Unit Testing

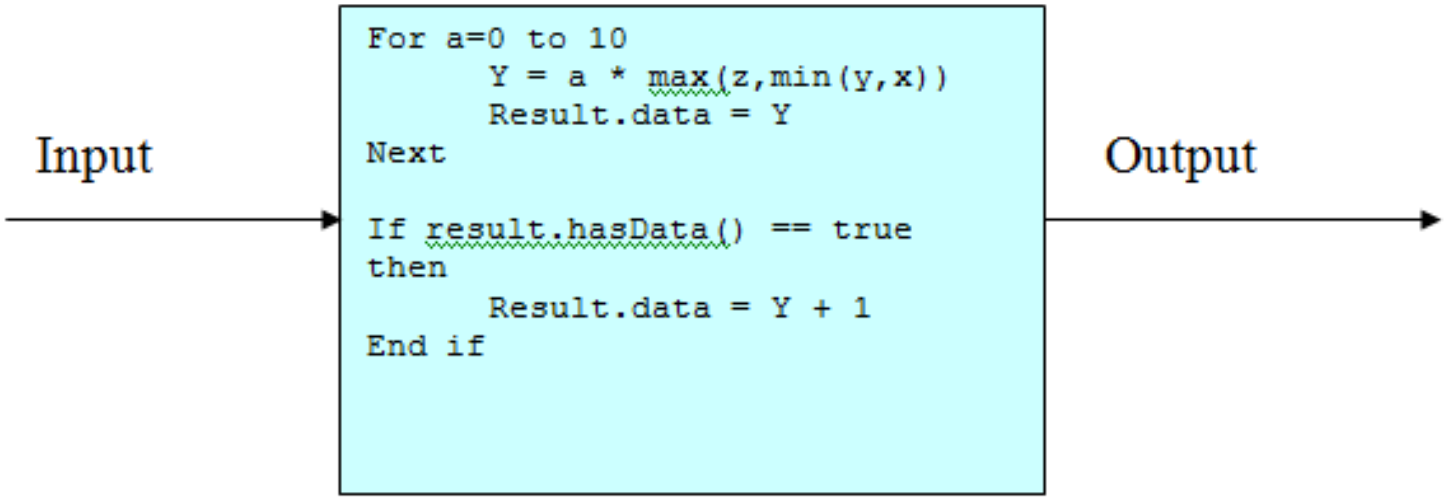
- Program reviews.
- Formal verification.
- Testing the program itself

Structural (White box) testing



- Testing based on an analysis of the internal structure of the component or system / architecture of the system, aspects such as a calling hierarchy, data flow diagram, design specification, etc.;
- May be performed at all test levels - system, system integration, or acceptance levels (e.g., to business models or menu structures);
- Structural techniques are best used after specification-based techniques;
- Can assist in measuring the thoroughness of testing by assessing the degree of coverage of a structure;
- Tools can be used to measure the code coverage.

Example



White box testing diagram

Black box testing

- The program is treated as black box;
- Inputs are fed into the program, outputs observed;
- Search for interesting and challenging input combinations and conditions – they are most likely to expose an error.

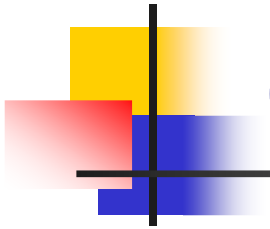
Three kinds of Black-box testing methods

Based on the 3 kinds of information:

- **Input coverage tests**, based on an analysis of the *intended inputs*, independent of their actions or outputs
- **Output coverage tests**, based on an analysis of the *intended outputs*, independent of their inputs or actions
- **Functionality coverage tests**, based on an analysis of the *intended actions/functions*, with or without their inputs and outputs.

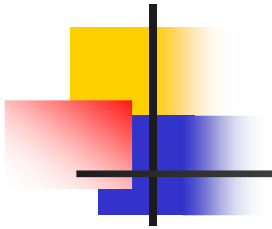
Unit Testing

- Does "Unit Testing" falls under white box or black box testing?



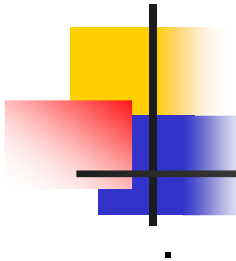
Go for black box or white box?

- Maximum # of logic paths - determine if white box testing is possible.
- Nature of input data.
- Amount of computation involved.
- Complexity of algorithms.



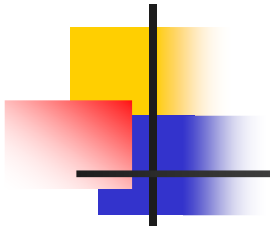
Unit Testing Details

- Interfaces tested for proper information flow.
- Local data are examined to ensure that integrity is maintained.
- Boundary conditions are tested.
- Basis path testing should be used.
- All error handling paths should be tested.
- Drivers and/or stubs need to be developed to test incomplete software.



Example

```
[TestFixture]
public class CalculatorTests
{
    [Test]
    public void TestPressEquals()
    {
        Calculator calculator = new Calculator();
        calculator.Enter(2m);
        calculator.PressPlus();
        calculator.Enter(2m);
        calculator.PressEquals();
        Assert.AreEqual(4m, calculator.Display);
    }
}
```



Generating Test Data

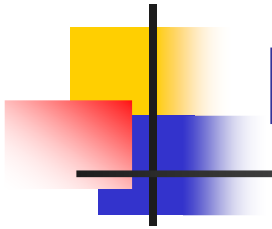
- Ideally want to test every permutation of valid and invalid inputs
- Equivalence partitioning it often required to reduce to infinite test case sets
  - Every possible input belongs to one of the equivalence classes.
  - No input belongs to more than one class.
  - Each point is representative of class.

Testing Data Guidelines

- **Simplicity:** To make tests readable, test data should be simple. From a testing point of view, if there is no difference between “24.54” and “1”, then we use 1 for human reading.
- **Heterogeneity:** To handle multiple inputs, use different values for each input. For example, $p(2, 3)$ is better than $p(2, 2)$, for $p(3, 2)$ is order irreversible, i.e. $p(2, 3) \neq p(3, 2)$.

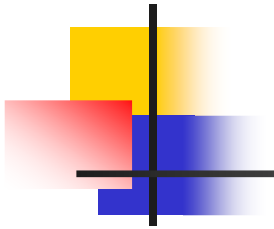
Testing Data Guidelines (cont'd)

- Realistic Data: Try to use realistic data which simulates how the system will be used.
- Evident Data: The variance between expected and actual results, if any, should be apparent when a program runs the unit test.



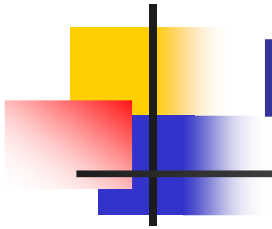
Regression testing (retesting)

- A test that was written when a bug was fixed. It ensure that this specific bug will not occur again. (retesting after changes are made.)
- The purpose of regression testing is to ensure that changes such as those mentioned above have not introduced new faults
- Applies to functional, non-functional and structural testing;
- Good candidate for automation.



