SZÉCHENYI EGYETEM
UNIVERSITY OF GYŐR

System and Software Testing
Rendszer és szoftvertesztelés
(GKN/LM_INTA/INTM057)

Csapó Ádám - csapo.adam@sze.hu
Varjasi Norbert – norbert.varjasi@sze.hu

Széchenyi István University
Department of Information Technology

Topics for today

- 7 principles of testing
- Key terms: errors, faults and failures
- Key terms: verification and validation
- Types of testing

The 7 principles of testing

1. Testing shows the presence of defects

Crucially, it does not show their absence

2. Exhaustive testing is impossible

Imagine that you have 5 input fields that accept 1-digit numbers… that's 10^5 combinations!

3. Testing is best done early

The sooner they are detected, the cheaper it is to fix them as bugs (and bad design leading to bugs) can have all kinds of ramifications. In addition, the sooner the client gets involved, the sooner it becomes possible to address potential failures of communication

4. Defects are usually clustered

Defects do not usually occur in isolation and have cross-effects. Also, not all modules will in general have the same complexity. A good conceptual model is offered by the Pareto principle: 80% of bugs can be expected to be found in 20% of the modules.

The 7 principles of testing

5. Pesticide paradox

If we keep running the same test suite (same test cases), eventually they will find no more bugs. Yet, new bugs will no doubt appear, while the previous ones were fixed.

So, the number and type of test cases should be varied and increased with time

As we will see, this idea is taken to the extreme in some paradigms, such as in Test-Driven Development (TDD)

6. Testing is context dependent

One should obviously use different tools to test an e-commerce site, a word processing program or a 3D graphics engine

7. Absence of errors is a fallacy

Testing reduces the probability of undetected defects remaining in the software. It does not guarantee error-free software.

Key terms: errors, faults and failures

Colloquially, people often use these terms interchangeably. But in a technical sense, they all have a specific meaning.

Failure: the apparent, incorrect functioning of a system (expected and received output are different)

Fault: the cause (state of a system) that eventually leads to a failure

Error: the programmer's / developer's / operator's mistake that leads to an eventual fault

Verification and validation

Verification:

Are we building the software right?

That is: is the software working as determined by its specifications

Main question: does the product satisfy the requirements as they were written?

Validation:

Are we building the right software?

That is: does the software fulfil the needs of the customer? Is this what the customer had in mind when commissioning the software?

Verification and validation

How can a software adhere to the requirements yet not fulfil the customer's requirements?

Easily!

There could have been communication problems between stakeholders and the development team

Formal specifications of a software are by nature a succinct representation of what is required – has both functional and qualitative parts!

Verification is done in a "lab environment", while the stakeholders are invested in real-world performance

Verification and validation

So, whereas verification testing is usually done at the component or sub-system level and carried out by the engineering team, validation testing is usually done in the real-world by the actual stakeholders behind the product.

One way to bridge the gap between the two is to perform early validation by simulating some parts of the system under test (dummy modules)

Key point to note: a product that has been verified / certified can be much more valuable on the market than one that hasn't.

Generally there are tons of documentation involved…

Methods of verification

In what ways can verification be performed?

Testing (system or parts thereof are in operation during test cases with specific expected results. Result of testing can be pass or failed)

Analysis (calculation or simulation that shows that the requirements are met. With analysis, no “expected output” is explicitly involved, but analysis can help inform test cases)

Demonstration (system is in operation but less formal than in testing)

Inspection (for physical systems, you can visually inspect them, or use NDI – non-destructive inspection tools like X-ray or other imaging techniques to verify the properties of the structure)

Types of testing – Black-box and White-box testing

Key question: based on what information are test cases created?

Black-box (a.k.a. Glass-box) testing:

Specification-based (i.e., not source code based)

We only have the specification at our disposal (let's pretend)

Only one option: run the code (or parts thereof), and see if it gives the right answer

White-box (a.k.a Structural) testing:

We also have the source code at our disposal. Thus, we can examine the coverage of our test cases

Types of testing – Levels of testing, testing roles

Key question: at what level are we carrying out the tests?

Component tests:

Carried out by engineers / the developers of the software

Focus on a single component of the software. Two main types:

a.) unit tests – functions / methods with specific inputs and expected outputs. When the code changes, unit tests are re-run (hence the term regression testing)

b.) module tests – generally focus on non-functional requirements (speed, memory management / leaks, identifying bottlenecks)

Types of testing – Levels of testing, testing roles II

Key question: at what level are we carrying out the tests?

Integration tests:

Carried out by engineers / the developers of the software

Individual components are combined and tested as a group. This is expected to happen after unit testing / module testing and prior to validation testing, based on an integration test plan.

Several types of integration tests exist:

a.) big-bang testing: all components of the system are combined together and tested together at once. Bugs that are found may be more difficult to localize, and also all components have to be ready.

b.) bottom-up testing: lower-level components are integrated and tested first, followed by higher-level components that rely on their services. This leads to easily traceable progress (x% of testing done) but requires relevant components to be ready in time

Types of testing – Levels of testing, testing roles III

Key question: at what level are we carrying out the tests?

Integration tests:

Carried out by engineers / the developers of the software

Individual components are combined and tested as a group. This is expected to happen after unit testing / module testing and prior to validation testing, based on an integration test plan.

