Chapter 1

Software Testing

ISTQB Foundation Exam Practice

FUNDAMENTALS OF TESTING

1 Fundamentals	2 Lifecycle	3 Static testing
4 Dynamic test techniques	5 Management	6 Tools

CONTENTS

WHAT IS A "BUG"?

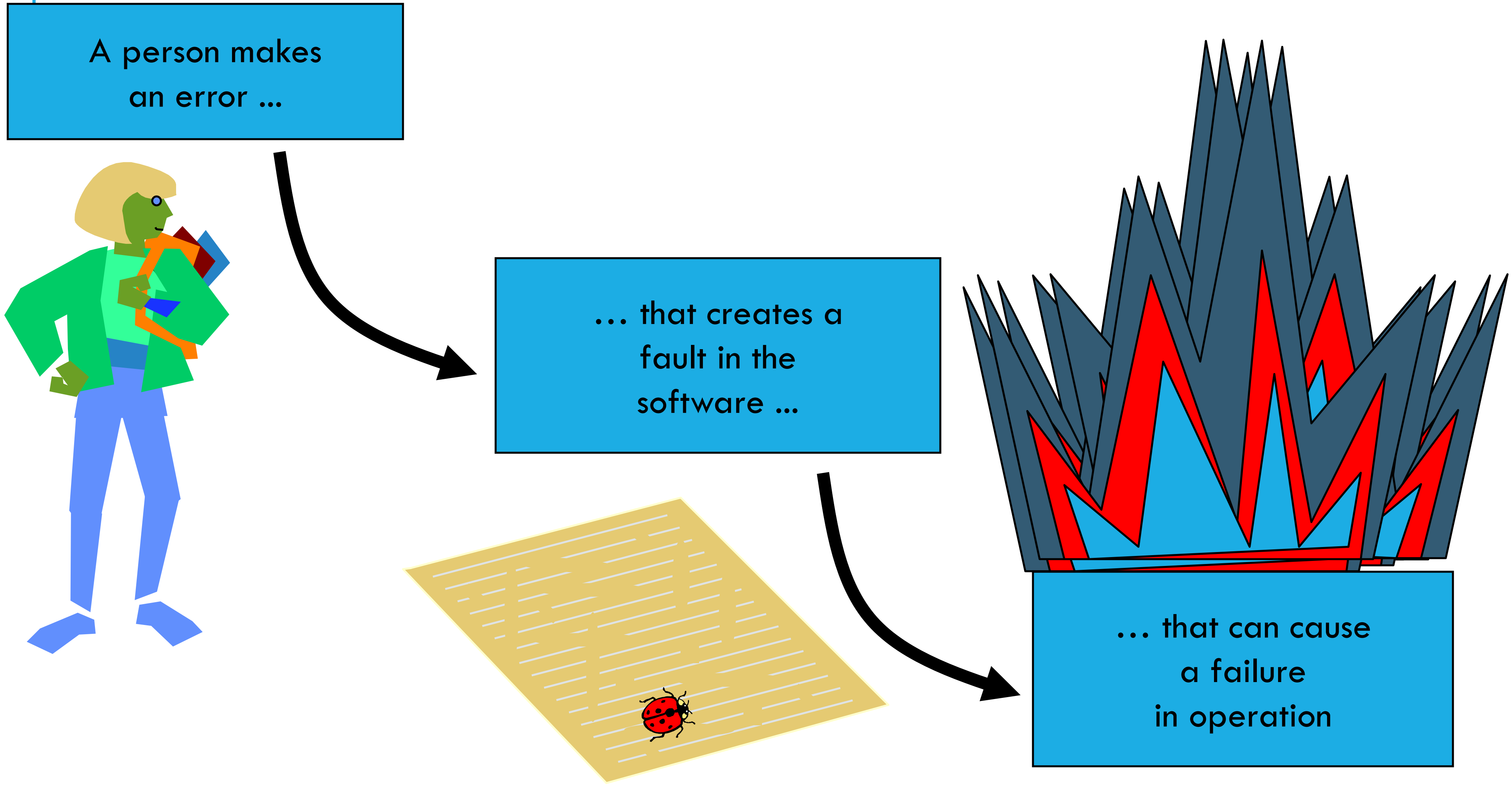
Error: a human action that produces an incorrect result

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? ✗

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

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

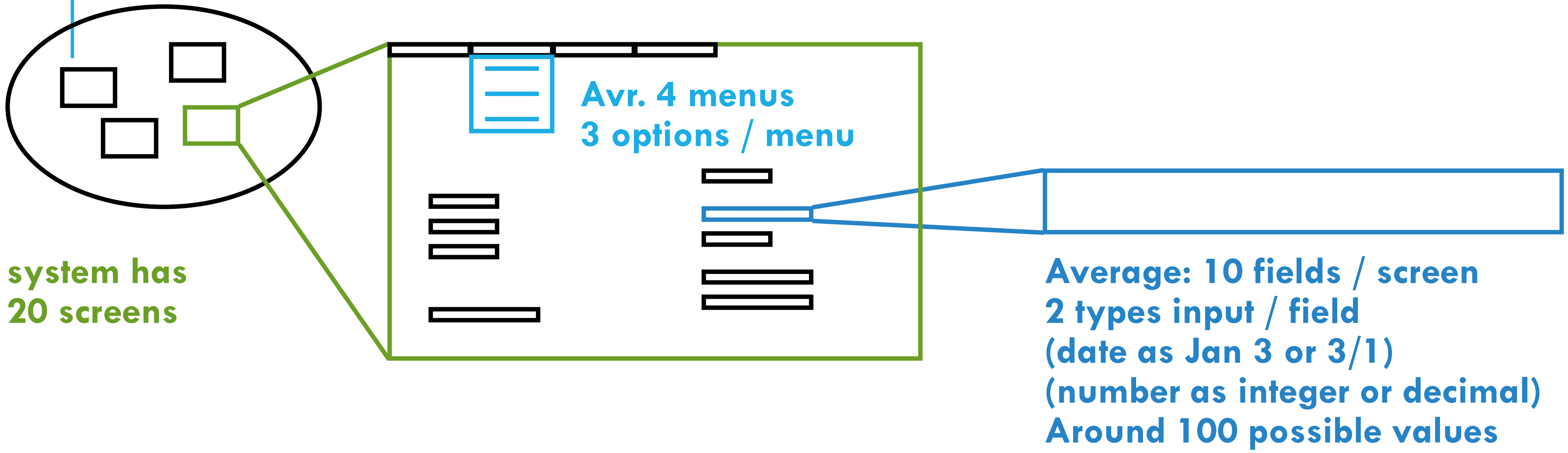
software faults can cause death or injury

- 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 <u>1 second</u> 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

} **i.e. where to place emphasis**

- what not to test (this time)

RISK

- allocate the time available for testing by prioritising testing ...

SEVEN TESTING PRINCIPLES

Principle 1: Testing Shows Defects Are Present In The Software

As per this principle, testing is a process which shows defects are present is software. Defects are identified by using different software testing execution techniques.

At the same time, testing doesn't prove that after finding defects that there are no defects present in the system. This principle talks about the reduction of a number of defects in software. There are always chances that the software has undiscovered defects, testing should not be considered as a proof of defect free software.

Principle 2: Exhaustive Testing Is Not Practically Possible

If we talk about this principle, it says it is not possible to test complete software. Test with all combinations of inputs and outputs. i.e. Test with all possible scenarios is not possible. Then you must be thinking how we will test the complete software. See, instead of performing complete or exhaustive testing we go for risk-based testing. Identifying the impact can help us to identify the module which are on high risk.

SEVEN TESTING PRINCIPLES

3rd Principle: Start Testing In Early Stage of SDLC

This principle asks to start testing software in the early stage of software development life cycle. This helps in identifying defects and fix them early with low budget and within assigned time period. It allows to handover software on time with expected quality.

4th Principle: Defects Clustering

Usually, maximum defects in software lie within the limited set of software areas. If you successfully identify this area, it's become quite a simple task for you to bring that sensitive area under the focus of testing. It is considered as one of the most efficient ways to perform testing efficiently.

5th Principle: The Pesticide Paradox

If you are using the same set of test cases repeatedly, then after some time those test cases do not find any new defects. So, it is always recommended to review and revise the test cases on a regular interval in order to find new defects. It is allowed to add new scenario or test cases even after the execution of particular test set.