Regression Testing

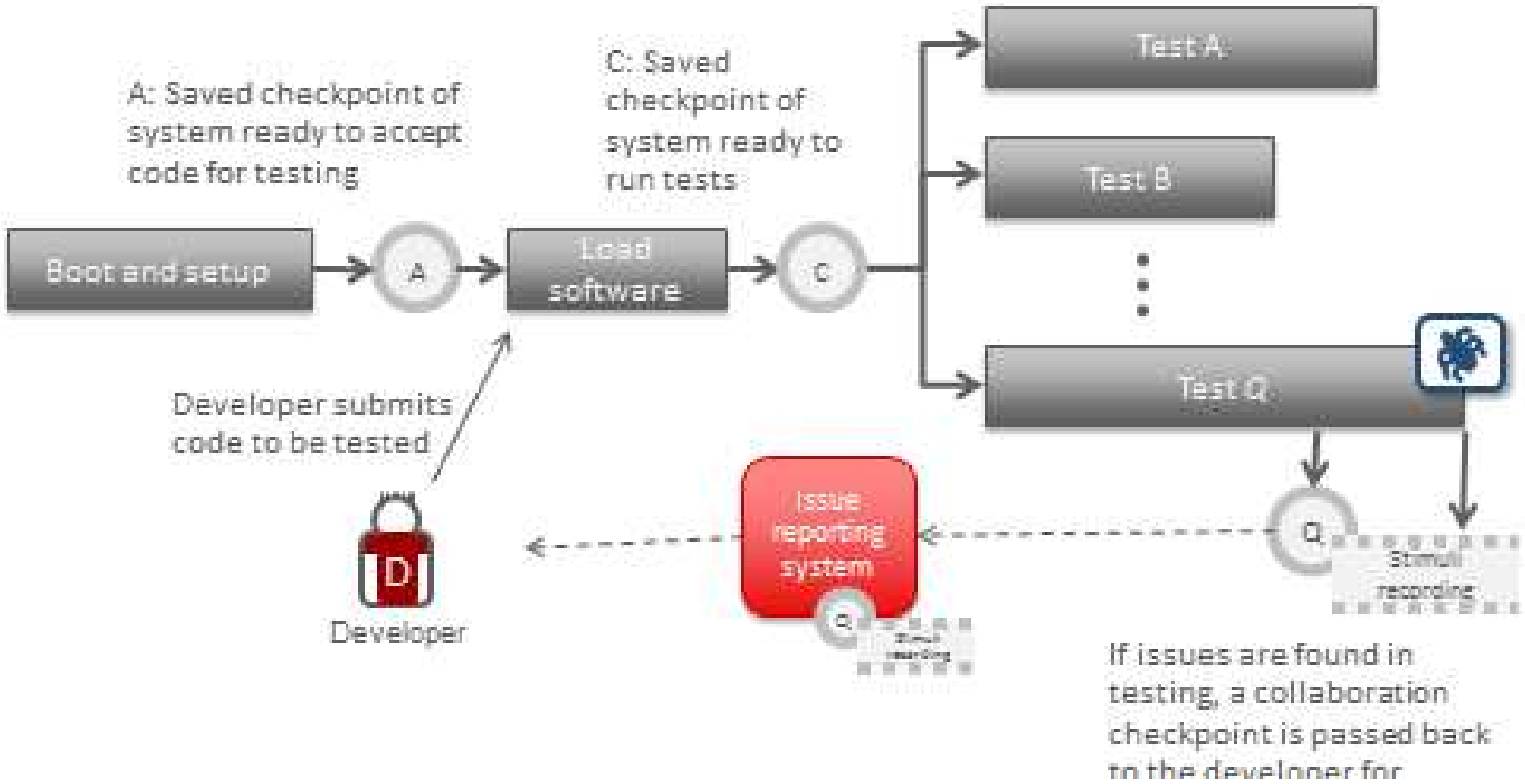
- Check for defects propagated to other modules by changes made to existing program
  - Representative sample of existing test cases is used to exercise all software functions.
  - Additional test cases focusing software functions likely to be affected by the change.
  - Tests cases that focus on the changed software components.

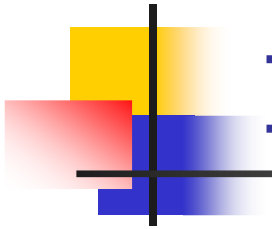


Discussion Points

- Could we reuse the same tests as the regression tests?
- Could we test the same areas as before, but we use different (increasingly complex) tests?

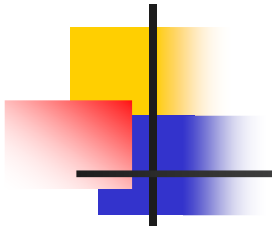
Example of Regression Testing





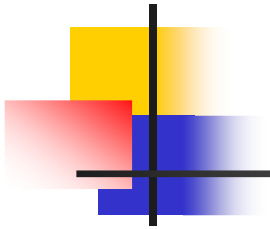
Integration Testing

- Bottom - up testing (test harness).
- Top - down testing (stubs).
- Big Bang.
- Sandwich testing.



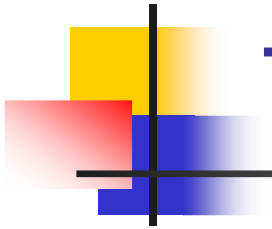
Bottom-Up Integration Testing

- Low level components are combined in clusters that perform a specific software function.
- A driver (control program) is written to coordinate test case input and output.
- The cluster is tested.
- Drivers are removed and clusters are combined moving upward in the program structure.



Top-Down Integration Testing

- Main program used as a test driver and stubs are substitutes for components directly subordinate to it.
- Subordinate stubs are replaced one at a time with real components (following the depth-first or breadth-first approach).
- Tests are conducted as each component is integrated.
- On completion of each set of tests and other stub is replaced with a real component.
- Regression testing may be used to ensure that new errors not introduced.



Testing Strategies

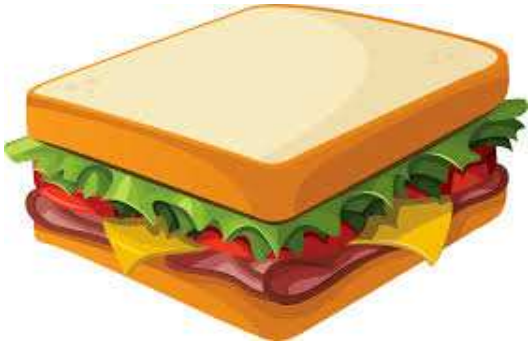
Incremental: testing modules as they are developed, each piece is tested separately. Once all elements are tested, integration/system testing can be performed.

Requires additional code to be written, but allows to easily identify the source of error

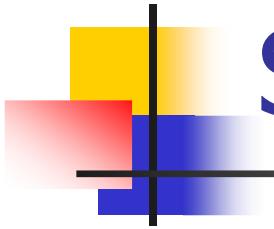
Big Bang: testing is performed on fully integrated system, everything is tested with everything else.

No extra code needed, but errors are hard to find.

Sandwich Testing

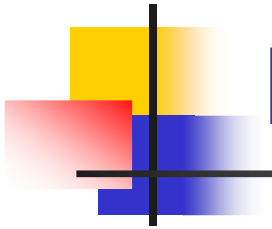


- Sandwich testing is a type of testing that consist of two parts, they are Top-down approach and Bottom-up approach.
- It combines the advantages of both Bottom-up testing and Top-down testing at a time.
- Sandwich approach is useful for very large projects having several subprojects.



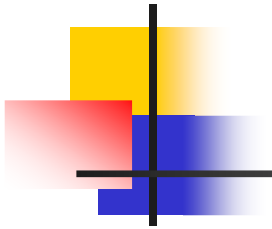
Summary

	Button UP	Top Down	Big Bang	Sandwich
Integration	Early	Early		Early
Stub	No	Yes	Yes	Yes
Parallelism	Medium	Low	High	Medium
Test Specification	Easy	Hard	Easy	Medium
Product Control Seq	Easy	Hard	Easy	Hard



Functional testing

- Testing based on an analysis of the specification of the functionality of a component or system.
- The functions are "what" the system does:
  - They are typically defined or described in work products such as a requirements specification, use cases, or a functional specification;
  - They may be undocumented;
  - Functional tests are based on both explicit and implicit features and functions;
  - They may occur at all test levels, e.g., tests for components may be based on a component specification;
- Functional testing focuses on the external behavior of the software (black-box testing).

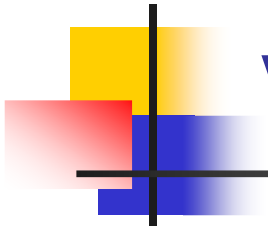


Non-Functional testing

- Focuses on "how" the system works;
- Non-functional tests are those tests required to measure characteristics of systems and software that can be quantified;
- These quantifications can vary and include items such as: response times, throughput, capacity for performance testing etc.
- Testing the attributes of a component or system that do not relate to functionality, e.g. reliability, efficiency, usability, maintainability, compatibility and portability.

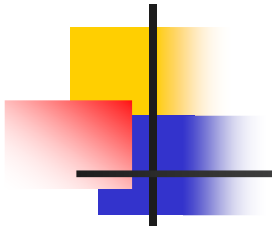
Smoke Test Cases

- A *smoke test* is a subset of the test cases that is typically representative of the overall test plan.
  - Good for verifying proper deployment or other non invasive changes.
  - Useful for verifying a build is ready to send to test.
  - Not substitute for actual functional testing.
  - *It is an analogy with electronics, where the first test occurs when powering up a circuit: if it smokes, it's bad.*



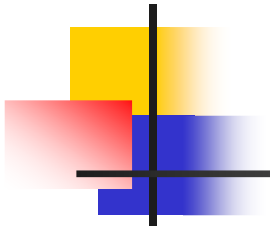
Validation Testing

- Ensure that each function or performance characteristic conforms to its specification.
- Deviations (deficiencies) must be negotiated with the customer to establish a means for resolving the errors.
- Configuration review or audit is used to ensure that all elements of the software configuration have been properly developed, cataloged, and documented to allow its support during its maintenance phase.