Several types of integration tests exist:

c.) top-down testing: higher-level components are integrated & tested first, while lower-level components are saved for later. In this case, the functionality of lower-level components is simulated using “fakes” or “stubs”.

d.) sandwich testing (a.k.a. hybrid testing) combines the bottom-up and top-down approach to test the integration of mid-level components

Types of testing – Levels of testing, testing roles IV

Key question: at what level are we carrying out the tests?

System tests:

Carried out by engineers / the developers of the software following integration tests

System tests are carried out on the whole system, to identify both functional and non-functional issues. Issues can be found in the linkages of components or in the functionality of the system as a whole

Several types of system tests exist:

a.) destructive testing: the goal is to have the software fail (the specimen break) in uncontrolled conditions. This provides information on the robustness of the system. Only viable when enough products exist and the cost is not too high

b.) non-destructive testing: the default option in system testing

Types of testing – Levels of testing, testing roles V

Key question: at what level are we carrying out the tests?

System tests:

Carried out by engineers / the developers of the software following integration tests

System tests are carried out on the whole system, to identify both functional and non-functional issues. Issues can be found in the linkages of components or in the functionality of the system as a whole

Several types of system tests exist:

c.) fault injection techniques: in the realm of software, these can be effected at compile time or at runtime

Compile time – lines of code are mutated / erroneous code is added in predictable ways

Runtime – memory space can be corrupted, system calls can be intercepted, network packets can be modified. In the case of fuzzing, semi-correct input is provided, and the effects deep down in the software analyzed (usually done with software expecting structured input)

Types of testing – Levels of testing, testing roles VI

Key question: at what level are we carrying out the tests?

Acceptance tests (“Átviteli teszt” magyarul):

Acceptance testing is a different word for validation testing.

It is carried out by / in the presence of the end users / stakeholders, to ascertain that the product meets both the requirements outlined in the specifications and also fits into the environment and the business processes of the commissioner.

Types of testing – Static versus dynamic testing

Finally, an important distinction is whether the system has to operate / code has to be run in order to carry out the tests

Static testing

Code is not run during this type of test.

Clearly this entails that the testing is white-box (otherwise what would one have to go on?)

Many different kinds of static testing:

a.) Code inspection by humans: can range from informal ("running code in one's head) to formal (code review), carried out internally or with the help of an auditing organization.

b.) Code inspection by software (code analysis). Static analysis tools can operate based on either code only, or based on code and a "model" (a formal description of the kinds of errors one expects to find, such as starting indexing at 1 rather than 0, failing to free allocated memory etc.). Note that even linting counts as static analysis, and of course style is important in reducing errors.

Types of testing – Static versus dynamic testing II

Finally, an important distinction is whether the system has to operate / code has to be run in order to carry out the tests

Dynamic testing

Code is run during this type of test. This kind of testing can be black-box or white-box. It is not necessary to look at the code, but if it is available, then that's all the better!

Prior to dynamic testing, one must decide a few questions:

What is the test condition ("a tesztelés alanya" magyarul)

i.e. what parameter of the system are we testing? (functionality, transaction, capability, performance metric, structural unit)

What are the test cases?

Usually many, many test cases for each condition

What is the test procedure?

When, under what circumstance, how often are the tests carried out and by whom? How are the results reported and bugfixes tracked?

Types of testing – Static versus dynamic testing III

Finally, an important distinction is whether the system has to operate / code has to be run in order to carry out the tests

Dynamic testing

Code is run during this type of test. This kind of testing can be black-box or white-box. It is not necessary to look at the code, but if it is available, then that's all the better!

Prior to dynamic testing, one must decide a few questions:

What is the test coverage? (in white-box case). What fault model do the tests adhere to? (maybe it is assumed that two different modules won't produce failures at the same time, etc.).

Models (a different type) can be also useful in the context of generating test cases (for example, a finite state automaton could do this with different parameterizations)

Types of testing – Static versus dynamic testing IV

Finally, an important distinction is whether the system has to operate / code has to be run in order to carry out the tests

Dynamic testing

Code is run during this type of test. This kind of testing can be black-box or white-box. It is not necessary to look at the code, but if it is available, then that's all the better!

Kinds of dynamic testing:

- Specification-based testing (black-box, functionality-first approach)
- Model-based testing (black-box, with test cases generated using UML diagrams, finite state automata, real-world data sampling etc.)
- Structure-based testing (white-box, with test cases generated with a view toward high coverage or statements, branches and code paths)

Testing – a high-level overview

Time in lifecycle	Test category	Done by whom	Test type	Examples
1	Unit testing / Module testing	Developers	Component-level, static or dynamic, black-box or white-box	Code review, static analysis, regression tests
2	Integration testing	Developers / engineers	Between-components, dynamic, black-box or white-box	Big-bang, bottom-up, top-down, sandwich
3	System testing	Engineers	System-level testing, dynamic, black-box or white-box	Destructive, non-destructive, fault injection
4	Acceptance testing	Company leadership / stakeholders	System-level testing, dynamic, black-box	Demonstration, performance analysis

Practical assignment

Download / install a static code analysis tool in a programming language of your choosing.

Investigate the kinds of errors / warnings the tool provides.