SEVEN TESTING PRINCIPLES

6th Principle: Testing is dependent on context

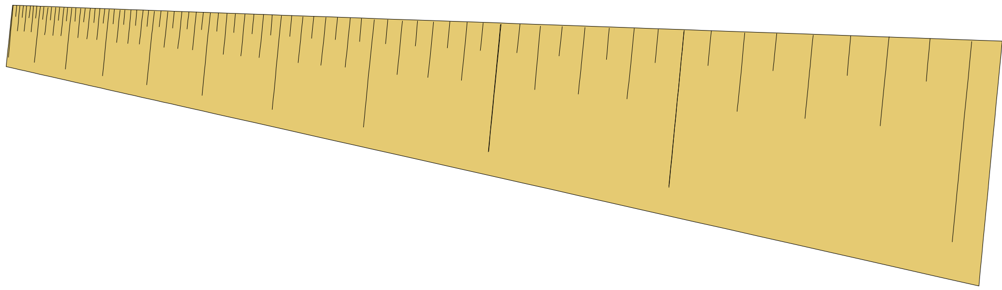
According to this principle; if you are testing web application and mobile application using same testing strategy, then it is wrong. This principle says the testing approach should be different and that’s depending on the application. Strategy for testing web application would be different from android mobile app testing.

7th Principle: Absence of errors – fallacy
This principle points towards the usefulness of the system. In other words finding the defects and fixing it will not help user unless and until the software is not developed according to the requirement.

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

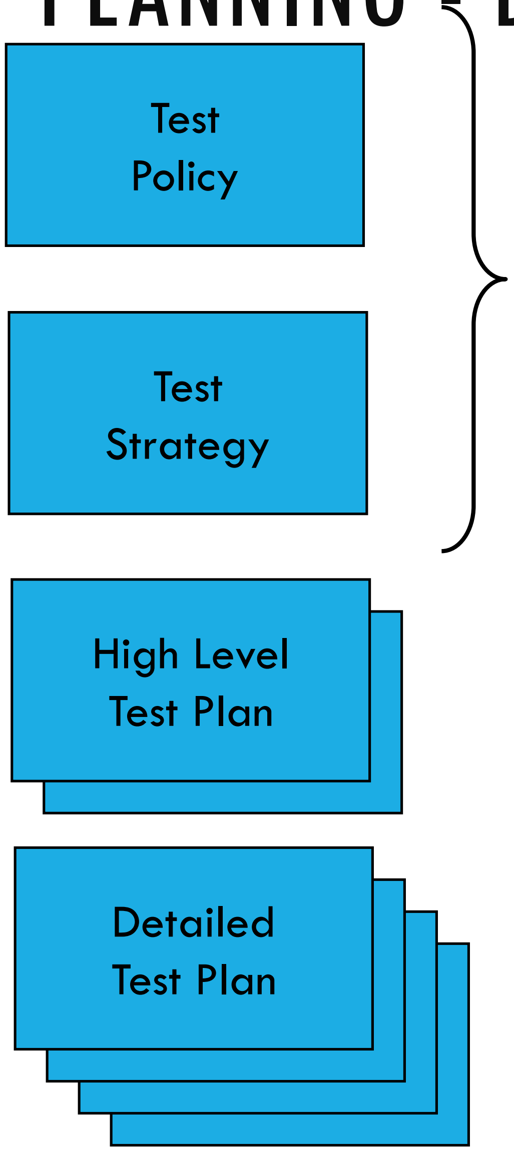
Principles

1	2	3
4	5	6

ISTQB / ISEB Foundation Exam Practice

CONTENTS

TEST PLANNING - DIFFERENT LEVELS

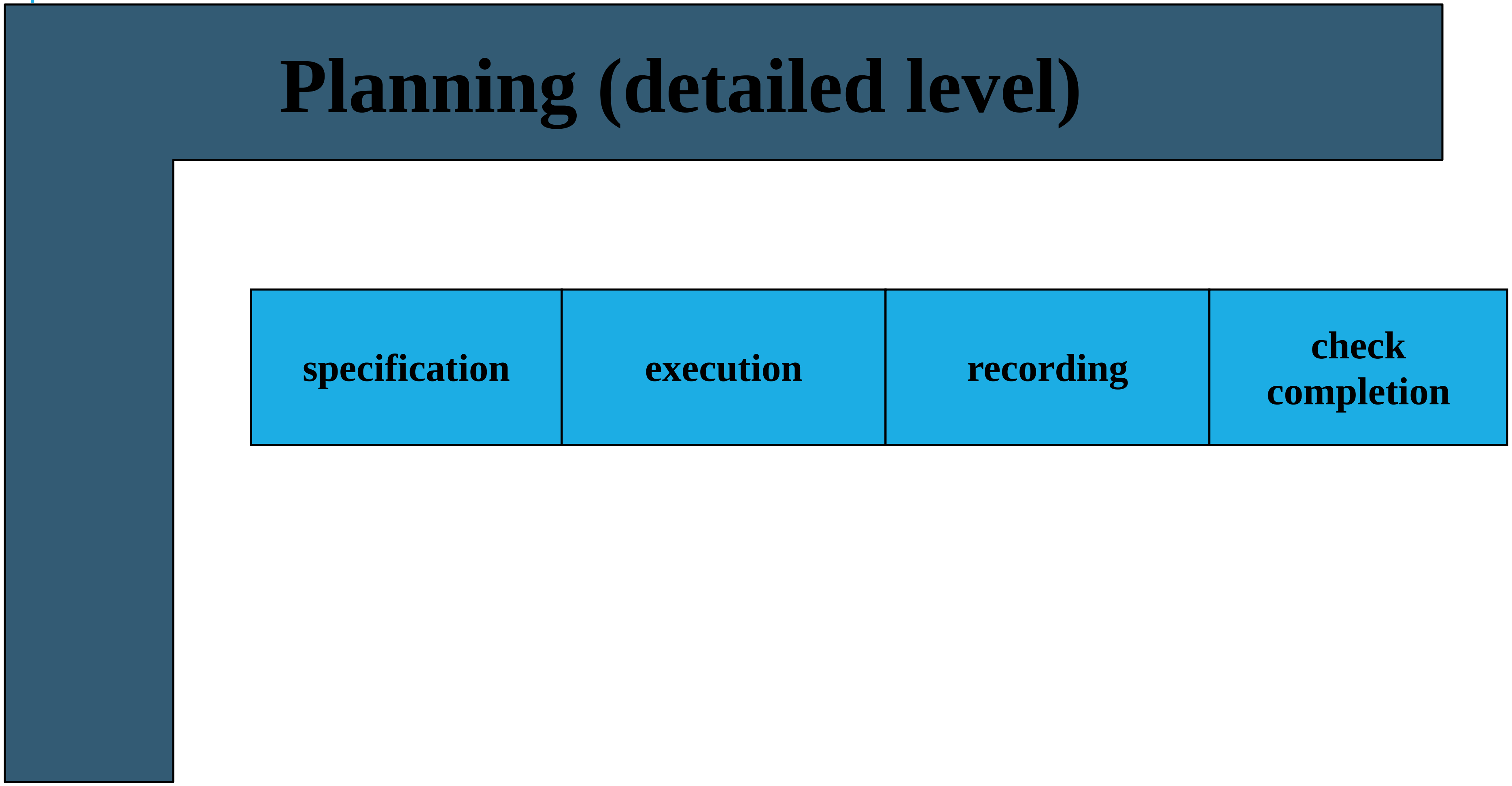


Company level

Project level (IEEE 829)
(one for each project)

Test stage level (IEEE 829)
(one for each stage within a project, e.g. Component, System, etc.)