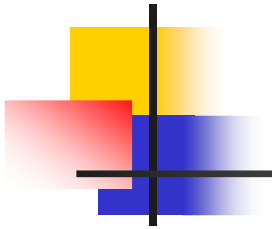
System Testing

- Recovery testing
  - checks system's ability to recover from failures
- Security testing
  - verifies that system protection mechanism prevents improper penetration or data alteration
- Stress testing
  - program is checked to see how well it deals with abnormal resource demands
- Performance testing
  - tests the run-time performance of software



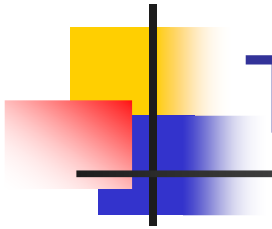
Acceptance Testing

- Making sure the software works correctly for intended user in his or her normal work environment.
- Alpha test
  - version of the complete software is tested by customer under the supervision of the developer at the developer's site
- Beta test
  - version of the complete software is tested by customer at his or her own site without the developer being present



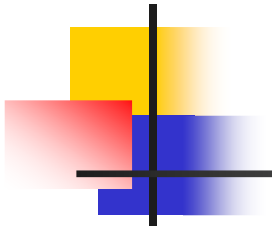
Testing Life Cycle

- Establish test objectives.
- Design criteria (review criteria).
  - Correct.
  - Feasible.
  - Coverage.
  - Demonstrate functionality .
- Writing test cases.
- Testing test cases.
- Execute test cases.
- Evaluate test results.



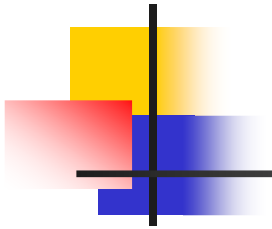
Test Team Members

- Professional testers.
- Analysts.
- System designers.
- Configuration management specialists.
- Users.



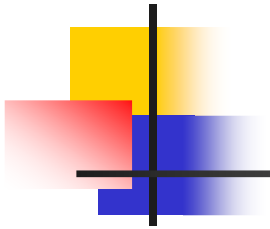
Testing Tools

- Simulators.
- Monitors.
- Analyzers.
- Test data generators
- Test Management



Defect Tracking

- The *defect tracking system* is a program that testers use to record and track defects. It routes each defect between testers, developers, the project manager and others, following a workflow designed to ensure that the defect is verified and repaired.
  - Every defect encountered in the test run is recorded and entered into a defect tracking system so that it can be prioritized.
  - The defect workflow should track the interaction between the testers who find the defect and the programmers who fix it. It should ensure that every defect can be properly prioritized and reviewed by all of the stakeholders to determine whether or not it should be repaired. This process of review and prioritization referred to as *triage.*



Test Automation

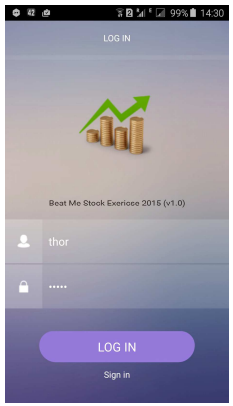
- *Test automation* is a practice in which testers employ a software tool to reduce or eliminate repetitive tasks.
  - Testers either write scripts or use record-and-playback to capture user interactions with the software being tested.
  - This can save the testers a lot of time if many iterations of testing will be required.
  - It costs a lot to develop and maintain automated test suites, so it is generally not worth developing them for tests that will executed only a few times.

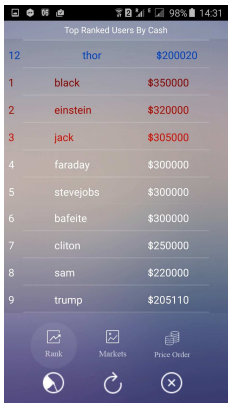
Discussion

Your boss put you in charge of testing a new system, to be developed by an internal team.

Before you start testing, what need to be done?

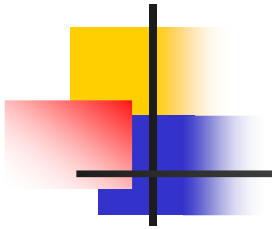
How to get ready?





6 Test Case Optimization

Equivalence class partitioning



- A black box test design technique in which test cases are designed to execute representatives from equivalence partitions. In principle, test cases are designed to cover each partition at least once.
- Creates the minimum number of black box tests needed to provide minimum test coverage
- Steps:
  - Identify equivalence classes, the input values which are treated the same by the software:
    - Valid classes: legal input values;
    - Invalid classes: illegal or unacceptable input values;
  - Create a test case for each equivalence class.

Equivalence class partitioning (cont'd)

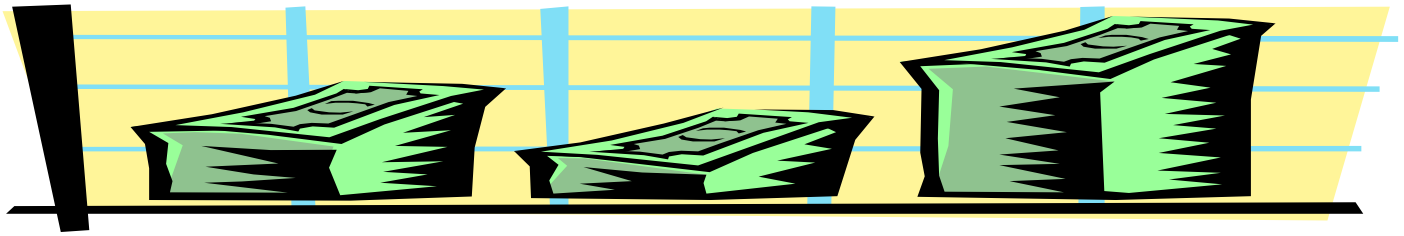
<$1000 $1000-70000 >$70000

Equivalence partition (class):

– A portion of an input or output domain for which the behavior of a component or system is assumed to be the same, based on the specification.

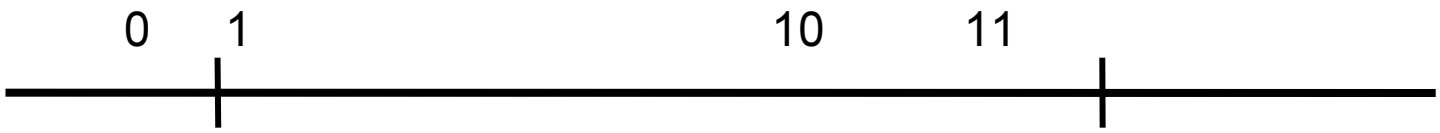
Boundary value testing

- A black box test design technique in which test cases are designed based on boundary values.
- Each input is tested at both ends of its valid range(s) and just outside its valid range(s). This makes sense for numeric ranges and can be applied to non-numeric fields as well. Additional issues, such as field length for alphabetic fields, can come into play as boundaries.
- Boundary value: An input value or output value, which is on the edge of an equivalence partition or at the smallest incremental distance on either side of an edge, for example the minimum or maximum value of a range.



Boundary value testing (cont'd)

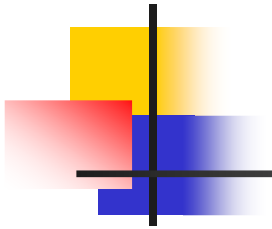
- Run test cases at the boundary of each input:
  - Just below the boundary;
  - Just above the boundary;
- The focus is on one requirement at a time;



- Can be combined across multiple requirements – all valid minimums together, all valid maximums together;
- Invalid values should not be combined.

What Test Case?

- A field can accept integer values between 20 and 50.
- What tests should we try?



Testing Integer-Input Tests

Common answers:

Test	*Why it's interesting*	*Expected result*
20	Smallest valid value	Accepts it
19	Smallest -1	Reject, error msg
0	0 is always interesting	Reject, error msg
Blank	Empty field, what's it do?	Reject? Ignore?
49	Valid value	Accepts it
50	Largest valid value	Accepts it
51	Largest +1	Reject, error msg
-1	Negative number	Reject, error msg
4294967296	2^32, overflow integer?	Reject, error msg

Too Many Combinations

- **Combinations of configuration parameter values**

  For example, **telecom app** may be configured to work with different types of call (local, long distance, inter), billing (low bandwidth, high bandwidth), access (A plan, B plan, C plan).

