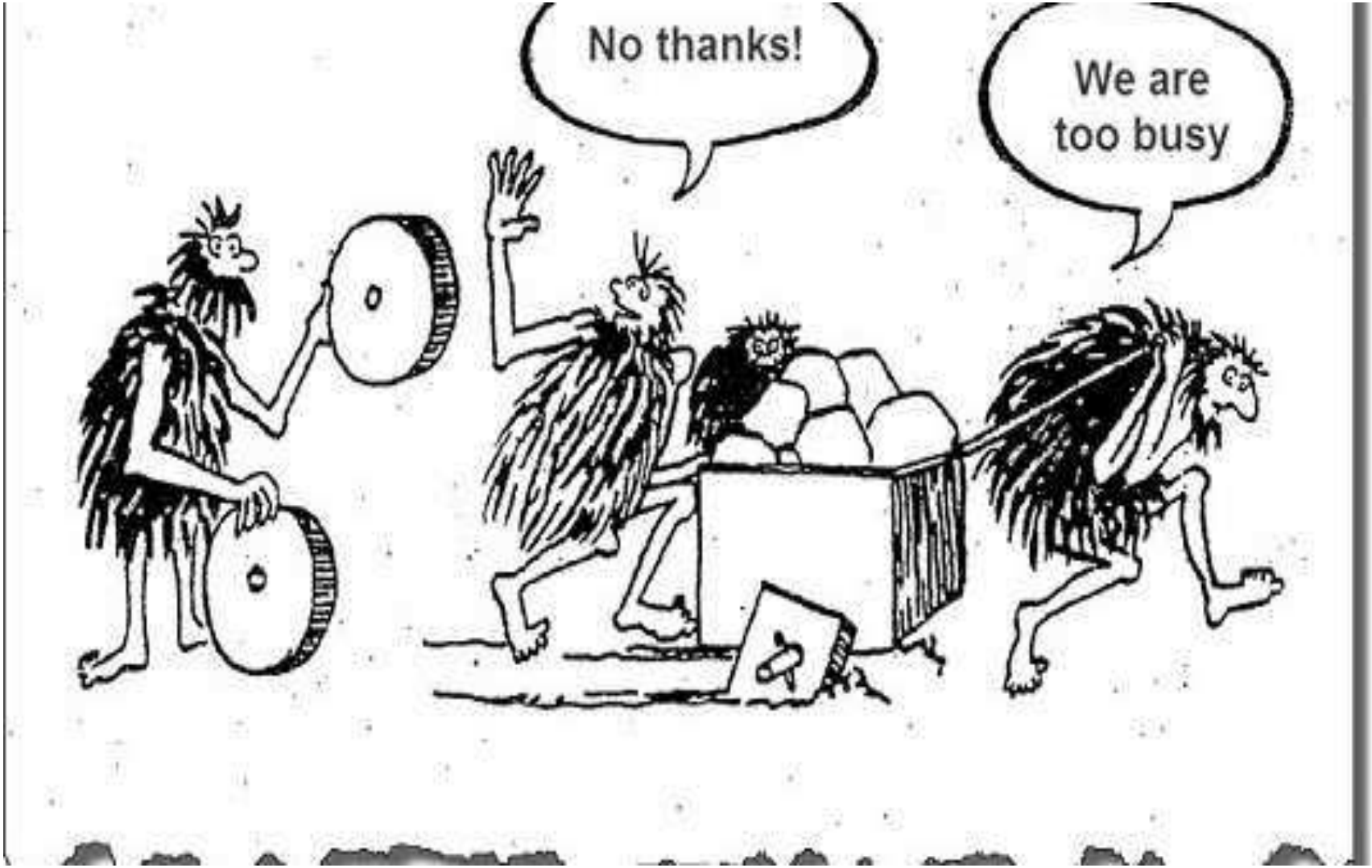
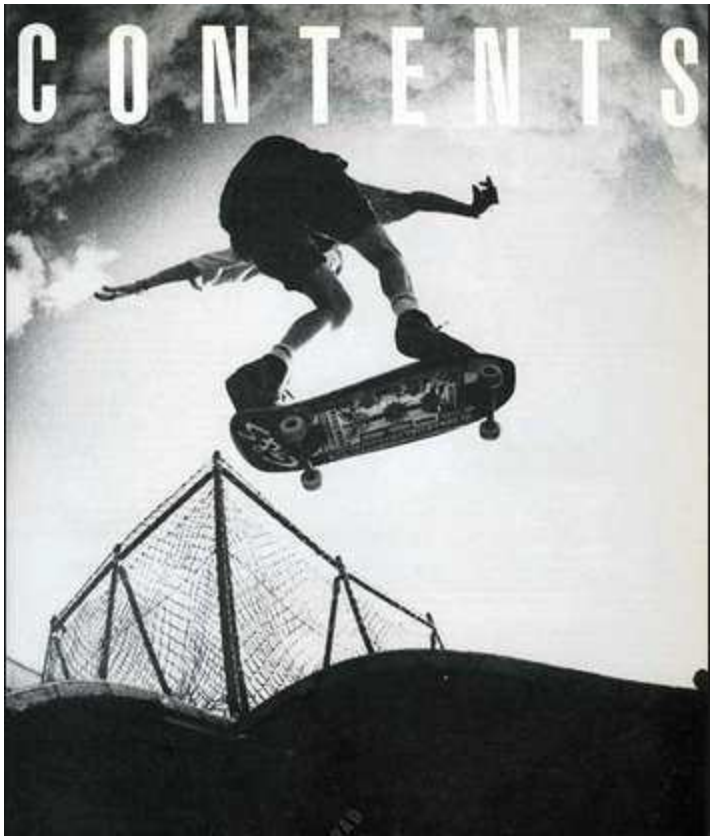

DEVELOPMENT
DATABASE
SCALABILITY
DATASETS
REFERRED
RELATED
COMBINATORIAL
DEBUGGER
ASSURANCE
ACCEPTANCE
REGRESSION
PRECONDITIONS
FUNCTION
FAULT
VERIFICATION
SOFTWARE
USABILITY
METHODS
COMMON
STAKEHOLDERS
OFFERED
ACTUAL
TESTING
MAINTAINABILITY
INTEGRATION
VALIDATION
CONDITION
INFORMATION
TEST
EXECUTION
ERROR

Why are we Training ☺





Why Testing is necessary

Testing Techniques

Test Planning

Test Specification and Execution

Psychology of Testing

Defect Management

Test Automation

What is Testing?

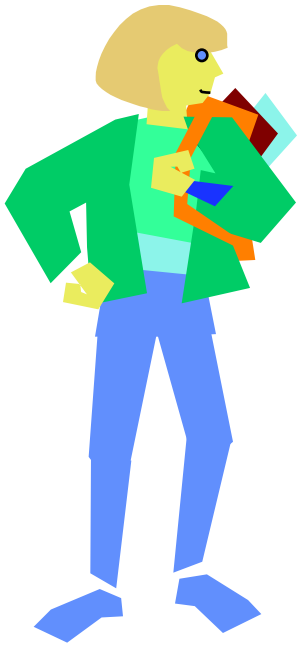
What is a “bug”?

- **Error:** a human action that produces an incorrect result
- **Fault:** a manifestation of an error in software
  - also known as a defect or bug
  - if executed, a fault may cause a failure
- **Failure:** deviation of the software from its expected delivery or service
  - (found defect)

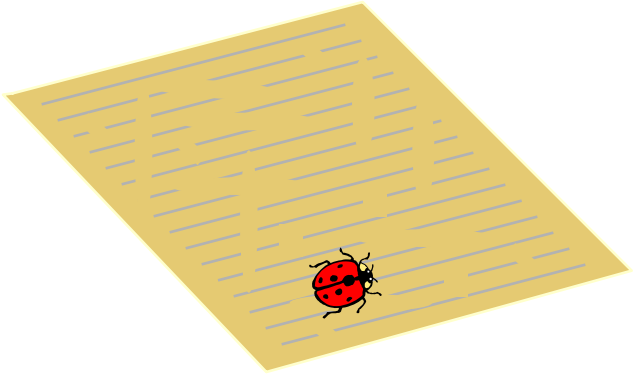
Failure is an event; fault is a state of the software, caused by an error

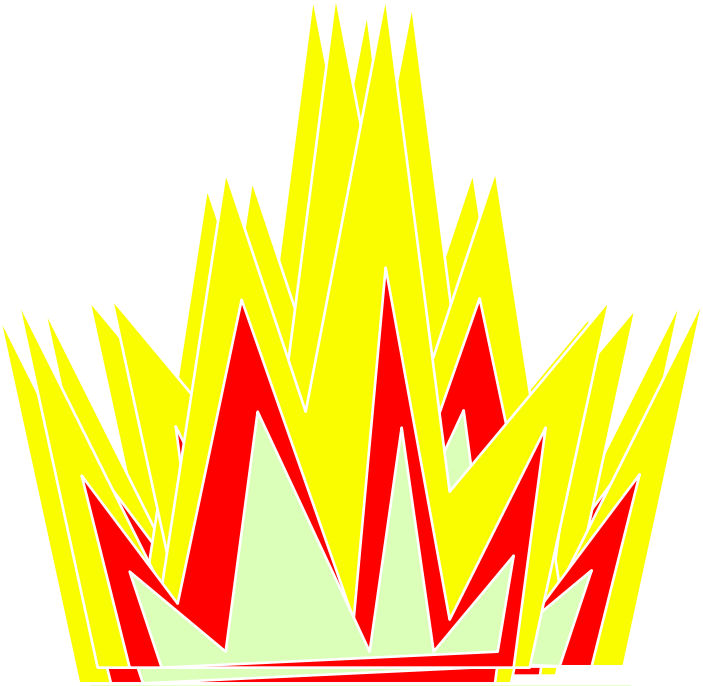
Error - Fault - Failure

A person makes an error ...



… that creates a fault in the software ...