THE TEST PROCESS



TEST PLANNING

how the test strategy and project test plan apply to the software under test

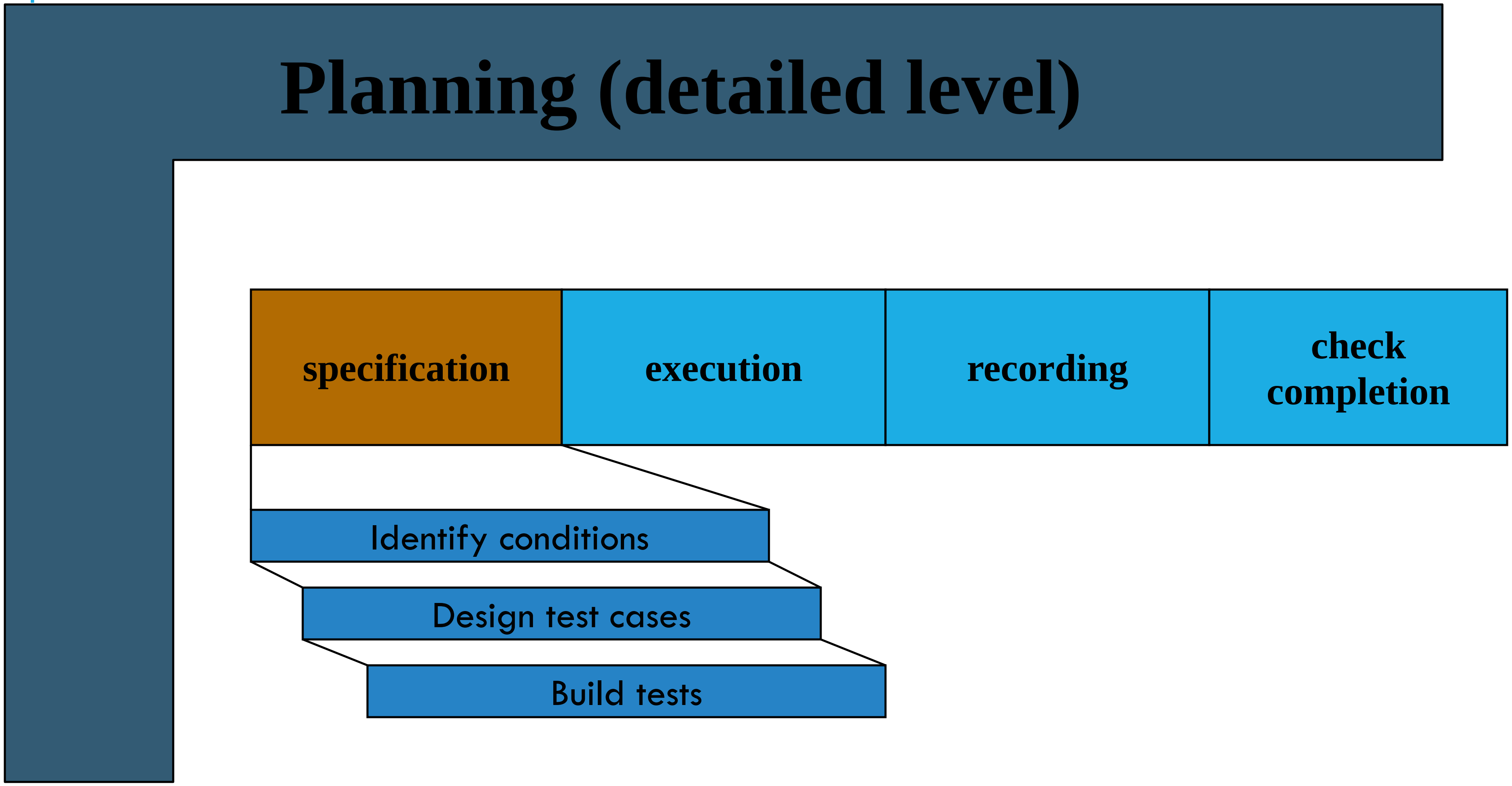
document any exceptions to the test strategy

- e.g. only one test case design technique needed for this functional area because it is less critical

other software needed for the tests, such as stubs and drivers, and environment details

set test completion criteria

TEST SPECIFICATION



A GOOD TEST CASE

effective

Finds faults

exemplary

Represents others

evolvable

Easy to maintain

economic

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

(determine ‘what’ is to be tested and prioritise)

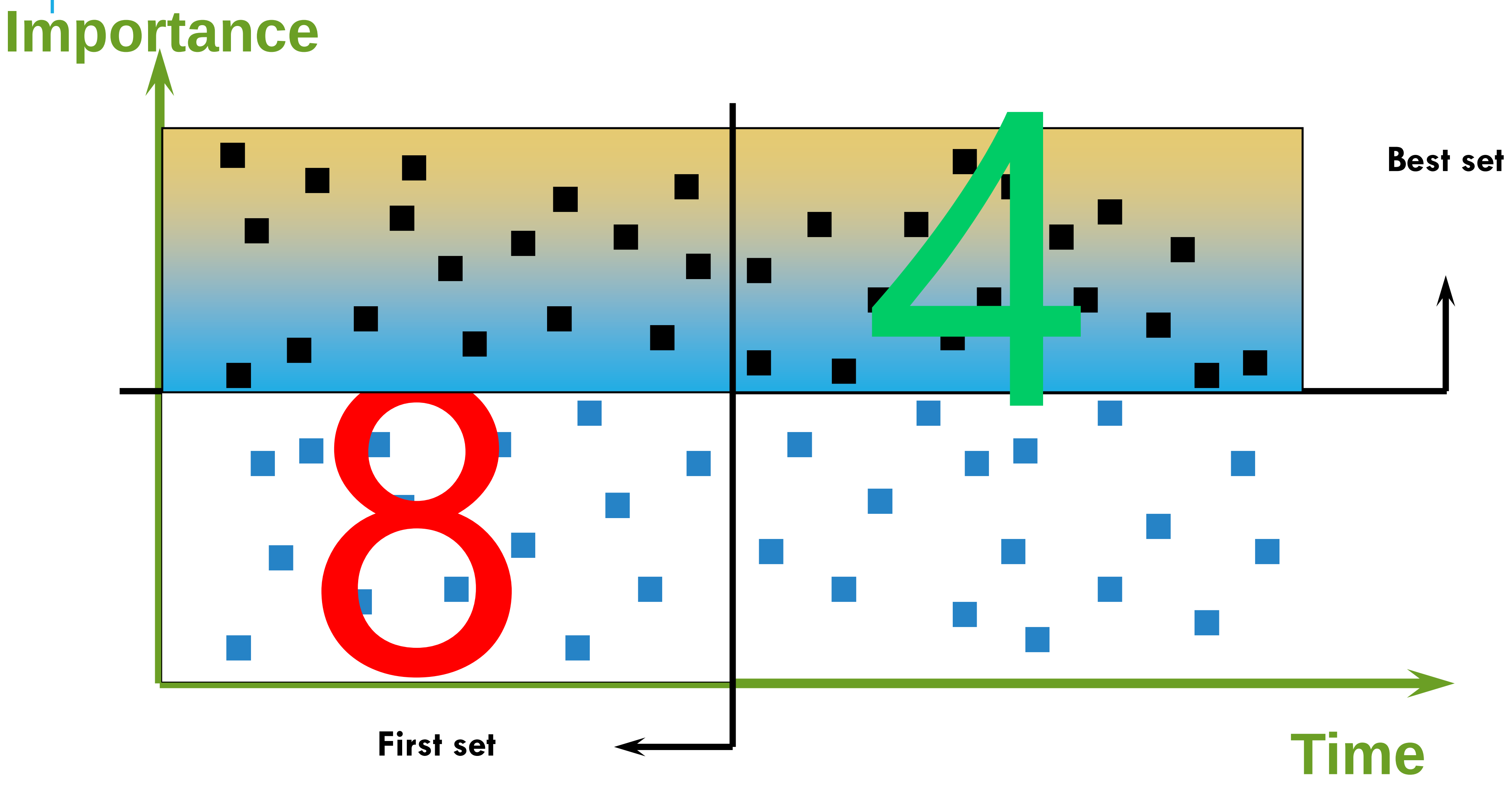
list the conditions that we would like to test:

- 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

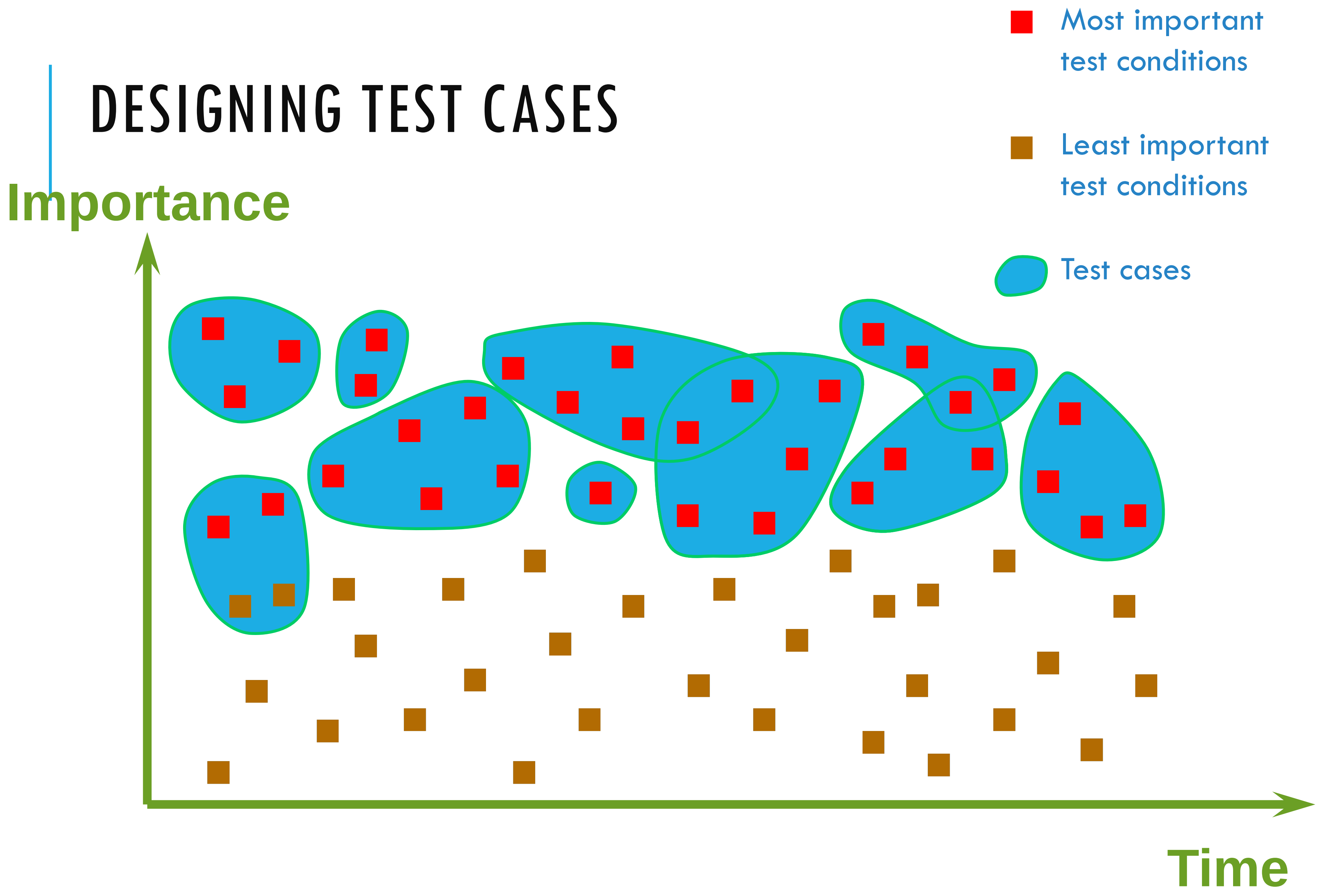
determine expected results

- 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


🟥 Most important test conditions
🟫 Least important test conditions
Test cases
Importance
Time