- **Combinations of test sequences**

➔ For example, in mobile application, there are many steps and each step has many choices.

☹ If there are k choices in each step, after n steps there are k x k x .. x k = $\mathbf{k^n}$ different test sequences.

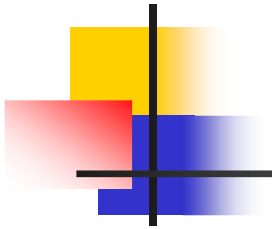
☹ It is not practical to test all these sequences!!

Combinatorial Example: Find

The Find takes 3 inputs:

- **Find what** : a text string
- **Match case:** yes or no
- **Direction:** up or down

We simplify this by considering only 3 values for the text string, say "Lee", "Ip" and "Chan".

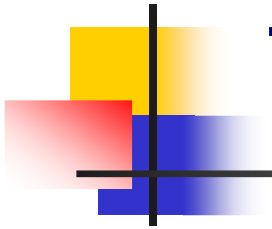


Combinatorial Example

1. How many combinations of these 3 variables are possible?
   - **Find what** has 3 values (Lee, Ip, Chan) (L I C)
   - **Match case** has 2 values (Yes / No) (Y N)
   - **Direction** has 2 values (Up / Down) (U D)

2. List ALL the combinations of these 3 variables (total 12 cases)

L Y U	I Y U	C Y U
L Y D	I Y D	C Y D
L N U	I N U	C N U
L N D	I N D	C N D

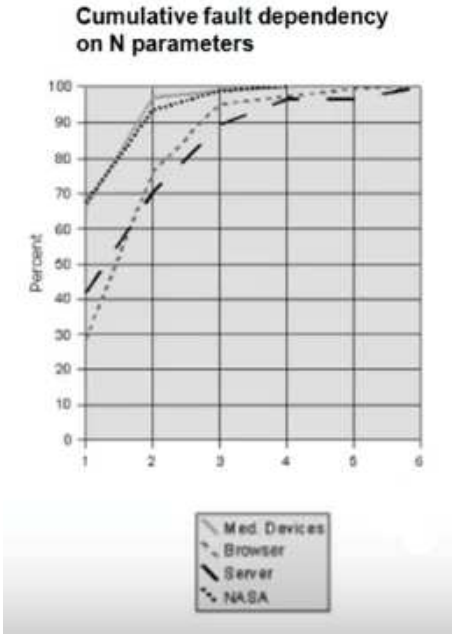


Test Cases

	Input			Output
TC	Find what	Match case	Direction	Expected result
1	Lee	yes	up	find Lee
2	lp	no	down	find lp
3	abc	yes	up	‘invalid input’

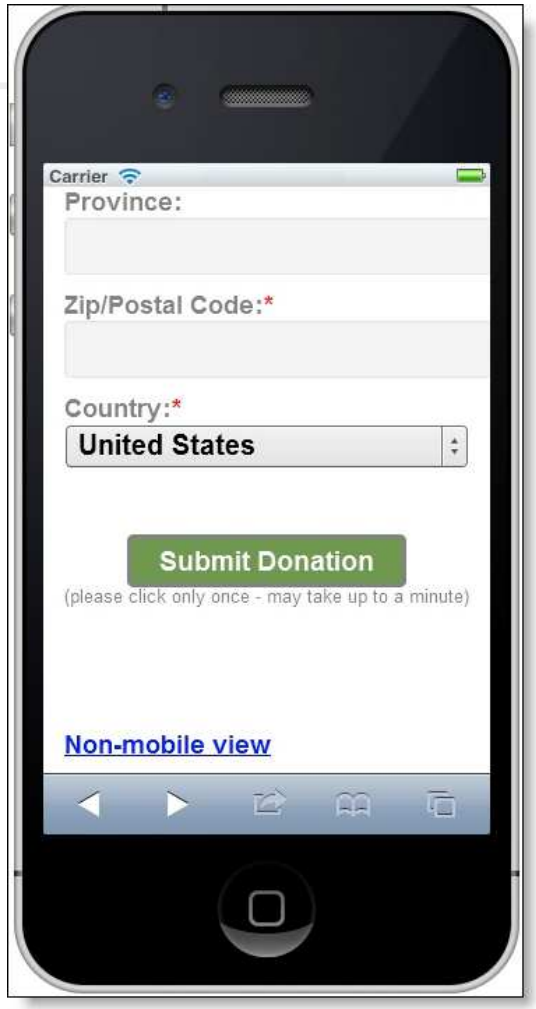
Pairwise testing

- All-pairs testing or pairwise testing is a combinatorial method of software testing that, for *each pair* of input parameters to a system (typically, a software algorithm), tests all possible discrete combinations of those parameters.
- Using carefully chosen test vectors, this can be done much faster than an exhaustive search of all combinations of all parameters, by "parallelizing" the tests of parameter pairs.



How and Why Pairwise Testing Works

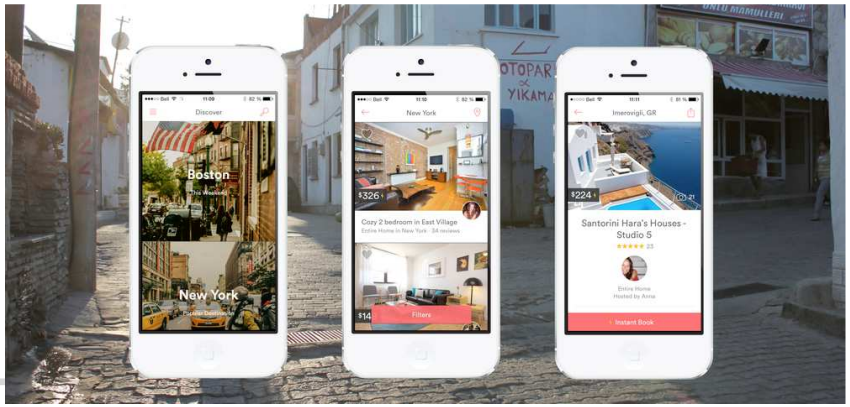
- It is based on the observation that *most faults are caused by interactions of at most two factors.*
- For example, *there could be higher chance for "Zip+Country" to get error than for "Province+Zip+Country"*
- Why?



3 Variables with 2 Values Each

- P1-P2, P2-P3, P1-P3

Example



- A Simple Airbnb-Like App

Provide additional information about ticket

Bed linen ☐
Tea ☐

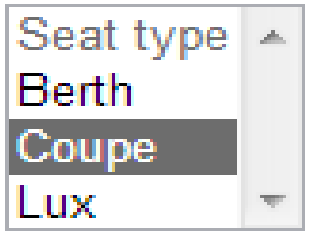
Next

- Combinations: 2x2 = 4

Example

Combination number	Bed linen	Tea
1	checked	checked
2	unchecked	checked
3	checked	unchecked
4	unchecked	unchecked

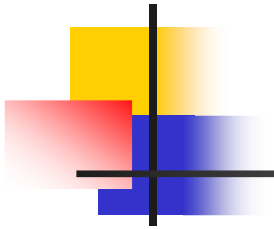
Example



Provide additional information about ticket

Bed linen ☐
Tea ☐
Gypsies ☐
Demobees ☐

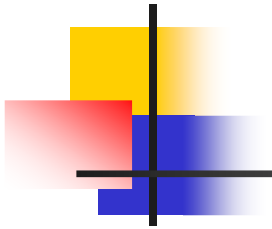
Next



Example

Combinations (all): 3x2x2x2x2 = 48

Combination number	Seat type	Bed linen	Tea	Gypsies	Demobees
1	Berth	checked	checked	checked	checked
2	Coupe	checked	checked	checked	checked
3	Lux	checked	checked	checked	checked
4	Berth	unchecked	unchecked	unchecked	unchecked
5	Coupe	unchecked	unchecked	unchecked	unchecked
6	Lux	unchecked	unchecked	unchecked	unchecked
7	Berth	unchecked	unchecked	unchecked	checked
8	Coupe	unchecked	unchecked	unchecked	checked
…	…	…	…	…	…