… that can cause a failure in operation

Reliability versus faults

- Reliability: the probability that software will not cause the failure of the system for a specified time under specified conditions

  - Can a system be fault-free? (zero faults, right first time) ✗
  - Can a software system be reliable but still have faults? ✓
  - Is a “fault-free” software application always reliable? ✗

Reliability versus faults

- Reliability: the probability that software will not cause the failure of the system for a specified time under specified conditions

  - Can a system be fault-free? (zero faults, right first time) ✗
  - Can a software system be reliable but still have faults? ✓
  - Is a “fault-free” software application always reliable? ✗

Why do faults occur in software?

- Software is written by human beings
  - who know something, but not everything
  - who have skills, but aren't perfect
  - who do make mistakes (errors)
- Under increasing pressure to deliver to strict deadlines
  - no time to check but assumptions may be wrong
  - systems may be incomplete
- If you have ever written software ...

What do software faults cost?

- Huge sums
  - Ariane 5 ($7billion)
  - Mariner space probe to Venus ($250m)
  - American Airlines ($50m)
- Very little or nothing at all
  - minor inconvenience
  - no visible or physical detrimental impact
- Software is not “linear”:
  - small input may have very large effect

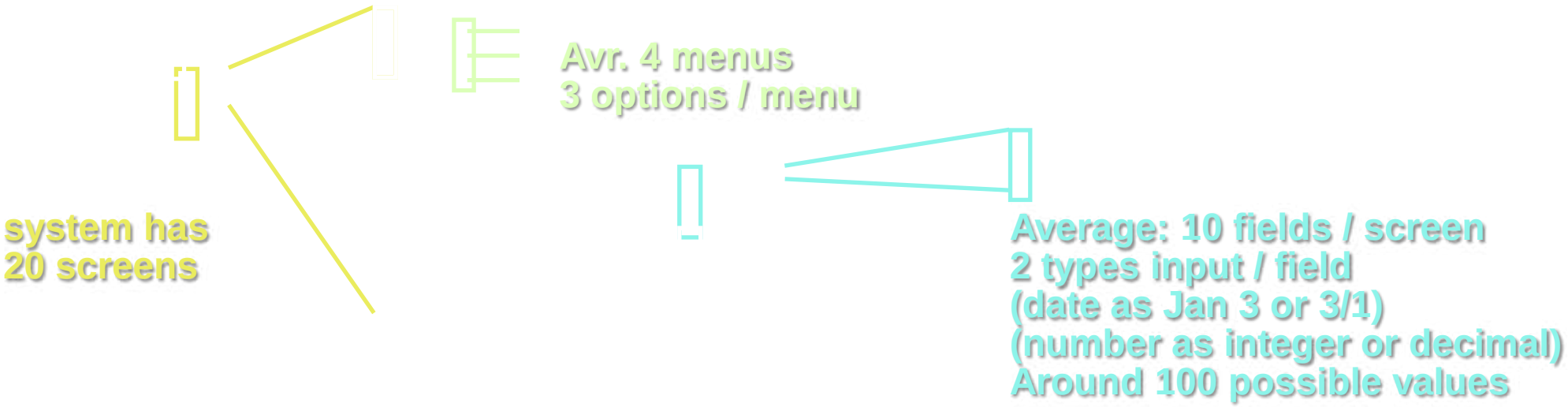
Safety-critical systems

- software faults can cause death or injury
  - radiation treatment kills patients (Therac-25)
  - train driver killed
  - aircraft crashes (Airbus & Korean Airlines)
  - bank system overdraft letters cause suicide

So why is testing necessary?

- because software is likely to have faults ✓
- to learn about the reliability of the software ✓
- to fill the time between delivery of the software and the release date ✗
- to prove that the software has no faults ✗
- because testing is included in the project plan ✗
- because failures can be very expensive ✓
- to avoid being sued by customers ✓
- to stay in business ✓

Why not just "test everything"?



Total for 'exhaustive' testing:

20 x 4 x 3 x 10 x 2 x 100 = 480,000 tests

If 1 second per test, 8000 mins, 133 hrs, **17.7 days**

(not counting finger trouble, faults or retest)

10 secs = 34 wks, 1 min = 4 yrs, 10 min = 40 yrs

Exhaustive testing?

- What is exhaustive testing?
  - when all the testers are exhausted ✗
  - when all the planned tests have been executed ✗
  - exercising all combinations of inputs and preconditions ✓

- How much time will exhaustive testing take?
  - infinite time ✗
  - not much time ✗
  - impractical amount of time ✓

How much testing is enough?

- it's never enough ✗
- when you have done what you planned ✗
- when your customer/user is happy ✗
- when you have proved that the system works correctly ✗
- when you are confident that the system works correctly ✗ ✓
- it depends on the risks for your system ✓

How much testing?

- It depends on RISK
  - risk of missing important faults
  - risk of incurring failure costs
  - risk of releasing untested or under-tested software
  - risk of losing credibility and market share
  - risk of missing a market window
  - risk of over-testing, ineffective testing

So little time, so much to test ..

- Test time will always be limited
- use RISK to determine:
  - what to test first
  - what to test most
  - how thoroughly to test each item
  - what not to test (this time)

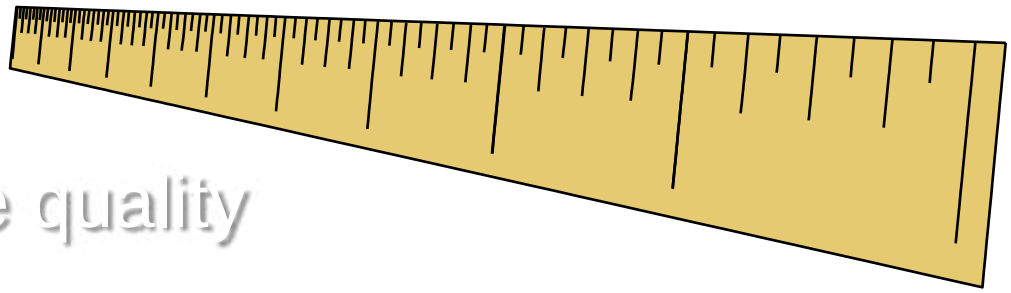
} i.e. where to place emphasis

- use RISK to
  - allocate the time available for testing by prioritising testing ...

Most important principle

Prioritise tests
so that,
whenever you stop testing,
you have done the best testing
in the time available.

Testing and Quality



- Testing measures software quality
- Testing can find faults; when they are removed, software quality (and possibly reliability) is improved
- What does testing test?
  - system function, correctness of operation
  - non-functional qualities: reliability, usability, maintainability, reusability, testability, etc.

Other factors that influence testing

- Contractual requirements
- Legal requirements
- Industry-specific requirements
  - e.g. pharmaceutical industry (FDA), compiler standard tests, safety-critical or safety-related such as railroad switching, air traffic control

It is difficult to determine
how much testing is enough
but it is not impossible

Testing Techniques

Verification Types

-
-
-

Verification "What to Look For?"

- Find all the missing information
  - Who
  - What
  - Where
  - When
  - Why
  - How

Peer Review

-

Walkthrough

-
-
-
-

Inspection

Formal meeting, characterized by individual preparation by all participants prior to the meeting.

- Objectives:
  - To obtain defects and collect data.
  - To communicate important work product information .
- Elements:
  - A planned, structured meeting requiring individual preparation by all participants.
  - A team of people, led by an impartial moderator who assure that rules are being followed and review is effective.
  - Presenter is “reader” other than the author.
  - Other participants are inspectors who review,
  - Recorder to record defects identified in work product

Checklists : the verification tool

-
-
-
-

Validation Strategies

Validation Strategies

White Box Testing

- Deals with the internal logic and structure of the code

- The tests are written based on the white box testing strategy incorporate coverage of the code written, branches, paths, statements and internal logic of the code etc.

- Normally done the developers

White Box testing

White Box Testing can be done by:

- Data Coverage

- Code Coverage

White Box Testing

Data Coverage

- Data flow is monitored or examined through out the program. E.g. watch window we use to monitor the values of the variables and expressions.

White Box Testing

Code Coverage

- It's a process of finding areas of a program not exercised by a set of test cases.

- Creating additional test cases to increase coverage

- Code coverage can be implemented using basic measure like, statement coverage, decision coverage, condition coverage and path coverage

Validation Strategies

- Does not need any knowledge of internal design or code

- Its totally based on the testing for the requirements and functionality of the work product/software application.

- Tester is needed to be thorough with the requirement specifications of the system and as a user, should know how the system should behave in response to the particular action.

Black Box testing Methods

Commonly used Black Box methods :

Equivalence Partitioning

-

-

Equivalence Partitioning

If we expect the same result from two tests, you consider them equivalent. A group of tests from an equivalence class if,

Equivalence Partitioning

For example, a program which edits credit limits within a given range ($10,000-$15,000) would have three equivalence classes:

- Less than $10,000 (invalid)
- Between $10,000 and $15,000 (valid)
- Greater than $15,000 (invalid)

Equivalence Partitioning

- Partitioning system inputs and outputs into 'equivalence sets'
  - If input is a 5-digit integer between 10,000 and 99,999 equivalence partitions are <10,000, 10,000-99,999 and >99,999
- The aim is to minimize the number of test cases required to cover these input conditions

Equivalence Partitioning

- If an input condition specifies a range, one valid and two invalid equivalence classes are defined.

- If an input condition requires a specific value, then one valid and two invalid equivalence classes are defined.

- If an input condition is Boolean, then one valid and one invalid equivalence class are defined.