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

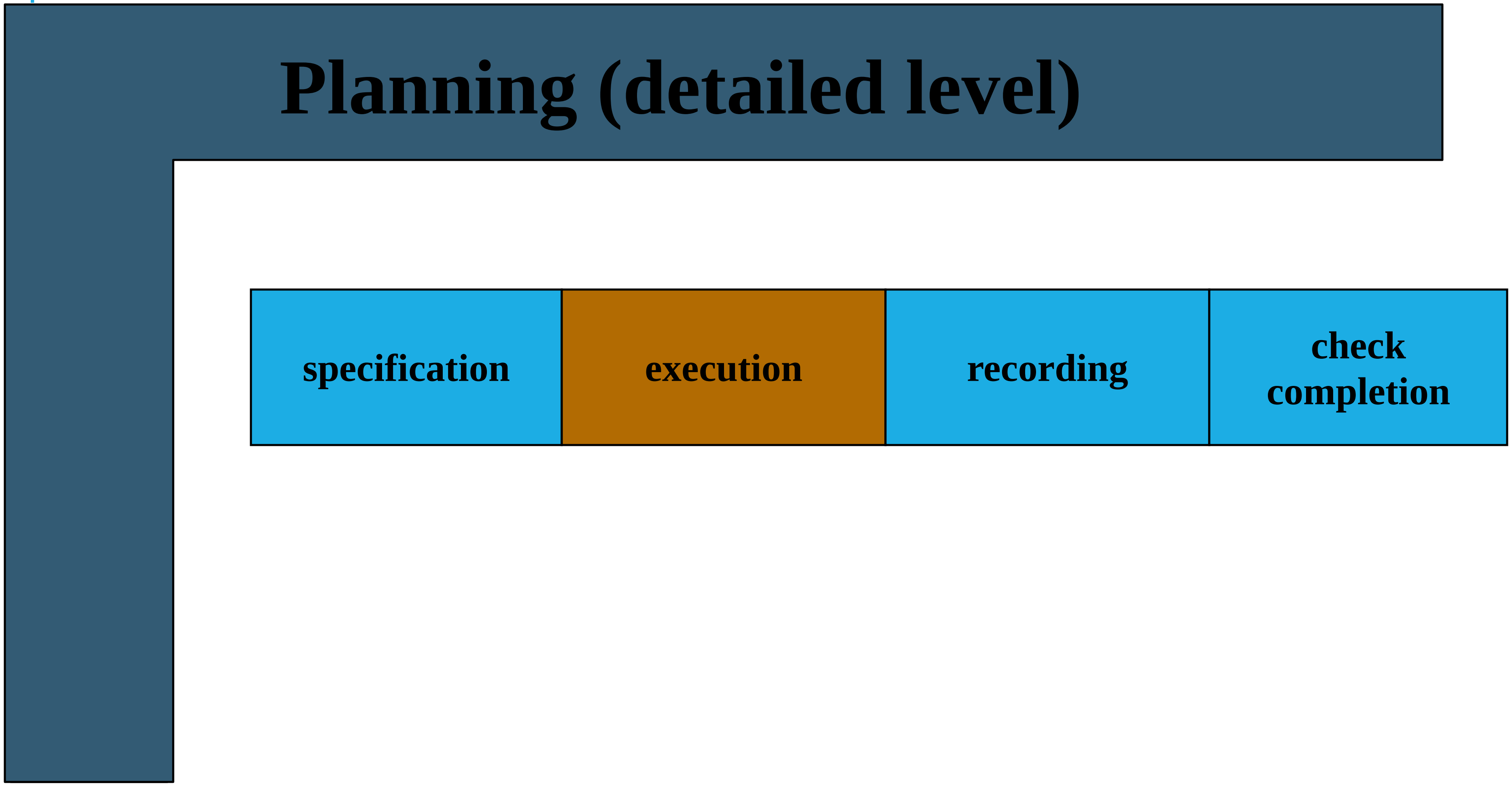
prepare test data

- 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

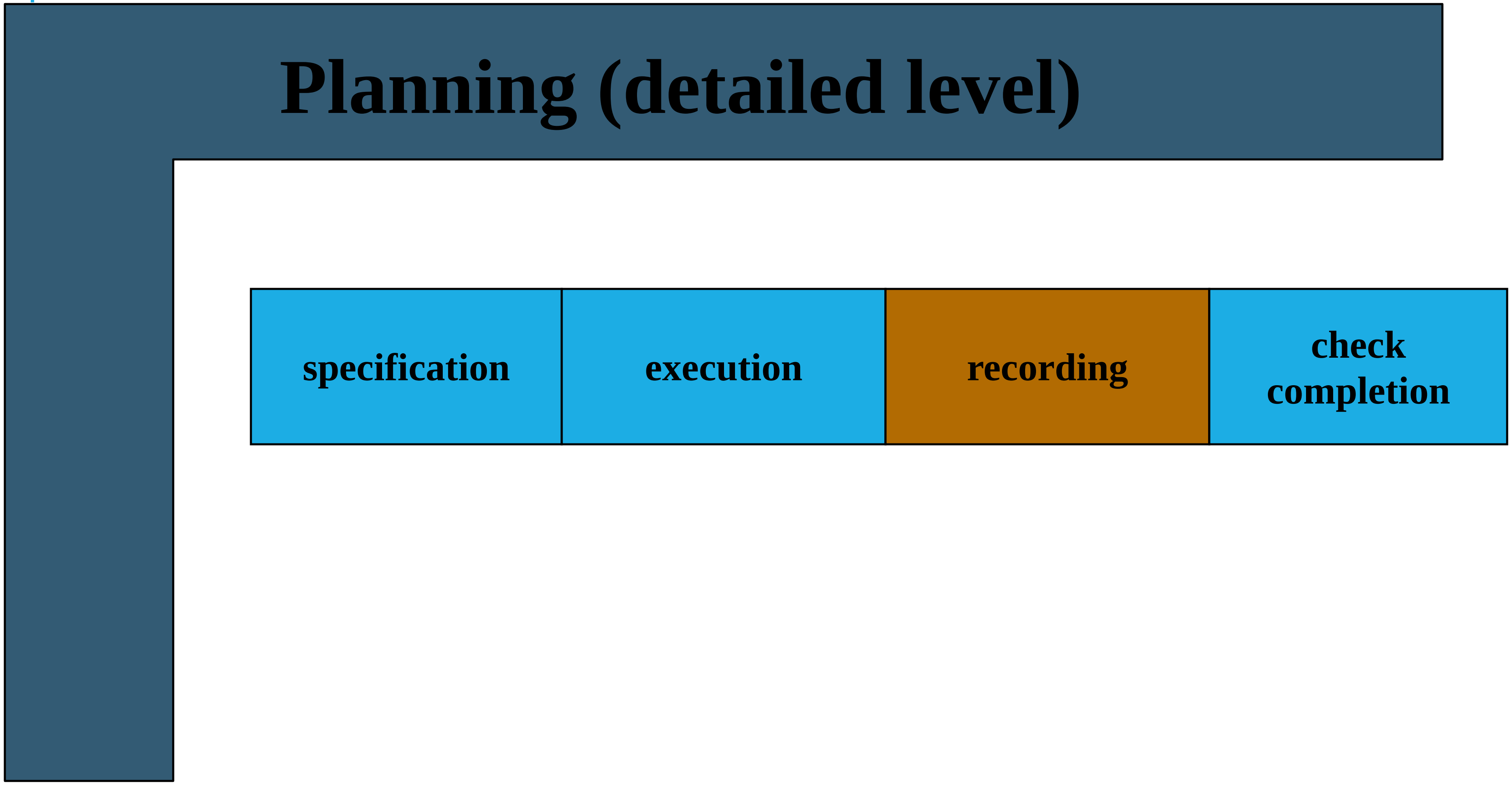


EXECUTION

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



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

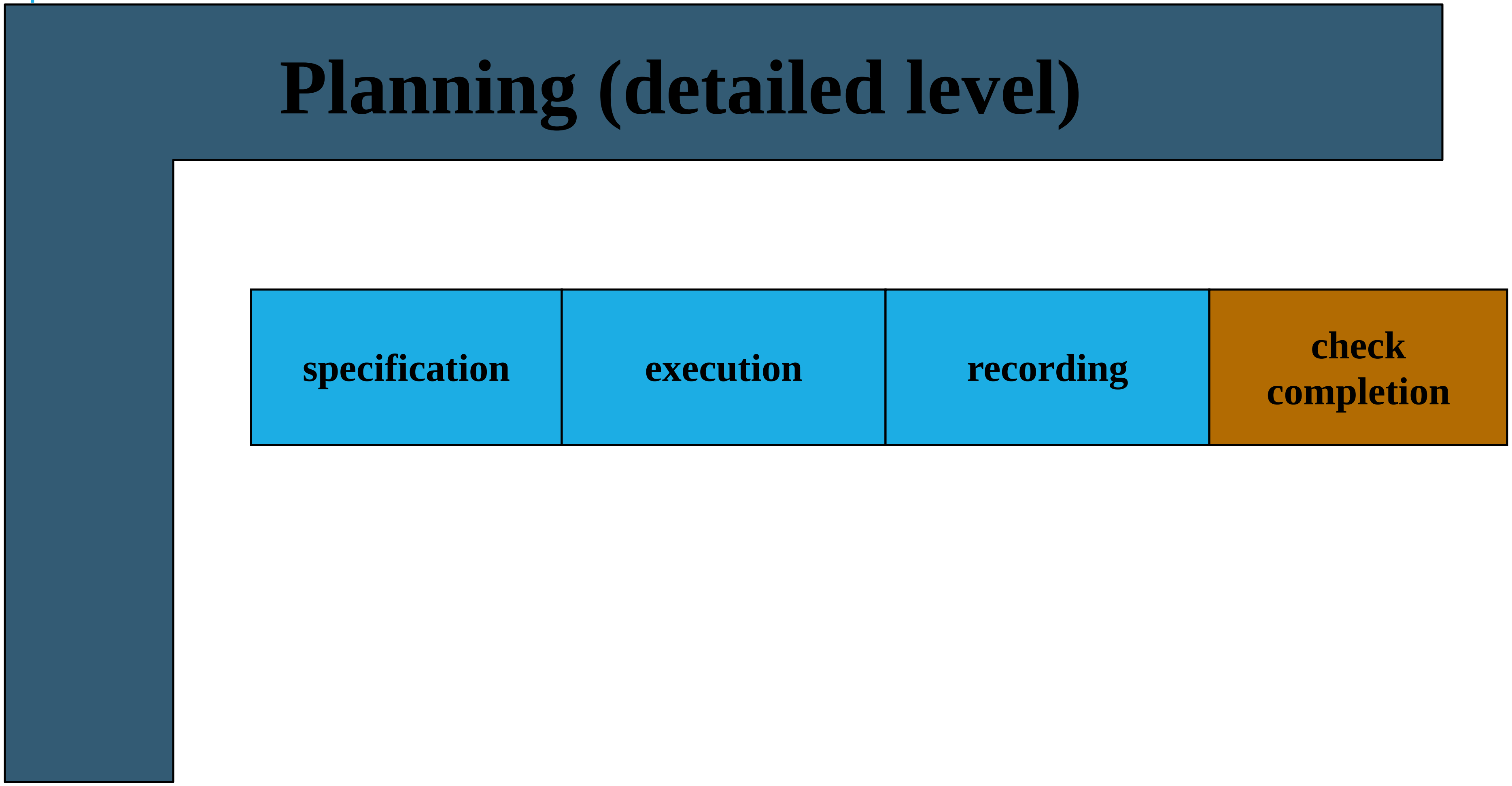
Compare actual outcome with expected outcome. Log discrepancies accordingly:

- 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)

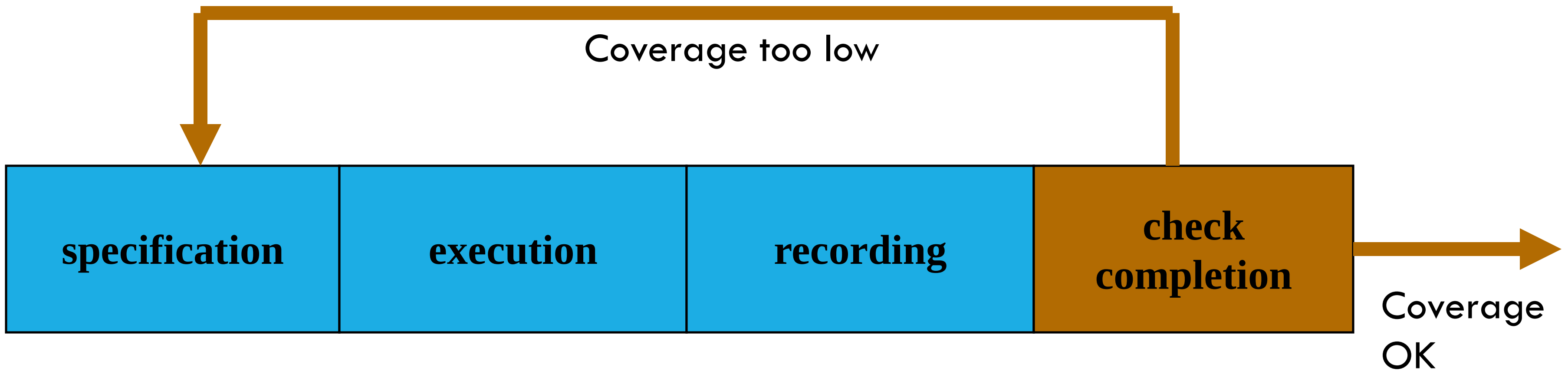
CHECK TEST COMPLETION



CHECK TEST COMPLETION

Test completion criteria were specified in the test plan

If not met, need to repeat test activities, e.g. test specification to design more tests