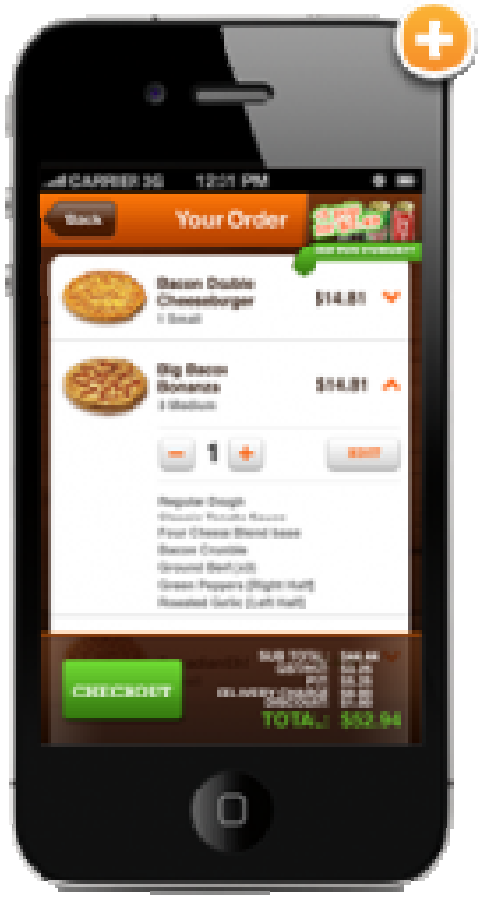
Example: Pairwise Applied

Combinations (pairwise technique applied) = 6

Combination	Seattype	Bedlinen	Tea	Gypsies	Demobees
1	Berth	checked	checked	checked	checked
2	Berth	unchecked	unchecked	unchecked	unchecked
3	Coupe	checked	unchecked	checked	unchecked
4	Coupe	unchecked	checked	unchecked	checked
5	Lux	checked	checked	unchecked	unchecked
6	Lux	unchecked	unchecked	checked	checked

For more information: http://www.pairwise.org/

Another Example: Pizza App



Step 1 Select your favorite size and pizza crust

Large Original Crust

Step 2 Select your favorite pizza toppings from the pull down

For a regular pizza, do not add toppings

I want to add or remove toppings on this pizza -- add on whole or half pizza.

Add toppings whole pizza

Extra Cheese Remove

Bacon Remove

Black Olives Remove

Add toppings 1st half

Add toppings 2nd half

Step 3 Select your pizza instructions

I want to add special instructions for this pizza -- light, extra or no sauce; light or no cheese; well done bake

Regular Sauce | Normal Cheese | Normal Bake | Normal Cut

Step 4 Add to order

Quantity 1

Simplified pizza ordering:

6x4x4x4x4x3x2x2x5x2

= 184,320 possibilities

How to minimize the number of test cases?

Try to sample from the input space in way that assures a certain level of combination coverage.

1. Test the **most common or important combination**, and then vary one or more parameters for the next test.

   All singles: *all individual valid values tested*

   For the **Find** example, use 3 test cases:

   (L, Y, U)
   (I, N, D)
   (C, Y, U)

2. Test all pair-wise combinations (given any 2 parameters, every combination of values of these two parameters are covered in at least one test case)

   All pairs (2-way) → *all pairs of valid values*

How to minimize the number of test cases?

3. ***All triples (3-way)*** *→ all triplets of valid values*

For the ordering Pizza example,

4. ***All N-tuples*** *→ all combinations of valid values*

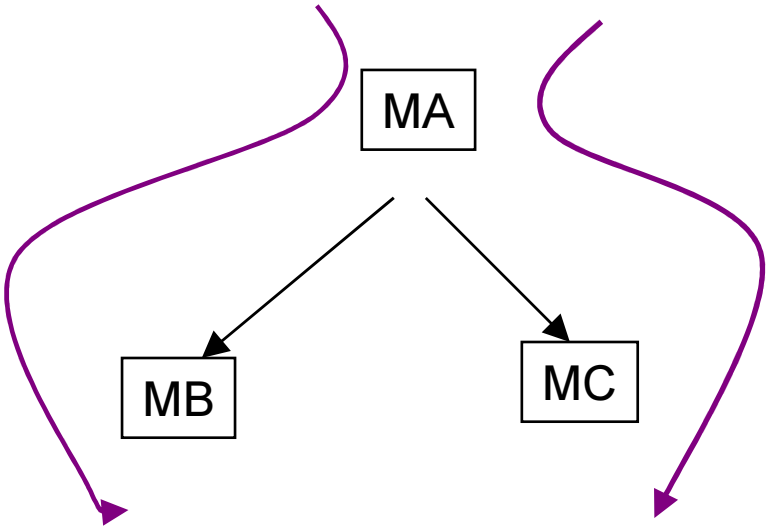
(N is the number of input parameters.)

Every possible combination of values of the parameters must be covered

Too many, *Not practical!*

Call Coverage

- Measure whether we executed each function call.
- The hypothesis is that faults commonly occur in interfaces between modules.
- Also known as **call pair coverage** (try to achieve 100% call coverage (test all calls)

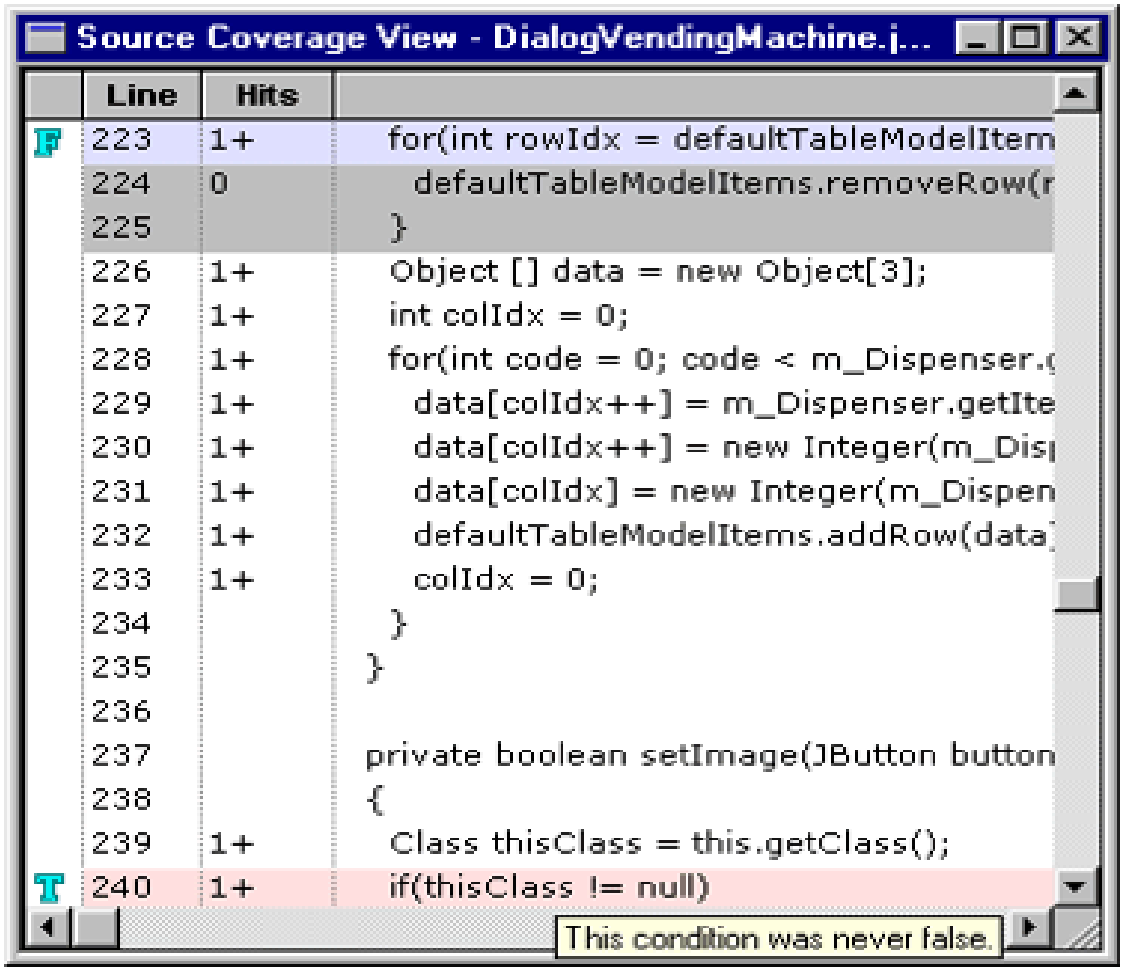


MA calls MB, and
MA also calls MC

Test tool computing the coverage

JCover for Java

Identify the statements executed during testing.