Equivalence Partitioning Summary

-
-
-
-
-
-
-
-
-

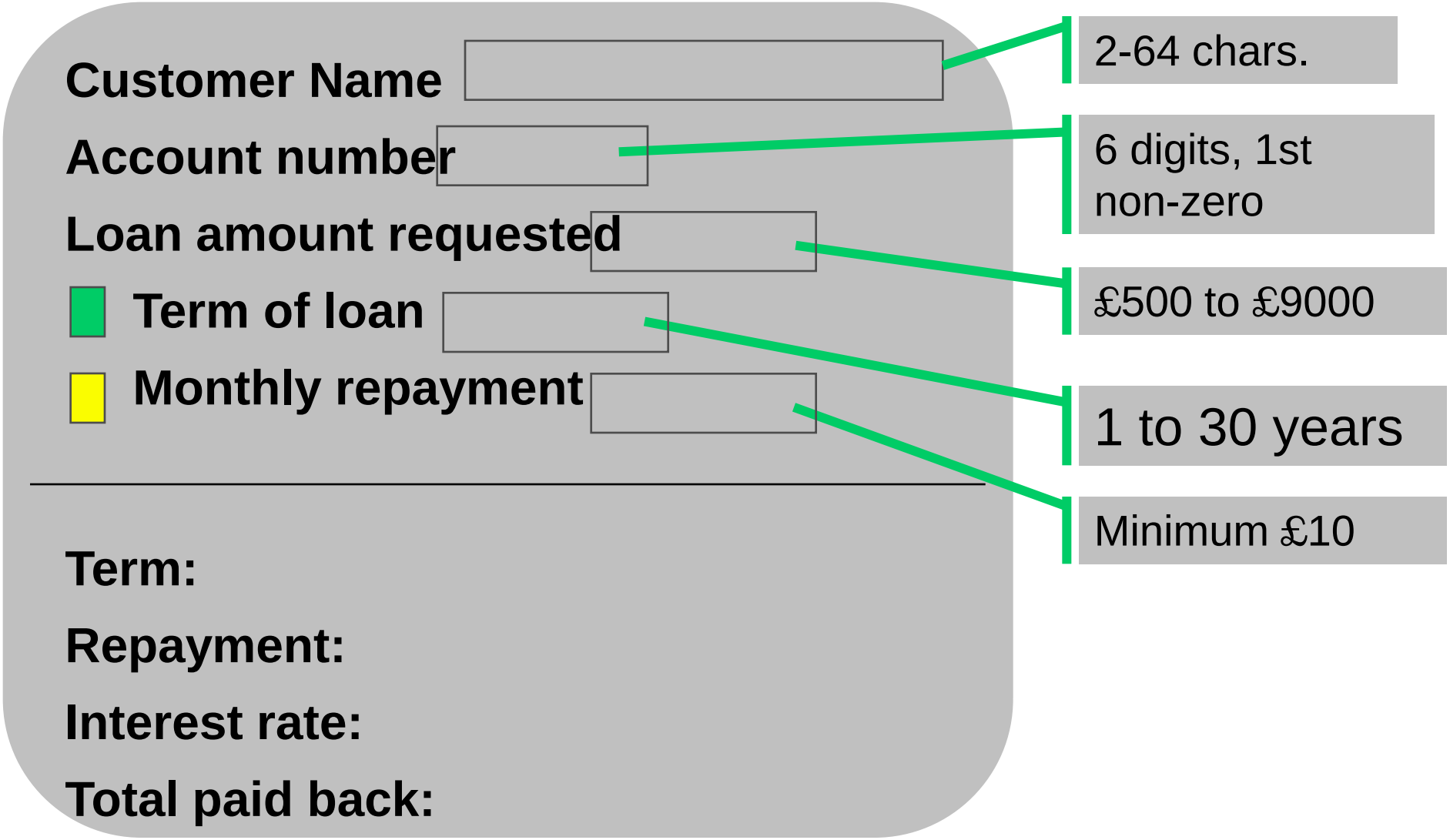
Boundary value analysis

Boundary value analysis

-
-
  - Low boundary plus or minus one ($9,999 and $10,001)
  - On the boundary ($10,000 and $15,000)
  - Upper boundary plus or minus one ($14,999 and $15,001)

Boundary value analysis

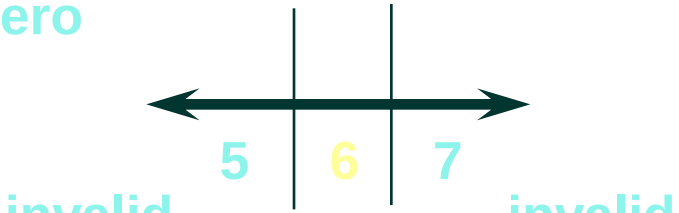
Example: Loan application



Account number

valid: non-zero

invalid: zero



Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
Account number	6 digits 1st non-zero	< 6 digits > 6 digits 1st digit = 0 non-digit	100000 999999	5 digits 7 digits 0 digits

Error Guessing

- Based on the theory that test cases can be developed based upon the intuition and experience of the Test Engineer

- For example, in an example where one of the inputs is the date, a test engineer might try February 29,2000 or 9/9/99

Various Types of Testing

Validation Activities

Validation is done at two levels

- Low Level
  - Unit testing
  - Integration Testing

White Box

- High Level
  - Function Testing
  - System Testing
  - Acceptance Testing

Black Box

Unit Testing

- Searches for defect and verifies the functionality of software, depending upon the context of the development

- It includes testing of functional and non-functional characteristics

- It occurs with access to code being tested and with the support of development environment

- Defects are fixed as soon as they are found with out formally recording incident

- If test cases are prepared and automated before coding, it is termed as test-first approach or test-driven development.

Integration Testing

-
-
-
-

Functional Testing

System Testing

System Testing Types

- Usability testing
- Performance Testing
- Load Testing
- Stress Testing
- Security Testing
- Configuration Testing
- Compatibility Testing
- Installation Testing
- Back up & Recovery Testing
- Availability Testing
- Volume Testing

Usability Testing

Performance Testing

- Performance testing is testing to ensure that the application response in the limit set by the user.

Stress Testing

- Subject the system to extreme pressure in a short span.
- E.g Simultaneous log-on of 500 users
  - Saturation load of transactions

Configuration Testing

-

Compatibility Testing

- The purpose of compatibility testing is to evaluate how well software performs in a particular hardware, software, operating system, browser or network environment.

Acceptance Testing

- Acceptance testing may assess the system readiness for deployment and use
- The goal is to establish confidence in the system, parts of system or non-functional characteristics of the system
- Following are types of Acceptance Testing:
  - User Acceptance Testing
  - Operational Testing
  - Contract and Regulation Acceptance Testing
  - Alpha and Beta Testing

Objectives of Different Types of Testing

- *to cause as many failures as possible.*
- *to confirm that system work as expected.*
- *to make sure that no new errors have been introduced.*
- *to access system characteristics such as reliability and availability.*

Other Testing Types

Mutation testing

Progressive/Regressive Testing

-

Regression Testing

- Regression testing is not another testing activity
- It is a re-execution of some or all of the tests developed for a specific testing activity for each build of the application
- Verify that changes or fixes have not introduced new problems
- It may be performed for each activity (e.g. unit test, function test, system test etc)

Regression Testing

- evolve over time
- are run often
- may become rather large

Retesting

Why retest?

- Because any software product that is actively used and supported must be changed from time to time, and every new version of a product should be retested

Localization Testing

Internationalization Testing

- The process of designing an application so that it can be adapted to various languages and regions without engineering changes.

Test Types : The Target of Testing

- Testing of functions (functional testing)

  - It is the testing of "what" the system does
  - Functional testing considers external behavior of the system
  - Functional testing may be performed at all test levels

- Testing of software product characteristics (non-functional testing)

  - It is the testing of "How" the system works
  - Nonfunctional testing describes the test required to measure characteristics of systems and s/w that can be quantified on varying scale
  - Non-functional testing may be performed at all levels

Test Types : The Target of Testing

- Structural testing is used in order to help measure the thoroughness of testing through assessment of coverage of a type of structure
- Structural testing may be performed at all levels.

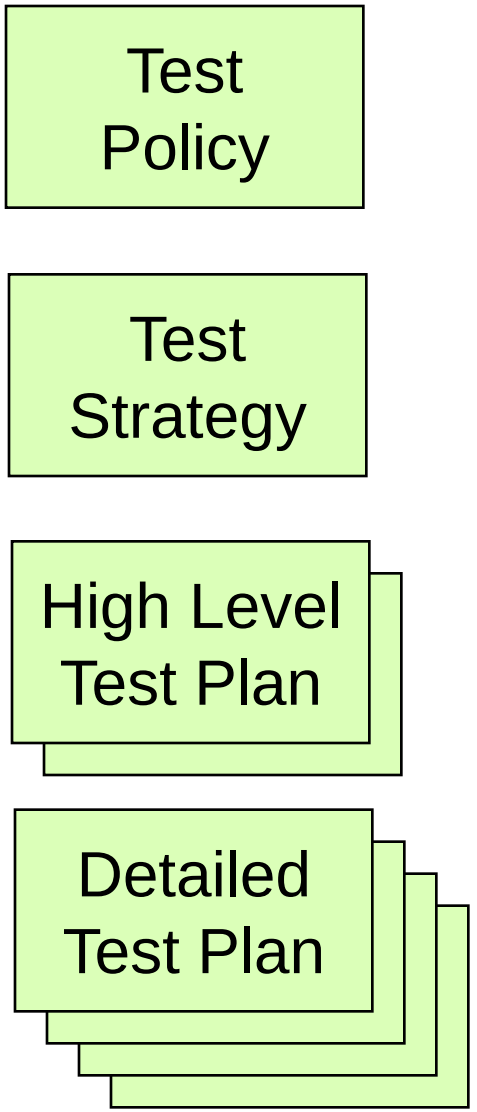
- When a defect is detected and fixed then the software should be retested to confirm that the original defects has been successfully removed. This is called Confirmation testing
- Regression Testing is the repeated testing of an already tested program, after modification, to discover any defects as a result of changes.
- Regression Testing may be performed at all levels.