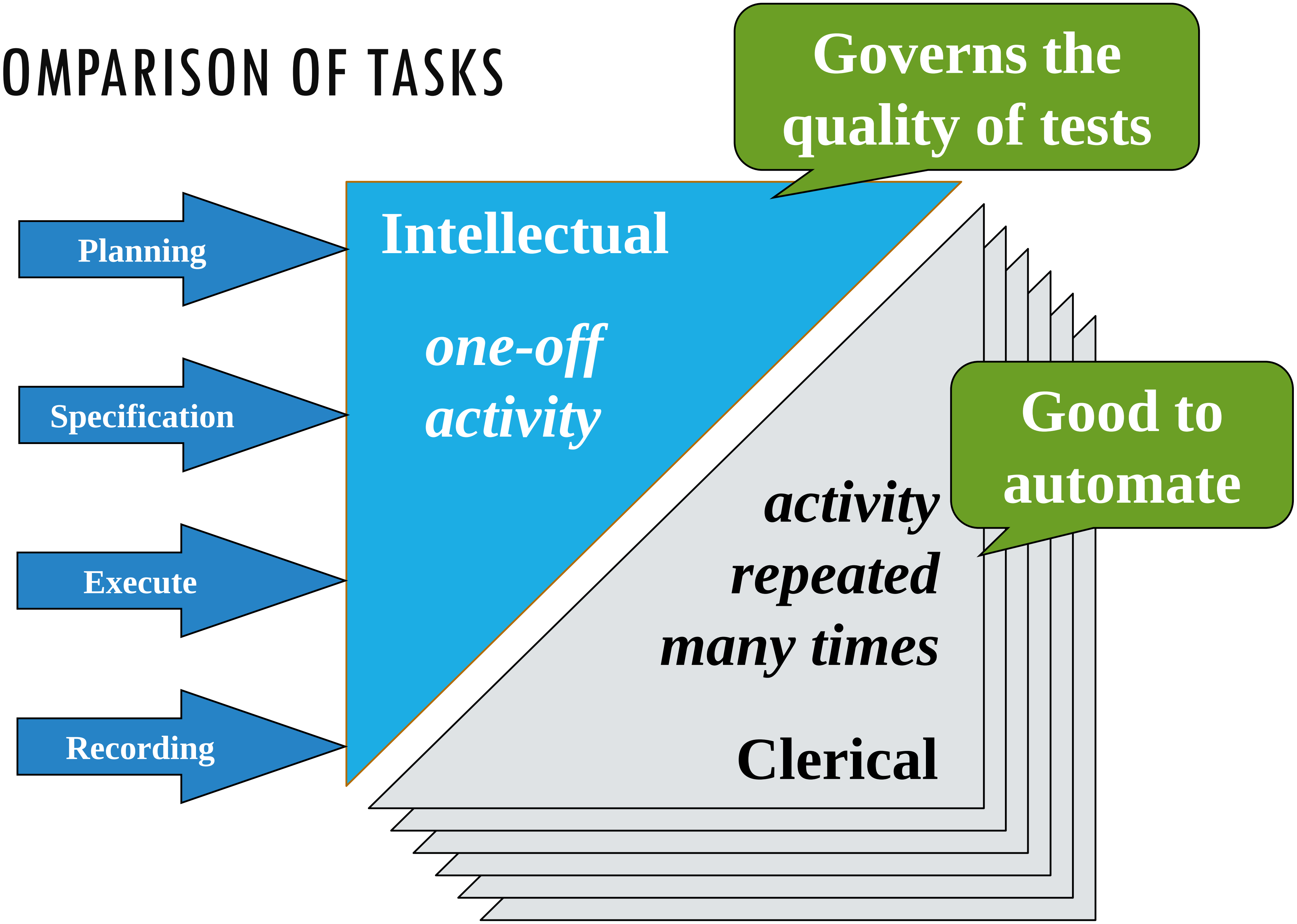


TEST COMPLETION CRITERIA

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



Principles

1	2	3
4	5	6

ISTQB / ISEB Foundation Exam Practice

CONTENTS

WHY TEST?