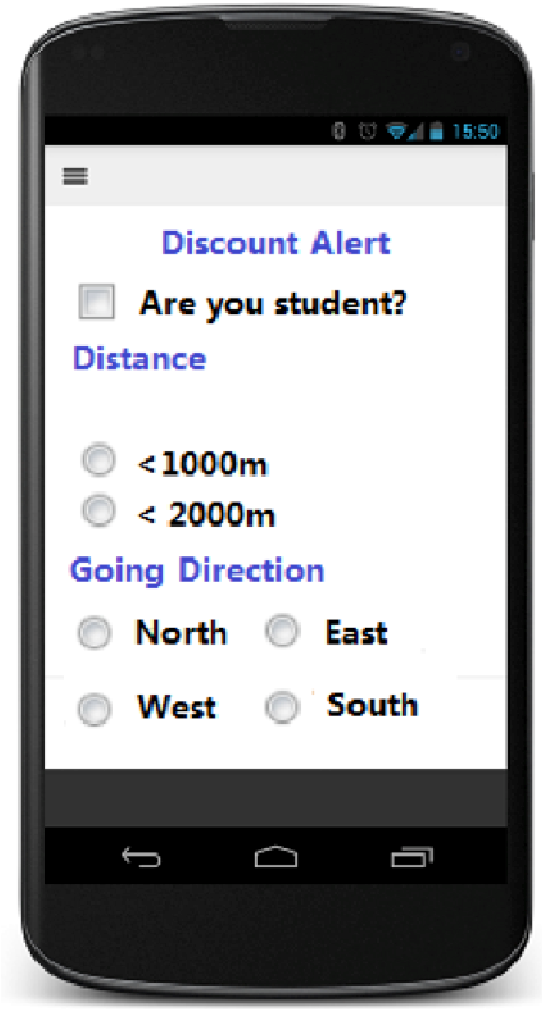


Exercise (10 mins)

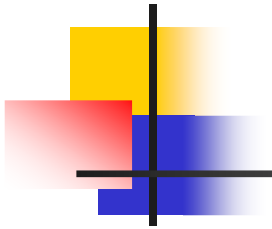
Use pair-wise technique to test the following app

Find the number of exhaustive test ?

Multiplying the two largest values to find out the number of pair wise test?



7 Measurement and Defect



Is this good results?

- A team testing 50,000LOC of code
- Created 1,200 test cases
- Found 99 defects
- Defects were fixed and no failures appeared in further testing

No more defects in the code?

Test Management Need Metrics

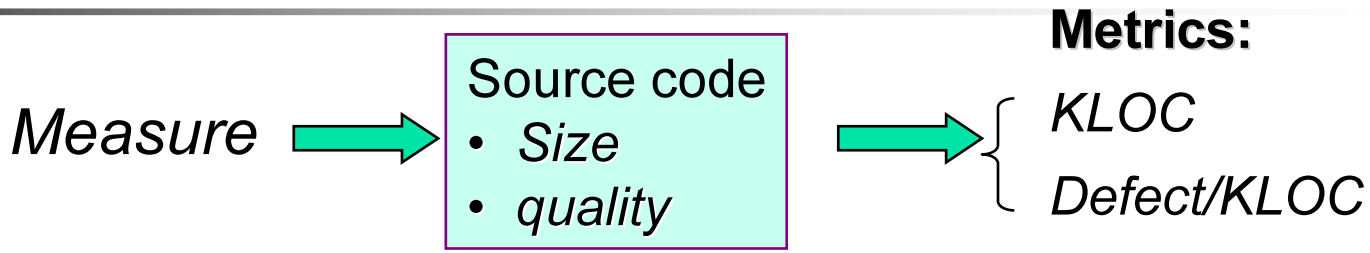
To be effective, managers must have access to the right information for making the crucial decisions.

Manager need to answer:

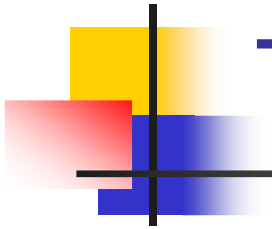
- Is the system ready to go live?
- If I go live now what risk is associated with that?
- What coverage have we achieved in our testing to date?
- How much more testing is there to do?
- Can I prove the system is really tested?
- What is the impact of this change and what must be retested?

"One accurate measurement is worth 1000 expert opinions" *Grace Murray Hopper, Rear Admiral, US Navy*

Example Metrics



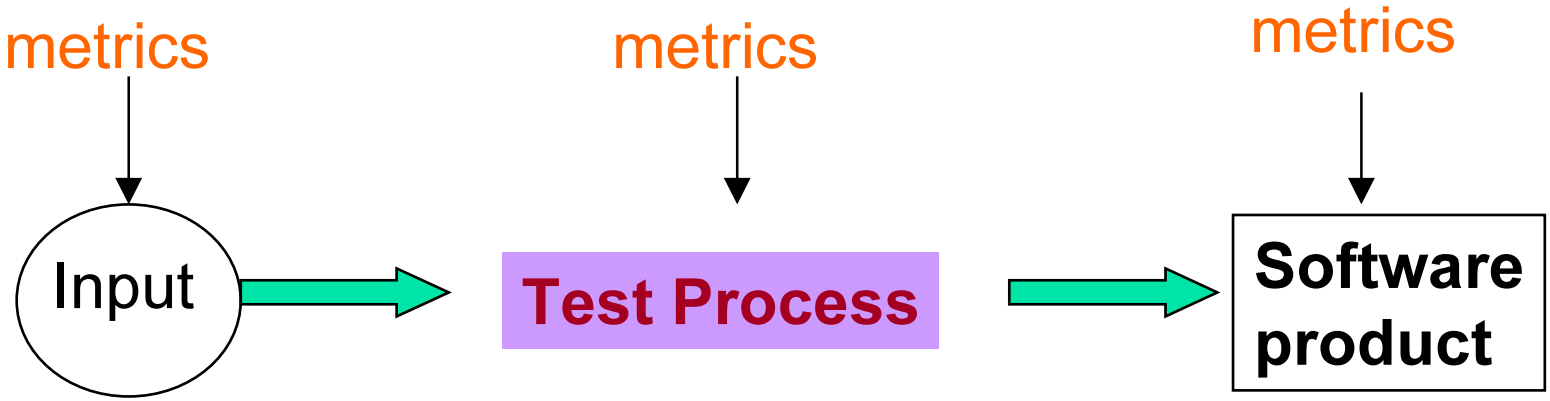
Entity	Attribute	Metric
Source code	Size	KLOC (line count)
Source code	Quality	defect/KLOC
Testing process	Duration	time in hours from start to finish
Testing process	Efficiency	test case/ defect



Tools

- You might want to consider Mylyn.
- It is a task based plugin for Eclipse.
- It lets you get a list of bugs assigned to you and then Mylyn tracks the time you spend on it.

What to Measure?



3 Types of Metrics

Product metrics: metrics related to test results or the quality (internal characteristics) of the product being tested [related to output of the test process]

Process metrics: metrics used to assess the effectiveness of the testing process.

Resource metrics (also called **project metrics):** metrics used to assess the cost and productivity of testing [related to input of the test process]

Product Metrics

1. Defect density
2. Defect age
3. Defect response time
4. Defect cost

Bug Free Software

Process Metrics

Coverage Metrics

Instruction coverage

Path coverage

Requirement coverage

Effectiveness Metrics

%Defects uncovered in testing

Test efficiency

Defect removal effectiveness (DRE)