Test Planning

- It is the process of defining a testing project such that it can be properly measured and controlled

- It includes test designing, test strategy, test requirements and testing resources

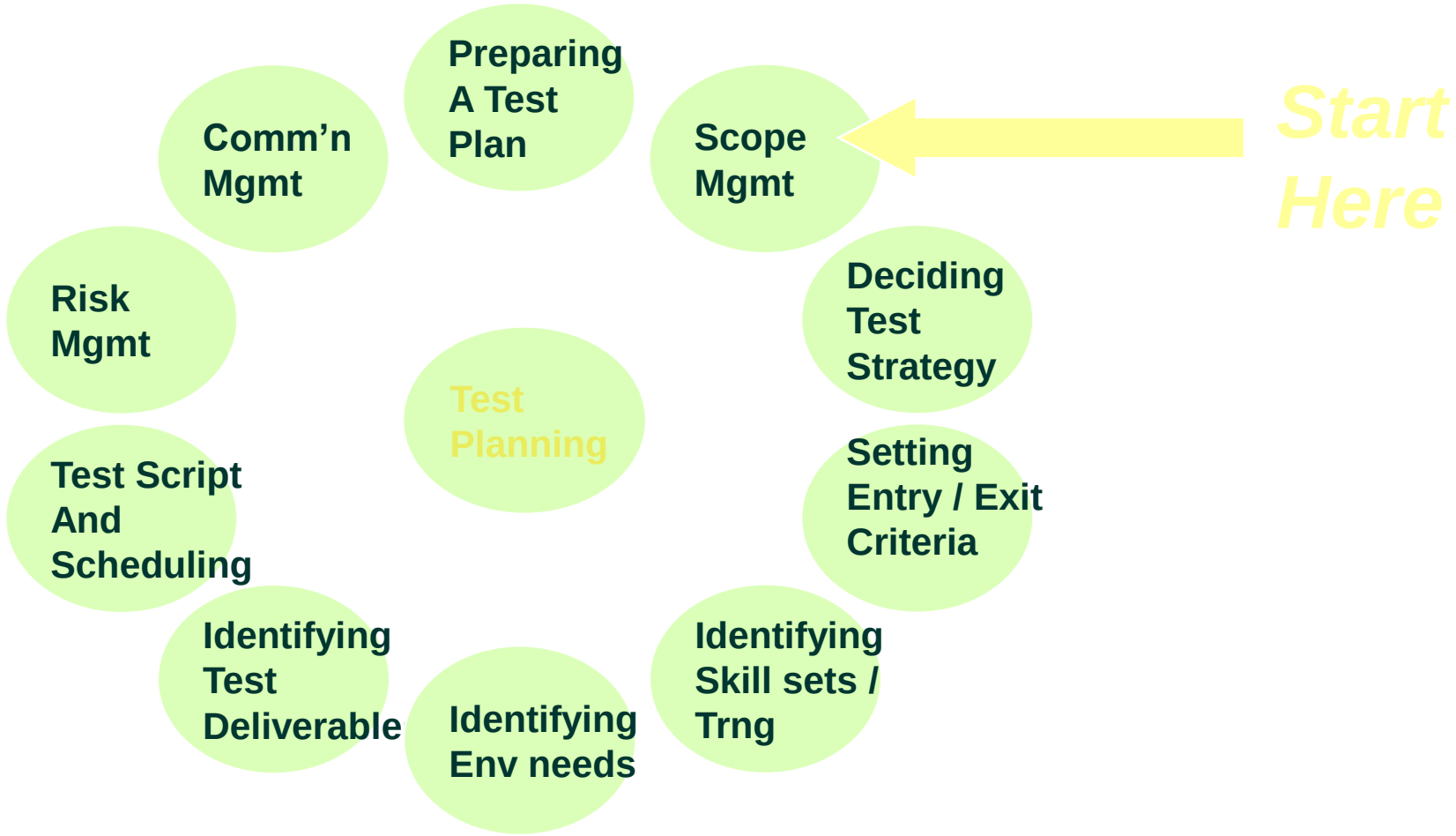
Test Planning - different levels



(one for each project)

(one for each stage within a project, e.g. Component, System, etc.)

Parts of Test Planning



Test Planning

- Test Planning is a continuous activity and is performed in all the life cycle processes and activities

- Test Planning activities includes:
  - Defining the overall approach
  - Integrating and coordinating the testing activities into software life cycle activities
  - Assigning resources for different tasks defined
  - Defining the amount, level of detail, structure and templates for test documentation
  - Selecting metrics for monitoring and controlling test preparation
  - Making decisions about what to test, what roles will perform the test activities, when and how test activities should be done, how the test results will be evaluated and when to stop the testing

Test Planning

- Exit Criteria – Defines when to stop testing

- Exit criteria may consist of
  - Thoroughness measures, such as coverage of code, functionality or risk
  - Estimates of defect density or reliability measures
  - Cost
  - Residual risk
  - Schedules such as those based on time to market

Risk Objectives

- **Suppliers Issues**
  - Failure of a third party
  - Contractual Issues
- **Organizational Factors**
  - Skill and staff shortage
  - Personal and training issues
  - Potential issues, such as problem with testers communication, failure to follow up the information found in Testing
  - Improper attitude towards testing
- **Technical Issues**
  - Problem in defining the right requirement
  - The extent that requirements can be met given existing constraints
  - Quality of design, code and tests

Risk Objectives

- Product/Project Risks Objective
  - Error prone software delivered
  - Potential that the software/hardware could cause harm to company/individual
  - Poor software characteristics
  - Software that does not perform its intended functions

- A risk based approach to testing provides proactive opportunities to reduce the levels of product risks, starting in the initial stages of project

Test Designing and Execution

Test Design Specification

Design(detailed level)

specification	execution	recording	check completion

Identify conditions

Design test cases

Build tests

A good test case

- Finds faults
- Represents others
- Easy to maintain
- Cheap to use

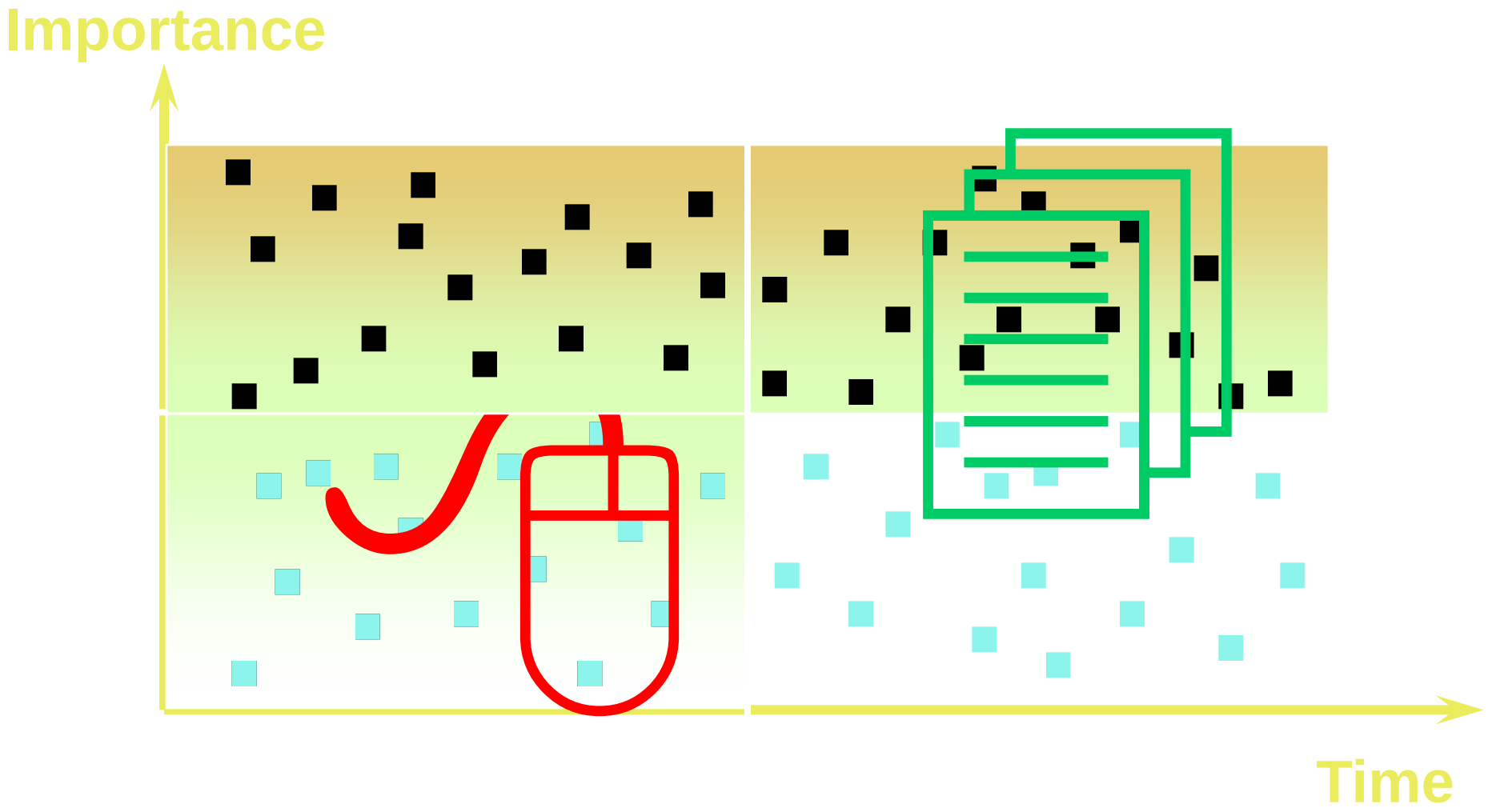
Test specification

- test specification can be broken down into three distinct tasks:
  1. identify: determine ‘what’ is to be tested (identify test conditions) and prioritise
  2. design: determine ‘how’ the ‘what’ is to be tested (i.e. design test cases)
  3. build: implement the tests (data, scripts, etc.)