build confidence ✓

prove that the software is correct ✗

demonstrate conformance to requirements ✓

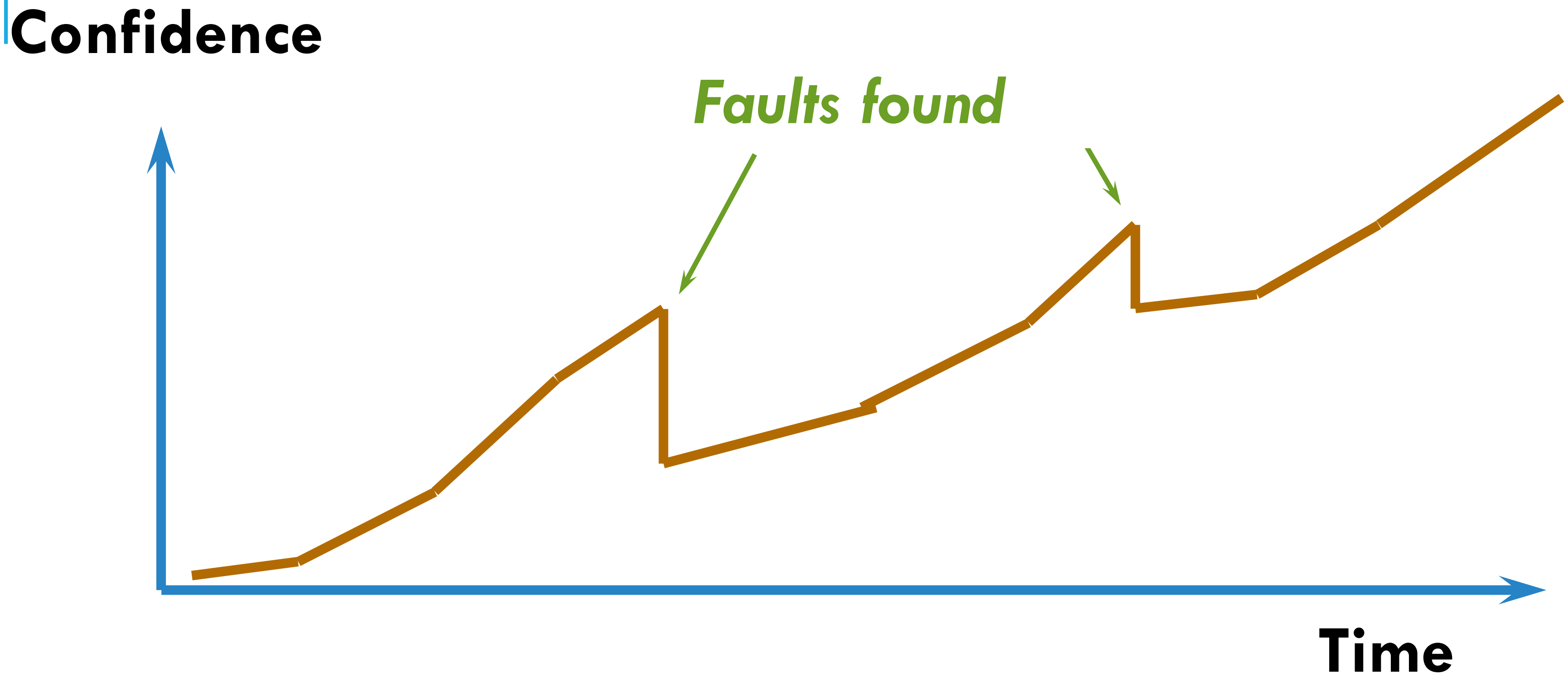
find faults ✓

reduce costs ✓

show system meets user needs ✓

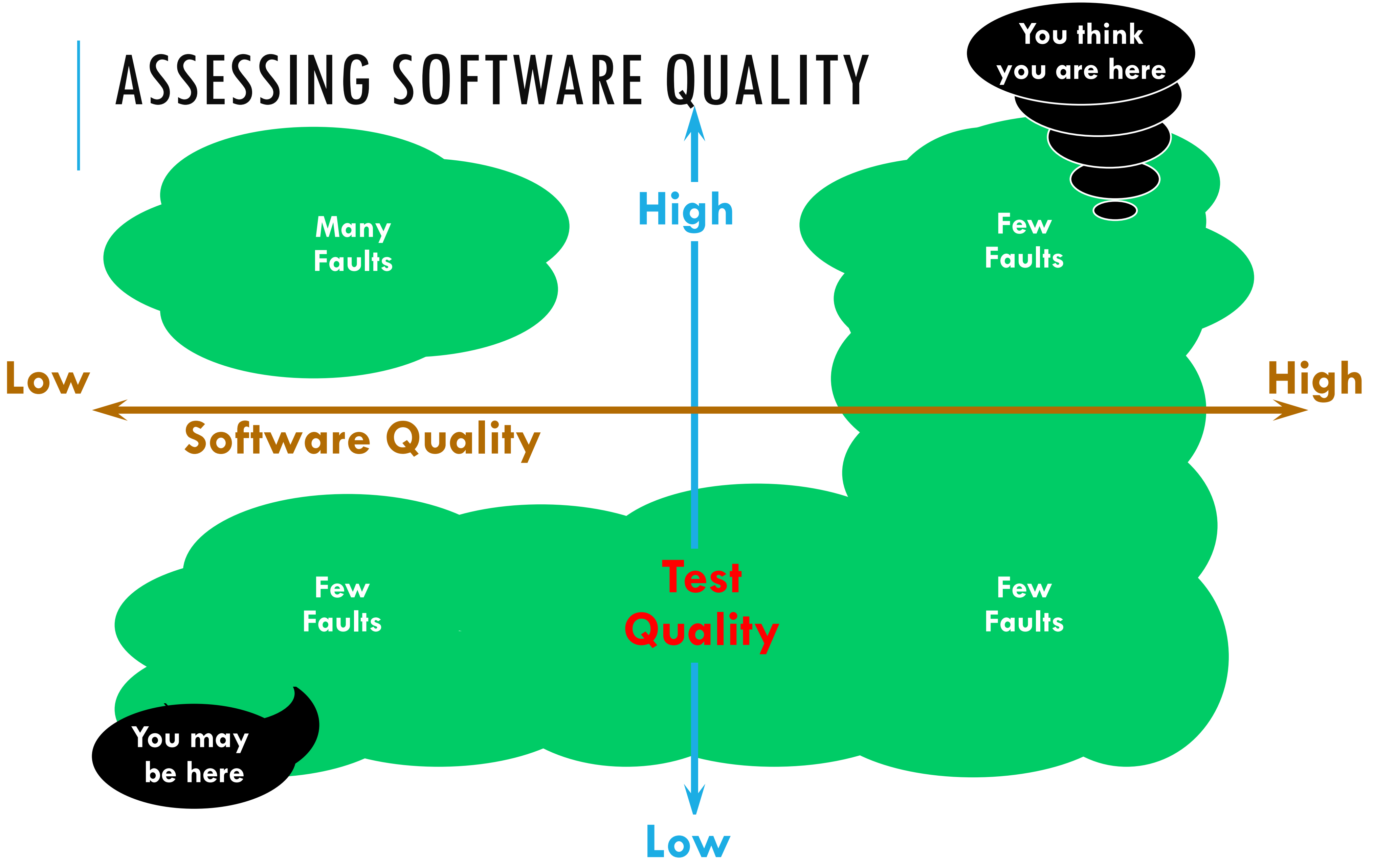
assess the software quality ✓

CONFIDENCE



No faults found = confidence?

ASSESSING SOFTWARE QUALITY



A TRADITIONAL TESTING APPROACH

Show that the system:

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

Goal: show working

Success: system works

Fastest achievement: easy test cases

$\longrightarrow$ **Result: faults left in**

A BETTER TESTING APPROACH

Show that the system:

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

Goal: find faults

Success: system fails

Fastest achievement: difficult test cases

$\longrightarrow$ **Result: fewer faults left in**

THE TESTING PARADOX

Purpose of testing: build confidence

The best way to build confidence

is to try to destroy it

WHO WANTS TO BE A TESTER?

A destructive process

Bring bad news ("your baby is ugly")

Under worst time pressure (at the end)

Need to take a different view, a different mindset ("What if it isn't?", "What could go wrong?")

How should fault information be communicated (to authors and managers?)

TESTER'S HAVE THE RIGHT TO:

- accurate information about progress and changes
- insight from developers about areas of the software
- delivered code tested to an agreed standard
- be regarded as a professional (no abuse!)
- find faults!
- challenge specifications and test plans
- have reported faults taken seriously (non-reproducible)
- make predictions about future fault levels
- improve your own testing process

TESTERS HAVE RESPONSIBILITY TO:

- follow the test plans, scripts etc. as documented
- report faults objectively and factually (no abuse!)
- check tests are correct before reporting s/w faults
- remember it is the software, not the programmer, that you are testing
- assess risk objectively
- prioritise what you report
- communicate the truth

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

None: tests designed by the person who wrote the software

Tests designed by a different person

Tests designed by someone from a different department or team (e.g. test team)

Tests designed by someone from a different organisation (e.g. agency)

Tests generated by a tool (low quality tests?)

Principles

1	2	3
4	5	6

ISTQB / ISEB Foundation Exam Practice

CONTENTS

RE-TESTING AFTER FAULTS ARE FIXED

Run a test, it fails, fault reported

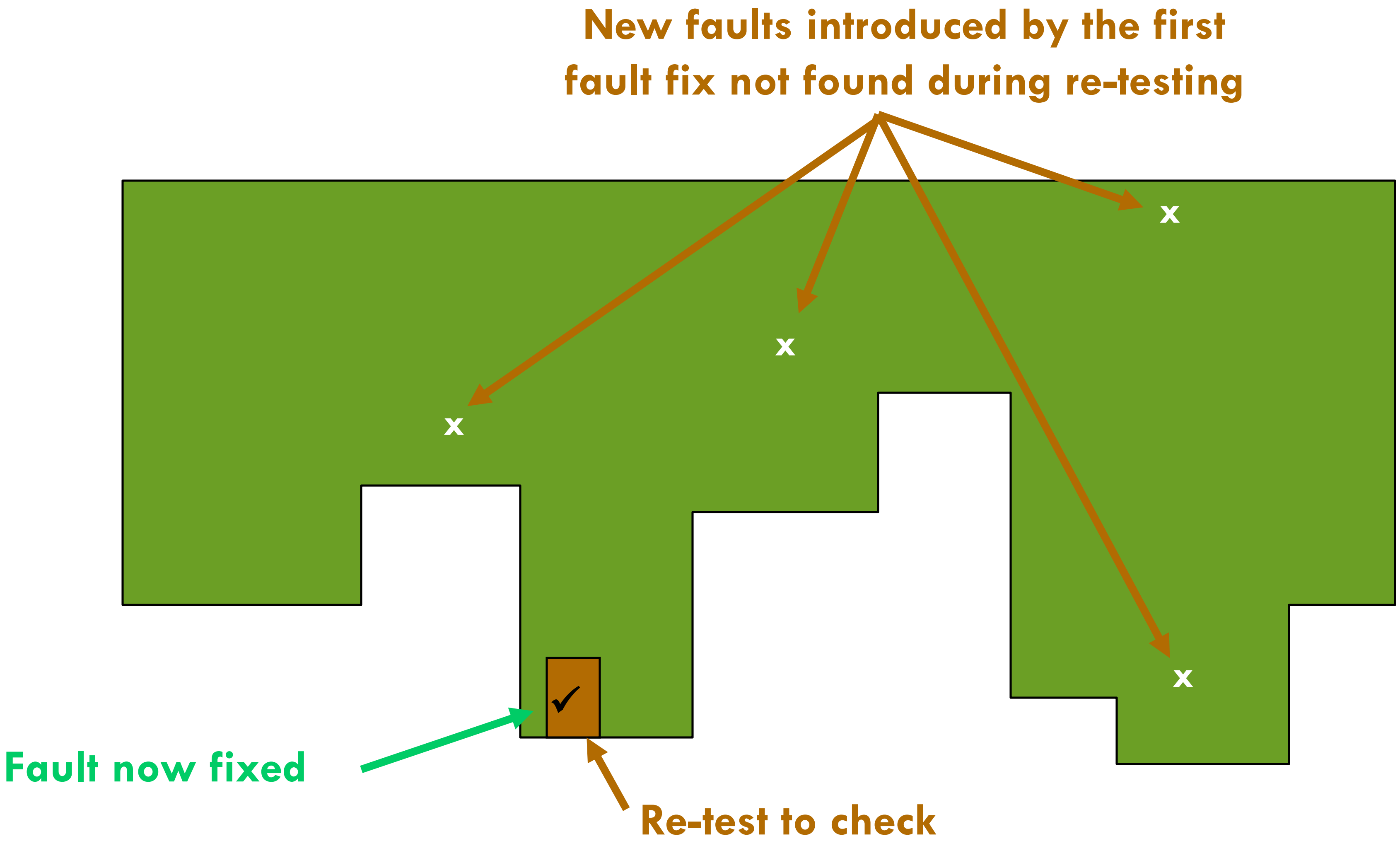
New version of software with fault "fixed"