% test cases successfully executed

% fixed defects

Resource Metrics

Resources consumed in testing process and the productivity of testing

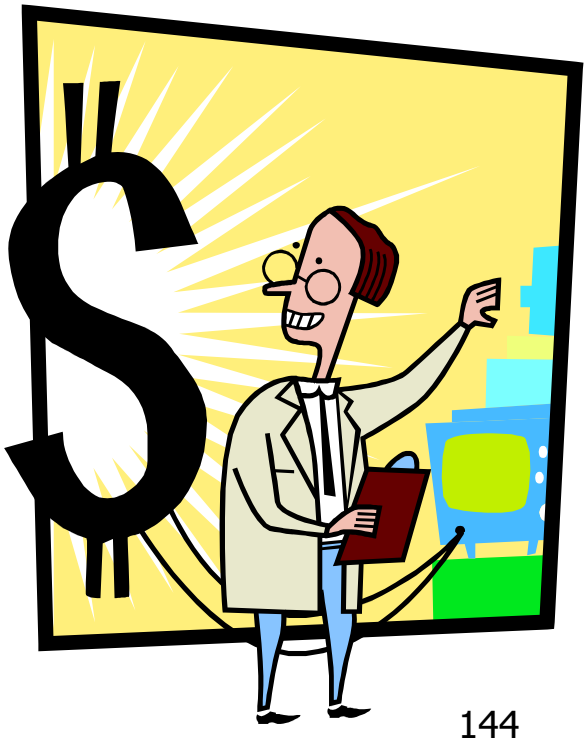
Relative test cost

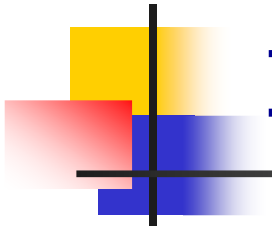
Cost to locate defect

%Achieving budget

% test cases prepared

Productivity



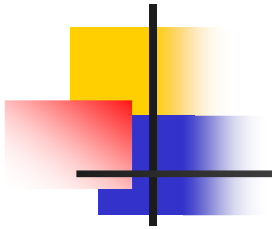


1. Defect Density

Defect density = number of defects detected / system size (e.g. defect/KLOC, defect/FP)

- The higher the number of defects, the poorer the product quality.
- Defect density can be used to perform the following:
  - predict remaining defects by comparison with expected defect density;
  - determine if sufficient testing has been completed based on predetermined goals;
  - establish standard defect densities for comparison and prediction.

Strength: good correlation to the ability of the test process to eliminate defects.



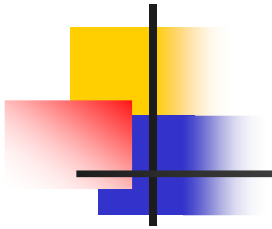
1. Defect Density

According to a study of 110 projects:

- DD range from 0.05 to 50 defects/KLOC
- Languages: C, Java, C++, Other

According to Steve McConnell,

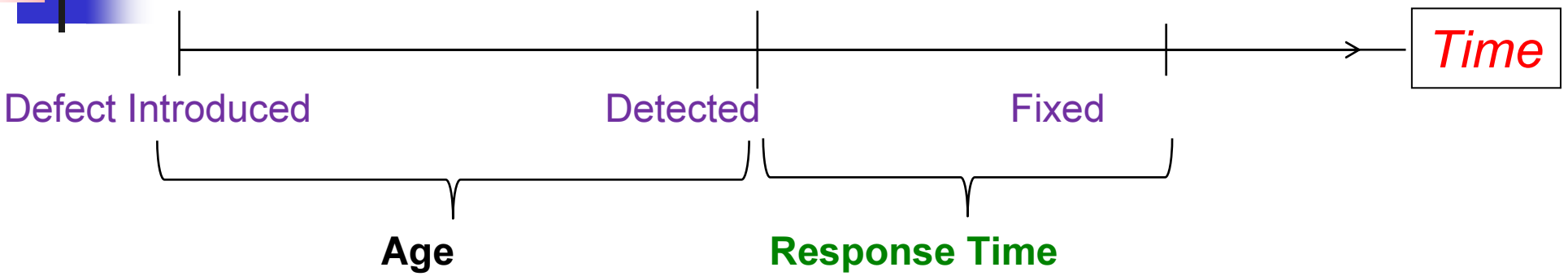
- Industry data is 1-25 defects/KLOC.



2. Defect Age

- Defects are injected and removed at different phases of a software development cycle
- The cost of each defect injected in phase X and removed in phase Y increases with the distance between X and Y
- An effective testing method would find defects earlier than a less effective testing method would.

2. Defect Age

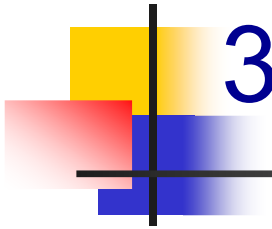


Defect age = time between when a defect is **introduced** to when it is detected

Defect age is the difference of the numbers corresponding to phase introduced & phase detected.

Average defect age=phase detected-phase introduced
number of defects
(Summed over all the defects)

High number means V&V should be done earlier.



3. Defect Response Time

Defect response time = time between when a defect is **detected** to when it is fixed or closed

The defect response time should not be too long; otherwise, the users will be dissatisfied (take too long to fix defects).

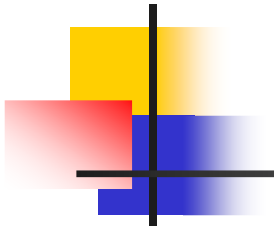
4. Defect Cost

Defect cost = cost to analyze the defect +
cost to fix it +
cost of failures incurred due to the defect

This metric provides data to calculate cost-benefit of any testing improvement project.

Example:

Will introducing new test tool bring benefits more than its cost?



Suppose

"We have executed 90% of the test cases and 80% passed."

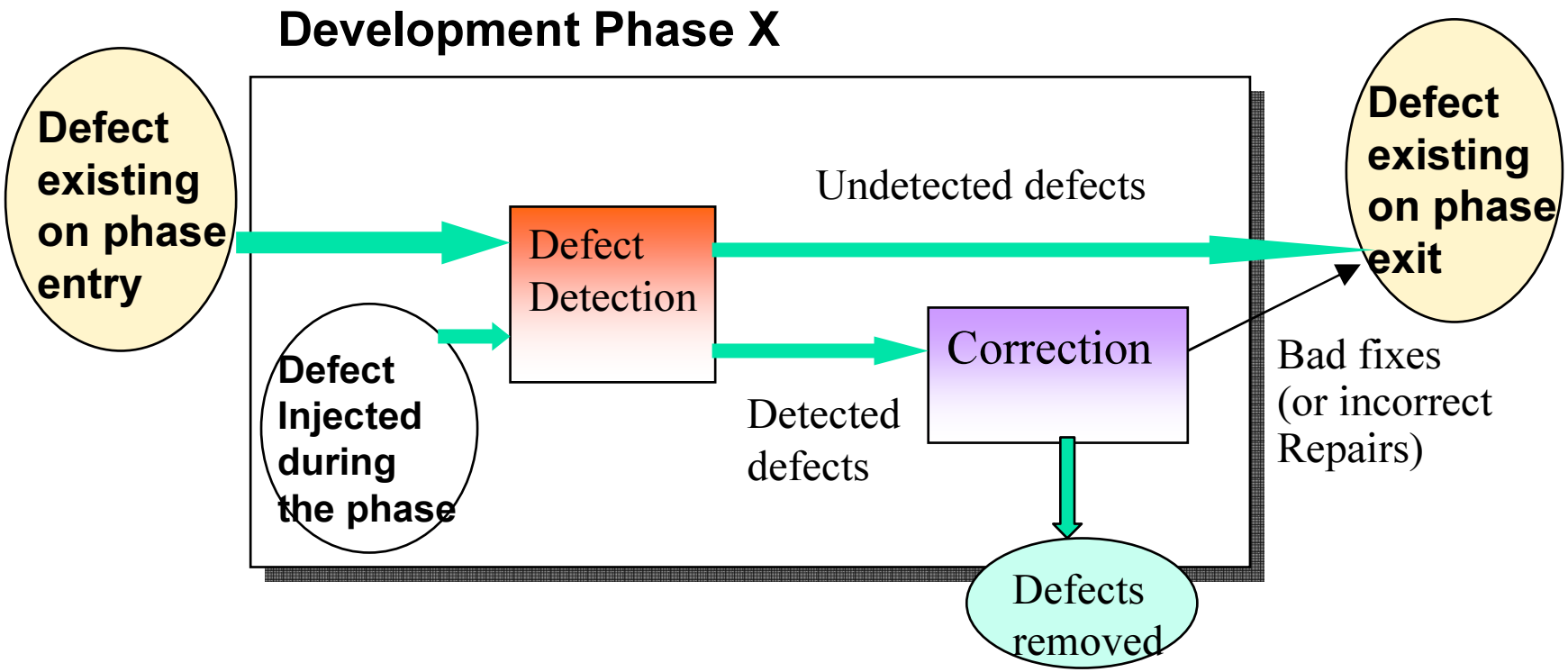
This means nothing unless it is known 'what' was tested, and how good were the test cases.

=> **We need coverage metrics (process metrics) to tell us what was tested.**

Question

Do you know how many test cases used by Microsoft for Office 2007?