Task 1: identify conditions

- list the conditions that we would like to test:
  - use the test design techniques specified in the test plan
  - there may be many conditions for each system function or attribute
  - e.g.
    - "life assurance for a winter sportsman"
    - "number items ordered > 99"
    - "date = 29-Feb-2004"
- prioritise the test conditions
  - must ensure most important conditions are covered

Selecting test conditions

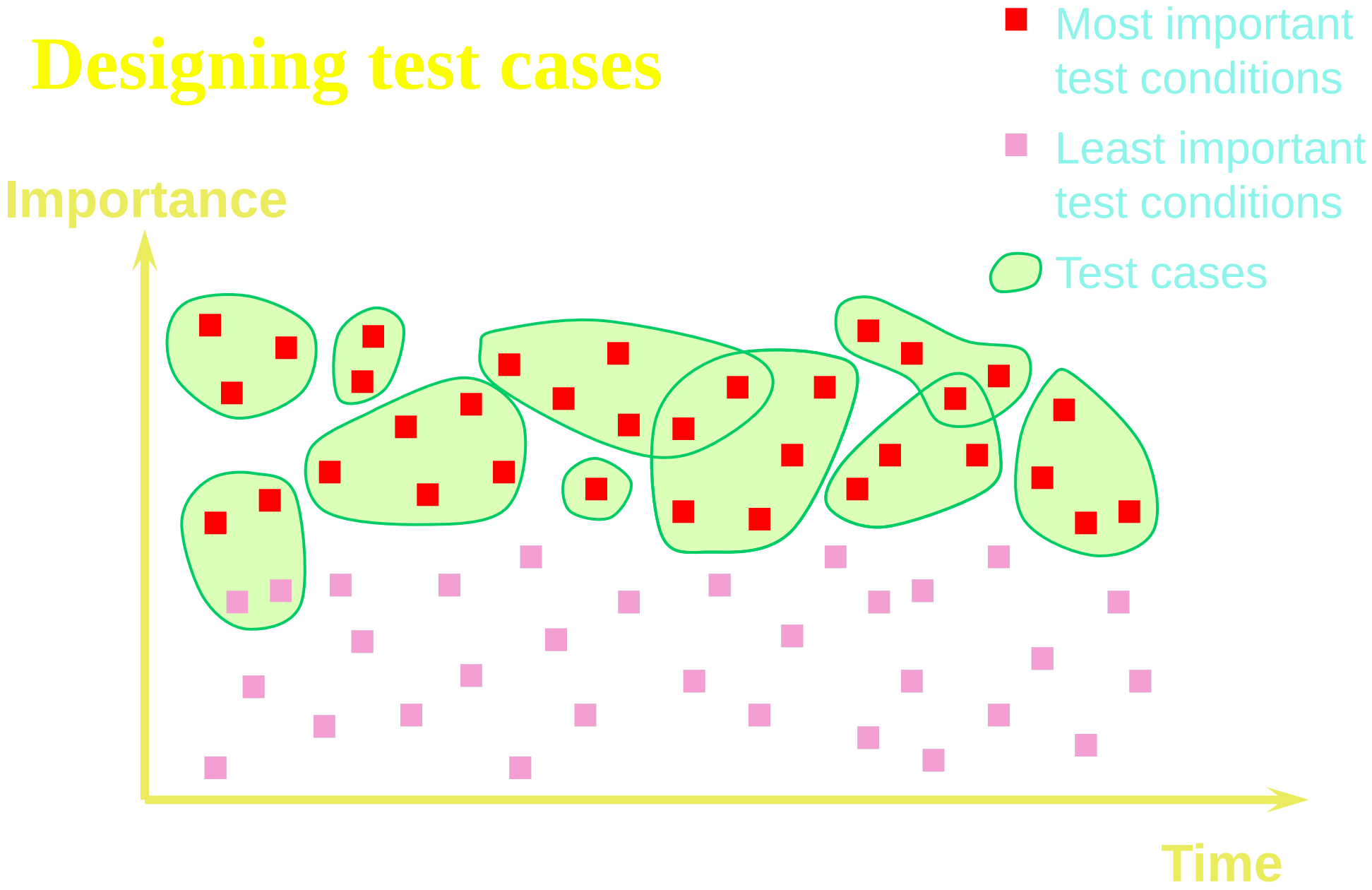


Task 2: design test cases

(determine 'how' the 'what' is to be tested)

- design test input and test data
  - each test exercises one or more test conditions
- determine expected results
  - predict the outcome of each test case, what is output, what is changed and what is not changed
- design sets of tests
  - different test sets for different objectives such as regression, building confidence, and finding faults

Designing test cases



Task 3: build test cases

(implement the test cases)

- prepare test scripts
  - less system knowledge tester has the more detailed the scripts will have to be
  - scripts for tools have to specify every detail
- prepare test data
  - data that must exist in files and databases at the start of the tests
- prepare expected results
  - should be defined before the test is executed

Test execution

Planning (detailed level)

specification	execution	recording	check completion

Execution

- Execute prescribed test cases
  - most important ones first
  - would not execute all test cases if
    - testing only fault fixes
    - too many faults found by early test cases
    - time pressure
  - can be performed manually or automated

Test Recording

Planning (detailed level)

specification	execution	recording	check completion

Test recording 1

- The test record contains:
  - identities and versions (unambiguously) of
    - software under test
    - test specifications
- Follow the plan
  - mark off progress on test script
  - document actual outcomes from the test
  - capture any other ideas you have for new test cases
  - note that these records are used to establish that all test activities have been carried out as specified

Test recording 2

- Compare actual outcome with expected outcome. Log discrepancies accordingly:
  - software fault
  - test fault (e.g. expected results wrong)
  - environment or version fault
  - test run incorrectly
- Log coverage levels achieved (for measures specified as test completion criteria)
- After the fault has been fixed, repeat the required test activities (execute, design, plan)

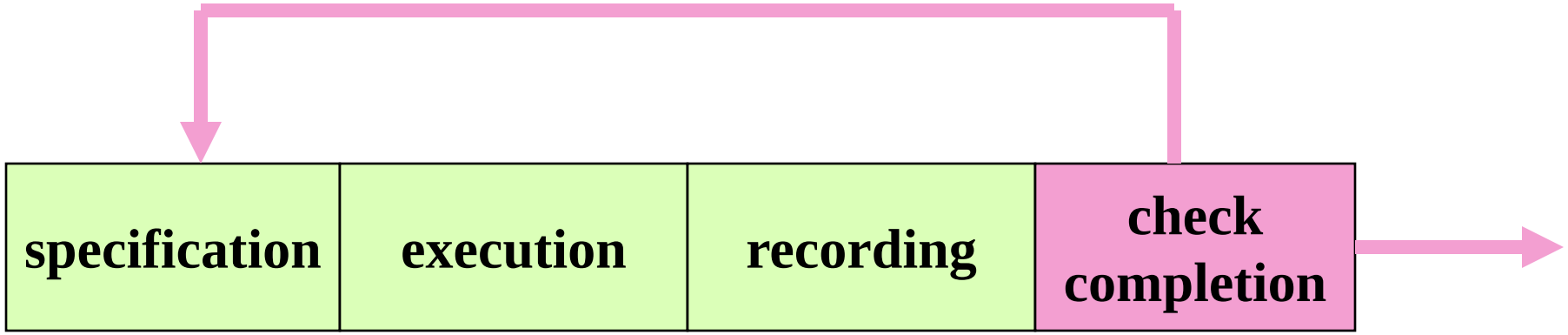
Check test completion

Planning (detailed level)