Re-run the same test (i.e. re-test)

- must be exactly repeatable
- same environment, versions (except for the software which has been intentionally changed!)
- same inputs and preconditions

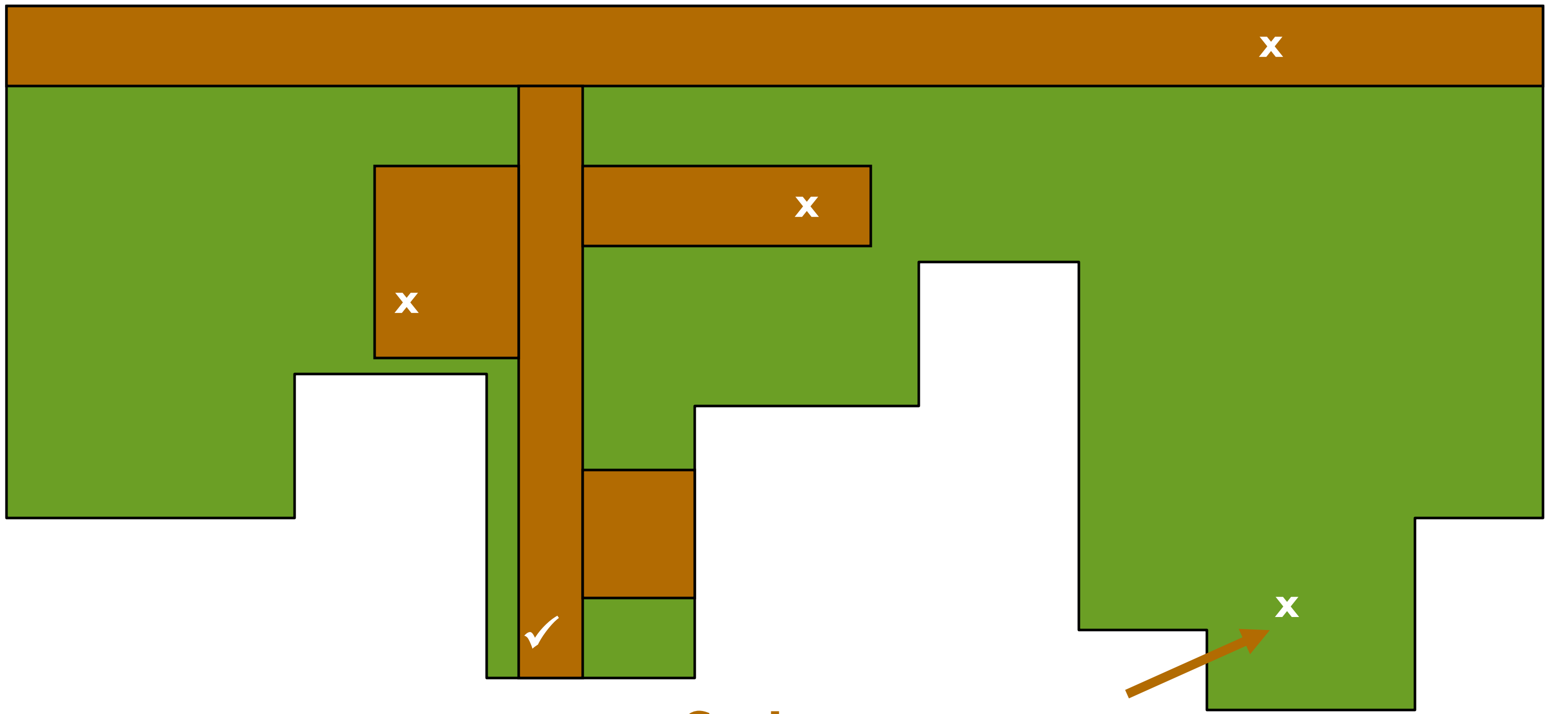
If test now passes, fault has been fixed correctly - or has it?

RE-TESTING (RE-RUNNING FAILED TESTS)


New faults introduced by the first fault fix not found during re-testing
x
x
x
x
✓
Fault now fixed
Re-test to check

REGRESSION TEST

to look for any unexpected side-effects



Can’t guarantee
to find them all

REGRESSION TESTING 1

standard set of tests - regression test pack

at any level (unit, integration, system, acceptance)

well worth automating

a developing asset but needs to be maintained

REGRESSION TESTING 2

Regression tests are performed

- after software changes, including faults fixed
- when the environment changes, even if application functionality stays the same
- for emergency fixes (possibly a subset)

Regression test suites

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

REGRESSION TESTING 3

Maintenance of the regression test pack

- eliminate repetitive tests (tests which test the same test condition)
- combine test cases (e.g. if they are always run together)
- select a different subset of the full regression suite to run each time a regression test is needed
- eliminate tests which have not found a fault for a long time (e.g. old fault fix tests)

REGRESSION TESTING AND AUTOMATION

Test execution tools (e.g. capture replay) are regression testing tools - they re-execute tests which have already been executed

Once automated, regression tests can be run as often as desired (e.g. every night)

Automating tests is not trivial (generally takes 2 to 10 times longer to automate a test than to run it manually

Don’t automate everything - plan what to automate first, only automate if worthwhile

Principles

1	2	3
4	5	6

ISTQB / ISEB Foundation Exam Practice

CONTENTS

EXPECTED RESULTS

Should be predicted in advance as part of the test design process

- ‘Oracle Assumption’ assumes that correct outcome can be predicted.

Why not just look at what the software does and assess it at the time?

- subconscious desire for the test to pass - less work to do, no incident report to write up
- it looks plausible, so it must be OK - less rigorous than calculating in advance and comparing

A TEST

A Program:

```
Read A
IF (A = 8) THEN
    PRINT ("10")
ELSE
    PRINT (2*A)
```

inputs	expected outputs
3	6?
8	10?

Principles

1	2	3
4	5	6

ISTQB / ISEB Foundation Exam Practice

CONTENTS

PRIORITISING TESTS

We can’t test everything

There is never enough time to do all the testing you would like

So what testing should you do?

MOST IMPORTANT PRINCIPLE

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

HOW TO PRIORITISE?

Possible ranking criteria (all risk based)

- test where a failure would be most severe
- test where failures would be most visible
- test where failures are most likely
- ask the customer to prioritise the requirements
- what is most critical to the customer’s business
- areas changed most often
- areas with most problems in the past
- most complex areas, or technically critical

Principles

1	2	3
4	5	6

SUMMARY: KEY POINTS

Testing is necessary because people make errors

The test process: planning, specification, execution, recording, checking completion

Independence & relationships are important in testing

Re-test fixes; regression test for the unexpected

Expected results from a specification in advance

Prioritise to do the best testing in the time you have