Defect Removal Model



Defects removed = defects detected – bad fixes

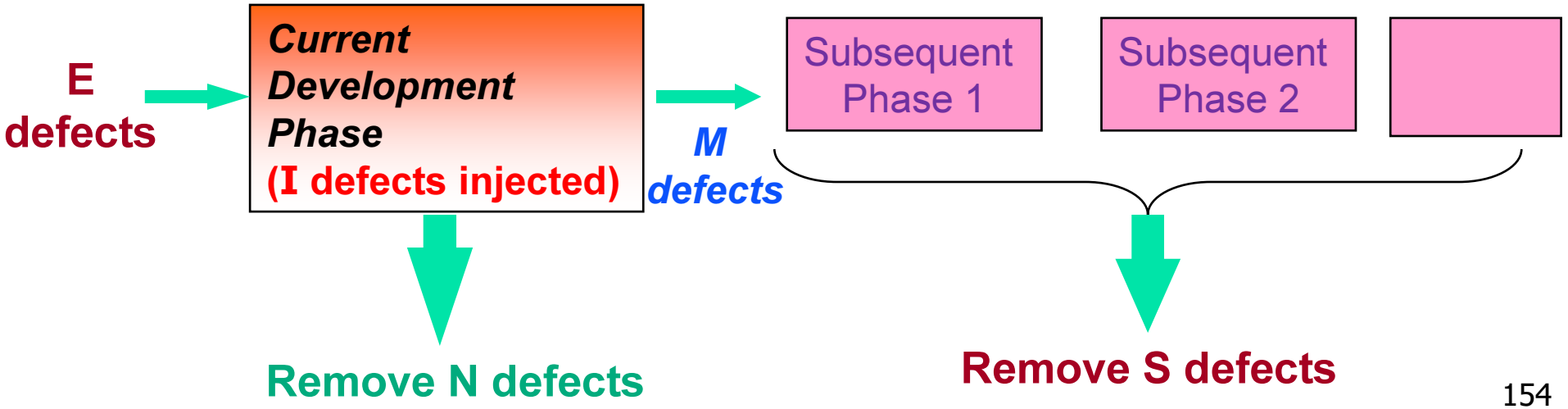
Defects at the **exit** of a development phase
= Defects escaped from previous phase + Defects injected in current phase - *Defects removed* in current phase

Defect Removal Effectiveness (DRE)

$$DRE = \frac{N}{(N+M)} \times 100\%$$

where

- N is number of defect removed by this development phase.
- But, since we don't know M, we use S to approximate M.
- S is number of defect removed by subsequent phases, which are **present** at the exit of the current development phase.



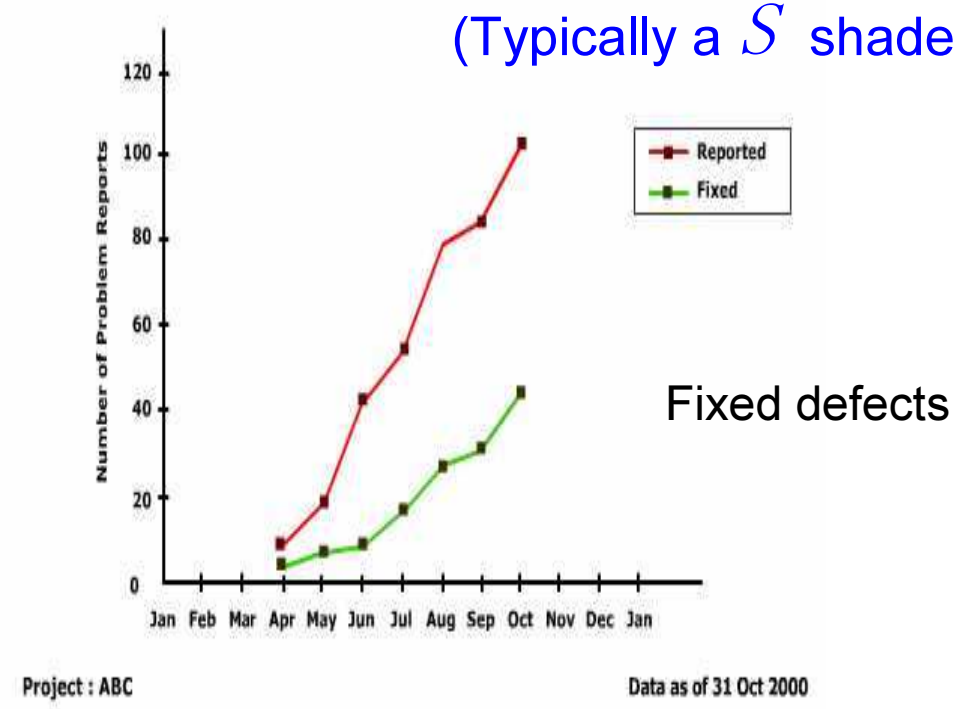
% Fixed Defects

$$\%\text{Fixed defects} = \frac{\text{Fixed defects}}{\text{Total defects}}$$

- From defect log.
- Look at the trend.
- Falling % should be a signal for alarm
- ☹ **Weakness:** The focus should be on the defect trend, rather than the %.

Example: Trend of Fixed Defects

(Typically a S shade curve)



Fixed defects

What Defects to Fix First?

Prioritize Defect

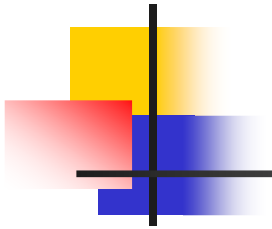
- Assign each defect a severity rating and a likelihood rating.

Priority = severity x likelihood

- Severity rating of 1 is the most severe.
- The defect with the 1 rating should be fixed first. Lower severity defects allow longer time to fix.

Severity Rating	Value
Hang	1
Loss, no workaround	2
Loss with Workaround	3
Inconvenient	4
Enhancement	5

Likelihood Rating	Value
Always	1
Usually	2
Sometimes	3
Rarely	4
Never	5



Relative Test Cost

Relative test cost = $\frac{\text{test cost}}{\text{total system cost}}$.

- test cost should be between 15-40% of total development effort

  For critical systems (financial systems), testing can consume > 70% of the budget

- ☺ **Strength:** shows the amount of the development effort that is allocated to testing. It provides an indication as to the extent of testing
- ☹ **Weakness:** there may not be a direct relationship between the effectiveness of testing and the amount of time/effort allocated to testing.

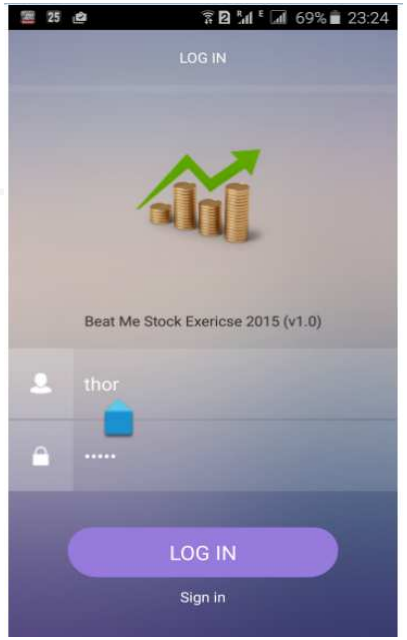
8 Government Initiative

Mobile Applications Testing Competency Training for SME

- This government-funded project aims to support a thriving ICT industry, through developing, training and nurturing ICT SME talents in mobile apps testing.
- We offer free mobile app testing seminars and training workshops to SMEs focusing on mobile apps development.

Coming Workshops

- Demo Mobile App
- Hands-on Approach:
  - Distributed JMeter
  - GUI Testing
  - Power Consumption
- Virtual Machine Environment
- Training Video



Workshop Highlight

Department of Computing
電子計算學系

Home | About MobAppTest | App Testing | Report | Workshops | Register | TMMi® | Video Training | Contributors

Together, we achieve higher software quality

In 2015, the Hong Kong Polytechnic University launched a project called **Mobile Applications Test Competency Knowledge Enhancement** for SME. This project is organised by the Hong Kong Polytechnic University, supported by Hong Kong Computer Society and funded by the SME Development Fund of the Trade and Industry Department, HKSAR Government. The project is to connect those SMEs who build mobile applications, provide open-source mobile testing toolkits, support local app developers, and introduce a proven international software testing framework TMMi®.

Thank you

- *Your feedback is important to us. Please fill in the evaluation form.*