specification	execution	recording	check completion

Check test completion

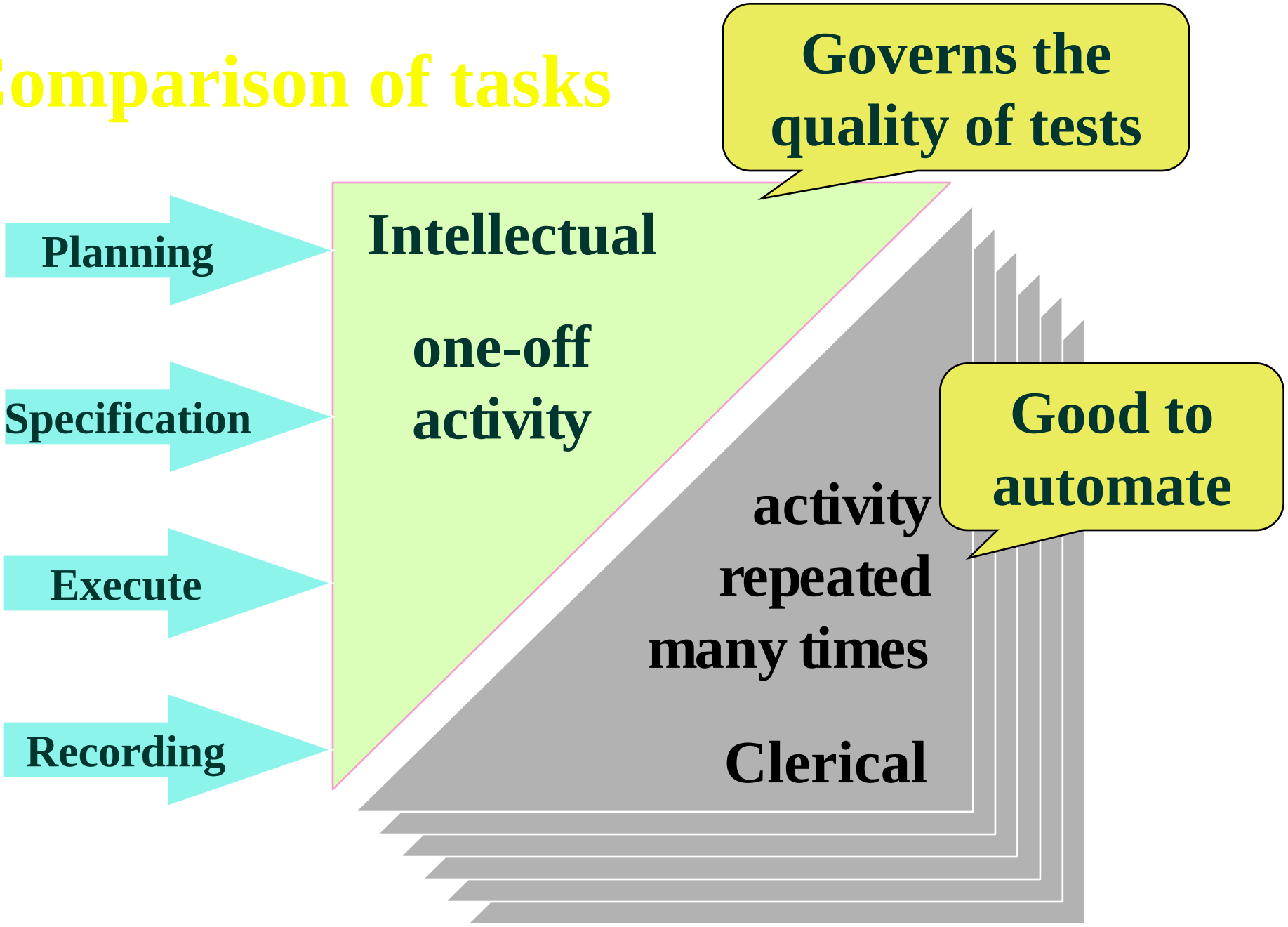
-
-



Test completion criteria

- Completion or exit criteria apply to all levels of testing - to determine when to stop
  - coverage, using a measurement technique, e.g.
    - branch coverage for unit testing
    - user requirements
    - most frequently used transactions
  - faults found (e.g. versus expected)
  - cost or time

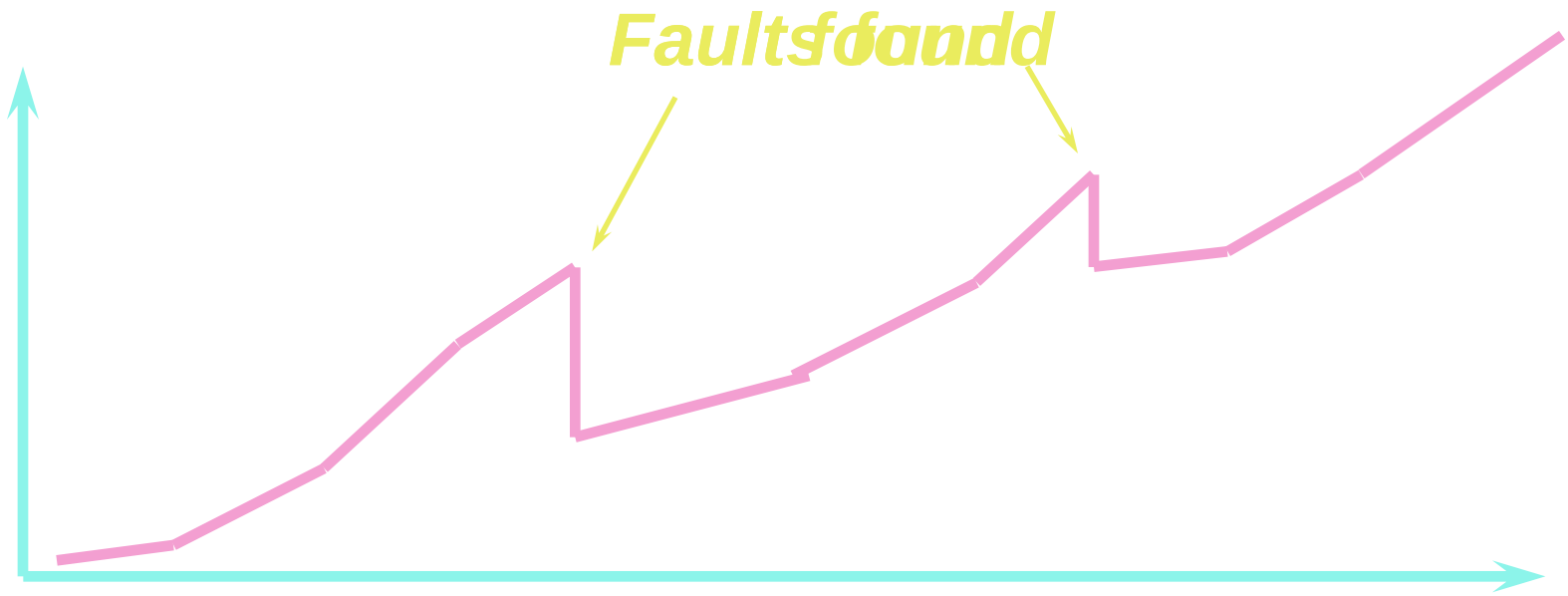
Comparison of tasks



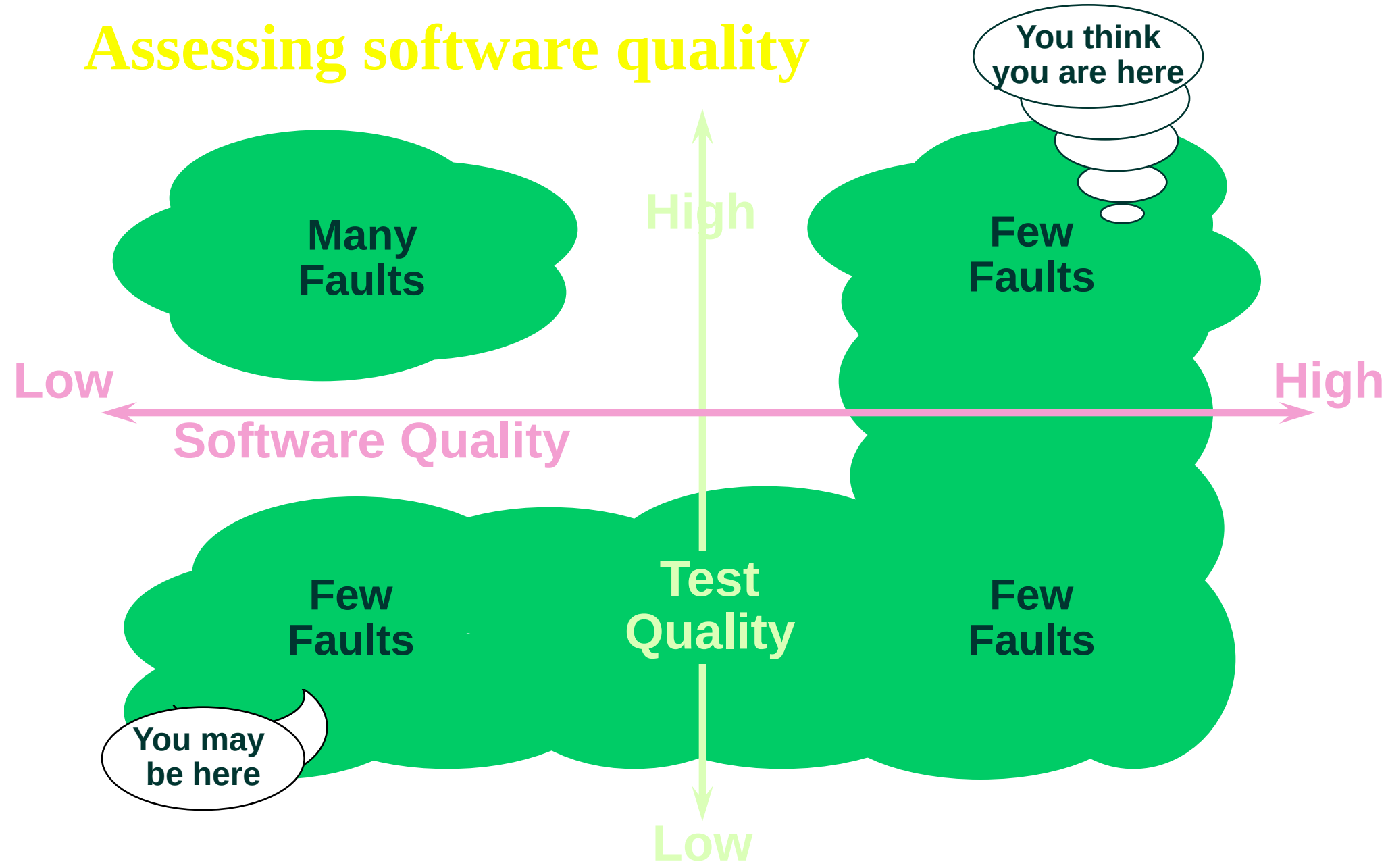
Psychology of testing

Why test?

Confidence



No faults found = confidence?


Assessing software quality
You think you are here
Many Faults
High
Few Faults
Low
High
Software Quality
Few Faults
Test Quality
Few Faults
You may be here
Low

A traditional testing approach

-
  - does what it should
  - doesn't do what it shouldn't

Goal:	**show working**
Success:	**system works**

Result: faults left in

A better testing approach

-
  - does what it shouldn't
  - doesn't do what it should

Goal:	**find faults**
Success:	**system fails**

Result: fewer faults left in

The testing paradox

The best way to build confidence

is to try to destroy it

Who wants to be a tester?

-
- (“your baby is ugly”)
-
-
-

Tester’s have the right to:

Testers have responsibility to:

Independence

- Test your own work?
  - find 30% - 50% of your own faults
  - same assumptions and thought processes
  - see what you meant or want to see, not what is there
  - emotional attachment
    - don't want to find faults
    - actively want NOT to find faults

Levels of independence

-
-
-
  (e.g. test team)
-
  (e.g. agency)
-
  (low quality tests?)

Software Defect Life Cycle

ILLUSTRATION BY SEGUE TECHNOLOGIES

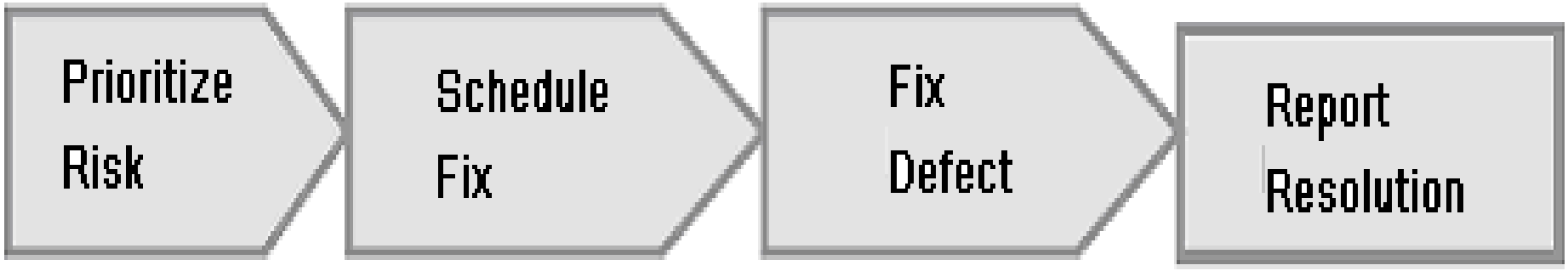
Defect Management

What is definition of defect?

Defect Discovery

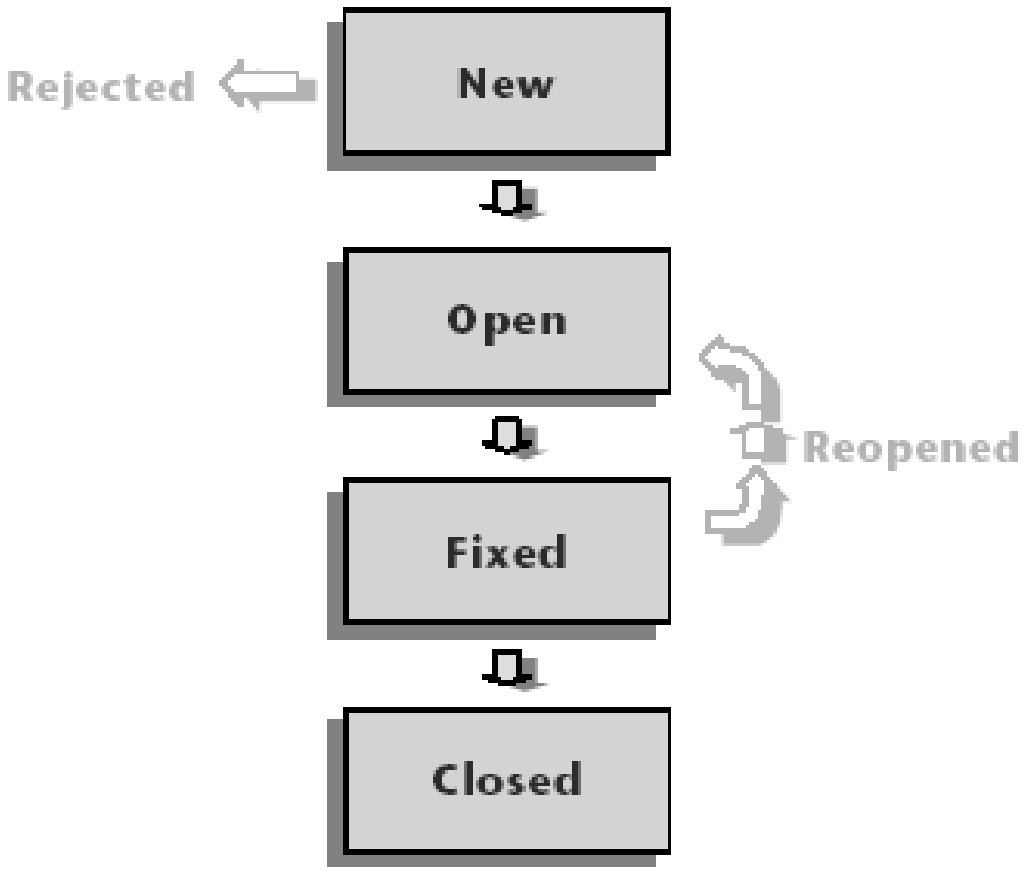
Defect Discovery Process

Defect Resolution



Defect Resolution Process

Defect Life Cycle



Defect Life Cycle

When a tester reports a Defect, it is tracked through the following stages: ***New***, ***Open***, ***Fixed***, **and** ***Closed***. A defect may also be ***Rejected***, or ***Reopened*** after it is fixed. A defect may be **Deferred** for a look at a later point of time.

By default a defect is assigned the status ***New***.

A quality assurance or project manager reviews the defect, and determines whether or not to consider the defect for repair. If the defect is refused, it is assigned the status ***Rejected***.

If the defect is accepted, the quality assurance or project manager determines a repair priority, changes its status to ***Open***, and assigns it to a member of the development team.

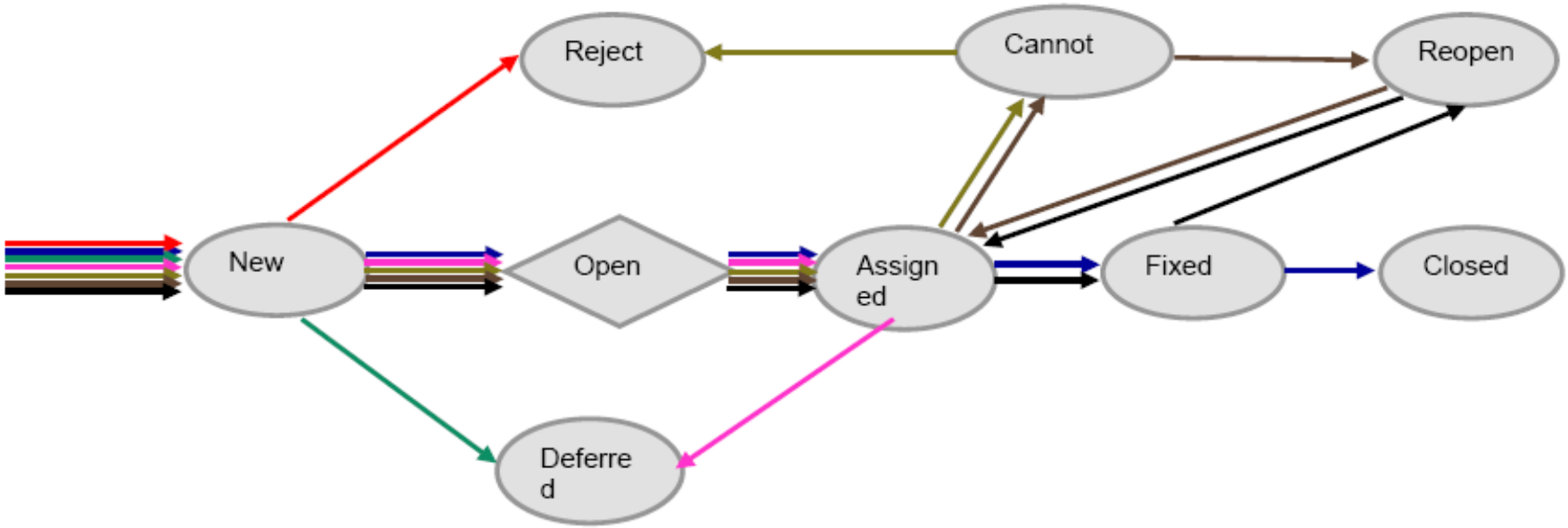
Defect Life Cycle

A developer repairs the defect and assigns it the status ***Fixed*.**

Tester retests the application, making sure that the defect does not recur. If the defect recurs, the quality assurance or project manager assigns it the status ***Reopened*.**

If the defect is actually repaired, it is assigned the status ***Closed*.**

Defect Life Cycle Paths



Defect Life Cycle Paths

1. New - Open - Assigned - Fixed - Closed

2. New - Reject

3. New - Deferred

4. New - Open - Assigned - Deferred

5. New - Open - Assigned – Cannot Reproduce - Reject

6. New - Open – Assigned – Cannot Reproduce – reopened - Assigned

7. New - Open – Assigned - Fixed – Reopen - Assigned

Defect Classification

4.3.1. Severity Level of Defects

Severity Types	Description	Examples
1-Show Stopper	❖ Defect that causes total failure of the software system or subsystem or unrecoverable data loss or severe impact to data integrity ❖ There is no workaround ❖ Testing can not continue without rectifying this defect	Defects that cause the system to crash, corrupt data files, or completely disrupt services
2-High	❖ Defect that results in severely impaired functionality ❖ A work around may exist but its use is unsatisfactory and may cause excessive delay in completing the functionality ❖ Product can not be released with such a defect	Error in Account Opening and work around is to create a manual feed and pump new accounts into the database

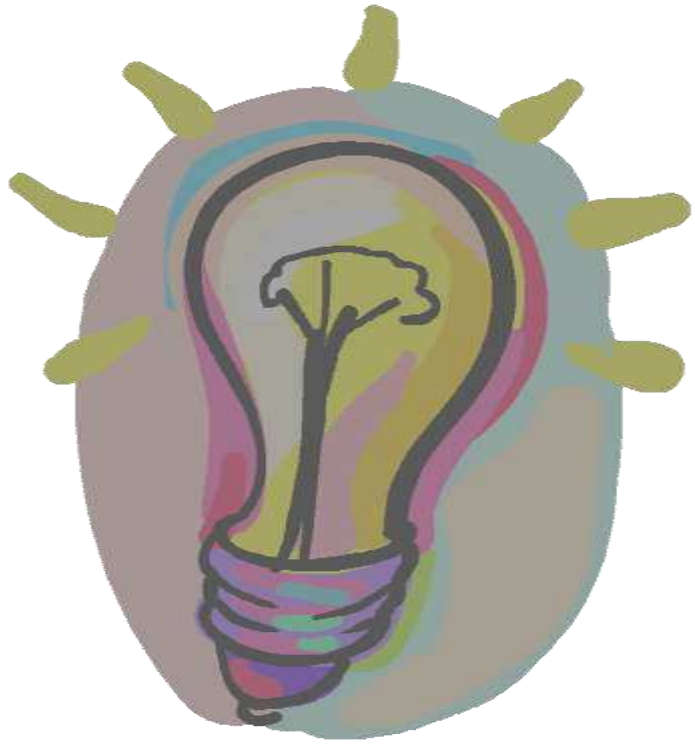
Defect Classification

4.3.1. Severity Level of Defects

Severity Types	Description	Examples
3-Medium	❖ Defect that causes failure of non-critical aspects of the system, or produce incorrect, incomplete or inconsistent results ❖ There is a reasonably satisfactory work-around ❖ The product may be released with this defect, but the existence of the defect may cause delay in work or end-user dissatisfaction	Search option is not working in huge "Products Lists" screen
4-Low	❖ Defect of minor significance ❖ A work-around exists or, if not, the impairment is slight ❖ The product could be released with the defect and most customers would be unaware of the defect's existence or only slightly dissatisfied	A formatting error in printed output

How many testers do we need to change a light bulb?

-
-



What Do You Do When You Find a defect?

-
-

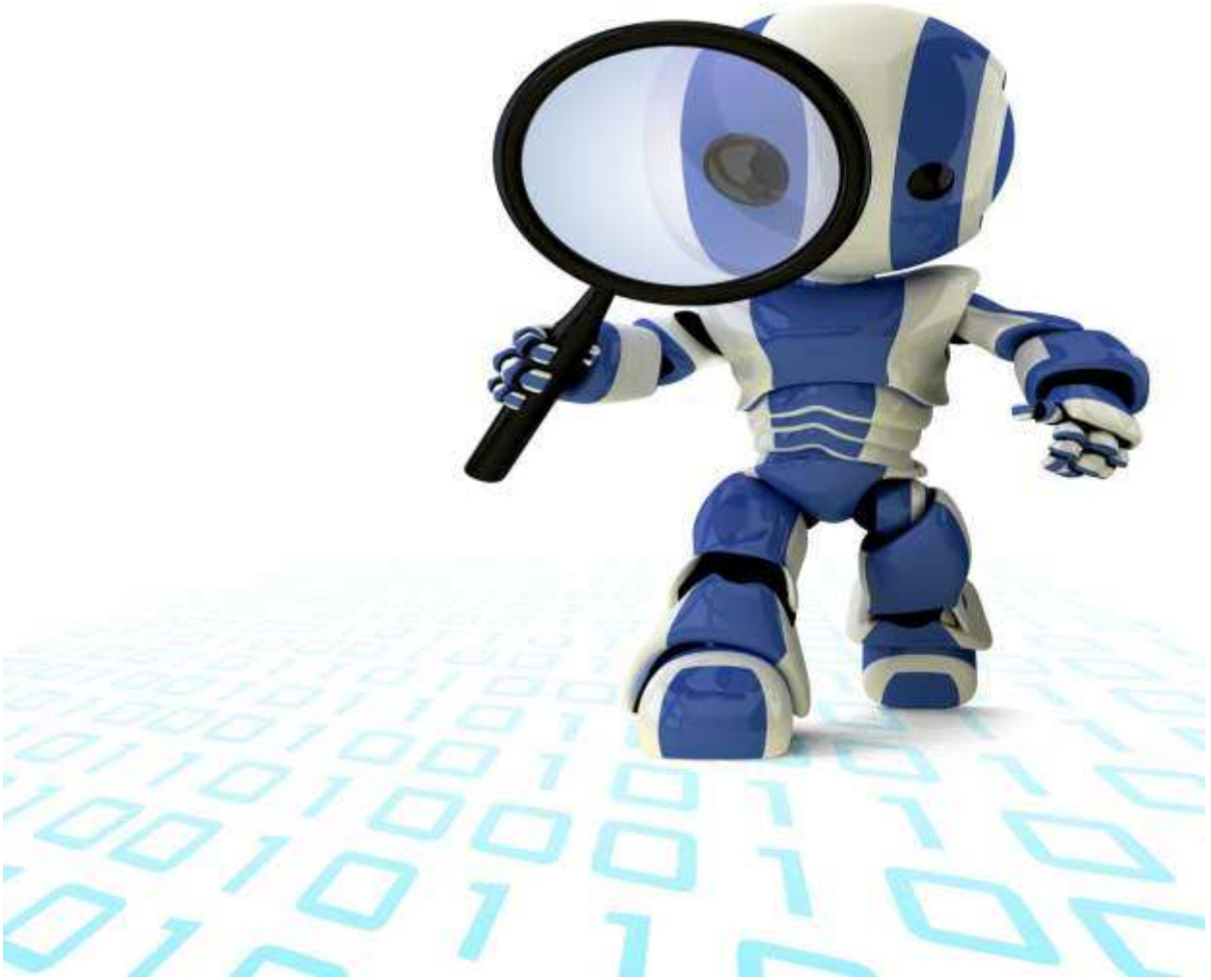
Some typical defect report fields

-
-
-
-
-
-
-
-
-
-
-
-

Who reads the defect reports?

-
-
-
-
-
-
-

Software Test Automation



Principles of Test Automation

1: Choose carefully what to automate

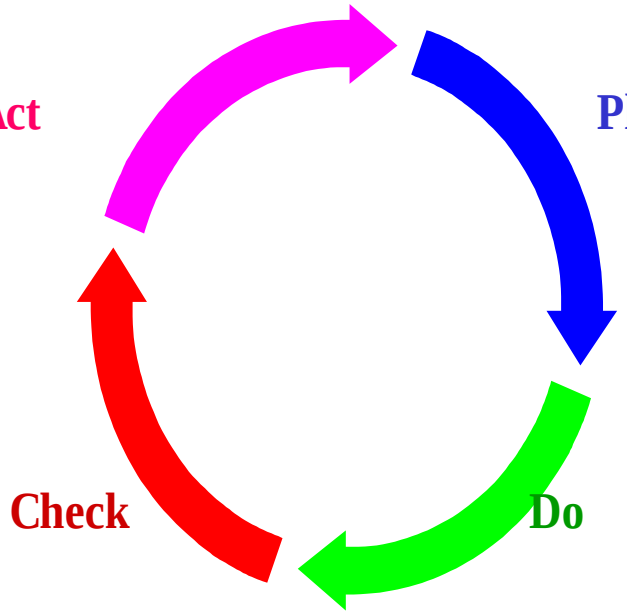
-
-
-
-
-
-
-

Principles of Test Automation

2: Ensure Automation Covers Full Circle

- Corrective Action Implementation
- Automatic Rollover to next runs
- Incorporation into Regression

Act



Plan

- Test Planning
- Automation Planning

Check

- Automatic Analysis
- Fish Bone Diagrams
- Problem Identification

Do

- Test Capture
- Test Execution
- Results Comparison

Principles of Test Automation

3: Choose Proper Automation Tool

-
-
-
-
-
-
-
-
-
-
-
-

Principles of Test Automation

4: Plan for Infrastructure

-
-
-

Principles of Test Automation

5: Account for Gestation Period

-
-
-
-

Principles of Test Automation

6: Run a Trial & Calibrate the Tool

-
-
-

Process of Test Automation

The V - Model of Software Development

Common Experiences in Test Automation

- There are plenty of tools available and rarely does one tool meet **all** the requirements
- The test tools are expensive (both in upfront costs and running costs)
- Test tools also require good amount of training and only few vendors available for training
- Training may not always keep pace with new versions of the tools
- Test tools expect the users to learn new language/scripts and may not use standard languages/scripts
- Deploying a test tool requires equal amount of effort as deploying a new product in a company – never underestimate the effort and pain involved!

Common Experiences in Test Automation

- Migrating from one test tool to another may be difficult and requires good amount of effort
- Test tools are one generation behind and may not provide backward / forward compatibility (eg. JAVA SDK support)
- Good number of test tools requires their libraries linked with product binaries – Causes portions of the testing to be repeated after those libraries are removed (eg. Performance)
- Test tools are not 100% cross platform – They are supported only on some platforms and the sources generated from these tools may not be compatible on other
- Developing sharewares/public domain test tools may not get same amount of participation/involvement/support as of standards/products (eg. As against Linux)

Common Experiences in Test Automation

The experiences

- Test tools may not go through same amount of evaluation for new requirements (eg Year 2000, 508)
- The test tools increases the system requirements and requires the H/W and S/W to be upgraded at compile/run-time
- The test tools are capable of testing only the product, not the impact because of the product/test tool to the system or network
- Good number of test tools can’t differentiate between a product failure and the test suite failure – Causing increased analysis time and manual testing

Common Experiences in Test Automation

The experiences

- The test tools may not provide good degree of trouble shooting / debug/error messages to help in analysis – Resulting in increased “printf”/log messages in the test suite
- The test tools determine the results based on messages and screen co-ordinates at run-time – Intelligence needed to proactively find out the changes

Common Pitfalls in Test Automation

- Automation shouldn’t be considered as stop-gap arrangement to engage test engineers (when no test execution, do automation!). Test Automation, like any other project, should start with the end in mind
- A separate team in the organization looking at automation requirements, tool evaluation and developing generic test suites would add more value (may not always apply to testing services organization)
- Automation doesn’t stop with automating the test cases alone. The test suite needs to be linked with other tools for increased effectiveness (e.g., Test case database, Defect filing, auto mails, preparing automatic reports, etc)
- Automation doesn’t stop with recording & playing back the user commands; Automated tool should be intelligent enough to say what was expected, why a test case failed and give manual steps to reproduce the problem


PREGUNTAS