UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

Compulsory assignment 1

Rune Hovde , Hans Henrik Sande , David Buverud

inf3121, spring 2017

1 Exercise 1: The seven testing principles

1.1 Testing shows the presence of defects

There is no way of knowing how many defects a software contains. The testing process will only show the presence of defects, never the absence of them. It will however, reduce the number of unknown defects (and chance of there being any remaining).
Example: If you ran a test, and the test came back showing a defect, that means that the software has defects. However, it says nothing about how many defects there are in total.

1.2 Exhaustive testing is impossible

Testing all the possible cases is either impossible (e.g. because there are too many things/cases to test), or takes to much resources. It is better to prioritize what to test, and what not to.
Example: If your program takes 1000 different inputs at 16 bytes each, you get 161000, and as each Byte leaves us with 28 different variations – the number of different inputs to test is $2^{8x16^{1000}}$.

1.3 Early testing

The earlier the test activities starts, the better.
Example:
"Let's start testing during the last phase" – BAD!!
"Let's start testing during the first phase" – GOOD!!

1.4 Defect clustering

Defects often has a tendency to cluster together – so if you find a defect, you should look close to it for others.
Example: Testing showed 3 defects in a method/function. Instead of fixing these 3 defects and move on, it might be smart to test this method/function more thoroughly.

1.5 Pesticide paradox

It is important to change up the tests every now and then. If you keep running the same tests over and over throughout the development of the software, it will give a false sense of confidence. There could be many defect clusters left undiscovered, this is why it is important to use different test strategies on different parts of the program, so you can better filter defects.

1.6 Testing is context dependent

Depended on what kind of software you are creating, the testing process will differ. Some kind of projects/software demands more thorough and precise testing.
Example: It's more harmful if a bank-server stops working for a day, than if ponyclub.com stops working for a day. If people and businesses can't reach their money, it will have a higher impact on society than if ponyclub-members can't reach their ponies.

1.7 Absence-of-errors fallacy

If the system you are creating doesn't do what it's supposed to do, fixing (and finding) any number of defects doesn't change that fact. A program can be virtually bug-free, but still miss on what the program is supposed to do.
Example: Just as with exam-questions: your answer can be 100% correct, but if you answer the wrong question, your answer will still be counted as incorrect.

2 Exercise 2:

(a) With the V-model during the different part of the waterfall model, you also develop the different tests. E.g, during user requirements you also develop the acceptance tests. Once you start implementing you run the already developed tests. The keyword is parallelism, you design tests parallel with development activities.
One of the benefits is that testing does not happen as the final stage of the development process. As stated in one of the seven principles of testing, early testing is important because the longer the defect is in the software the harder it is to find and remove it, in other words you reduce time spent and costs. This also makes time estimation easier, as the probability of discovering huge amounts of defects at the end is slim.

(b) As the V-model was designed with the waterfall method in mind, and agile programming often fixes the "one-way" problem found in the waterfall method (not moving back a step while developing), it's hard to see how one could implement the V-model in an agile programming project. While V-model implements a better way to work with the waterfall method, it is still not iterative.

(c) As said before, Early Testing (third principle) is safeguarded, but the absence-of-fallacy principle (seventh principle) is also covered because testing during the earlier phases makes it easier to spot errors in for example the user requirements. One could also argue that the exhaustive testing is impossible principle (second principle) is better covered with the V-model, because reviewing documents before implementing them makes it easier to prioritize what to test.

(d) Test levels are test activities that are grouped together, having the same position in the "test hierarchy". E.g component tests are "lower" down the hierarchy than integration tests.

The four different test levels are:

- **Component testing:** searches for defects in, and verifies the functioning of software items that are separately testable. Component testing are typically based on the requirements and detailed design specification applicable to the component under test, as well as the code itself.
- **Integration testing:** tests interfaces between components, interactions to different parts of a system such as an operating system, file system and hardware or interfaces between systems. Integration tests are typically based on the software and system design.
- **System testing:** is concerned with the behavior of the whole system/product as defined by the scope of a development project or product. It may include tests based on risk analysis reports, system, functional, or software requirements specifications, business process, use cases, or other high level description or system behavior, interactions with the operating system, and system resources.
- **Acceptance testing:** Testing if the system meets the user needs, requirements etc.

Test types:

- **Functional testing:** tests the functions of component or system. It refers to the activities that verify a specific action or function of the code. Functional test tends to answers questions like 'can the user do this' or 'does this particular feature work'. This is typically described in a requirements specification or in a functional specification.
- **Non-functional testing:** the quality characteristics of the component or system is tested. Non-functional refers to aspects of the software that may not be related to a specific function or user action such as scalability or security. 'How many people can log in at once?' Non-functional testing is also performed at all levels like functional testing.
- **Structural testing:** is the testing of the structure of the system component. Often referred to as 'white box testing' because we are interested in what is happening on the inside of the system. Therefore testers must have the required knowledge of the internal implementations of

the code.
Testing related to changes: testing to see if recent changes has caused other unexpected defects to occur in the system.

3 Exercise 3:

(a) – *Explain, in your own words, the difference between static and dynamic testing techniques.*

Static testing is done without executing the specific code, while dynamic executes it. Static uses different kinds of manual examinations, like reviews, to find defects or other wrongdoings in the code. Dynamic looks at input and output from executing the code.

(b) – *Explain the different steps in a formal review.*

- **Planning:** This is where you define the review criteria, select the personell, allocate roles, define the entry and exit criteria for more formal reviews, select which parts of the document to review and lastly check the entry criteria.
- **Kick-off (optional):** Here you distribute the documents and explain the objectives, process and documents to the participants.
- **Preparation:** Each reviewer individually reviews the document using information provided, and identifies questions, comments and defects.
- **Review meeting:** The main meeting, which consists of three phases:
  - **Logging phase** Each issue and defect that has been identified during the preparation phase are logged, and given a severity class (either critical, major or minor).
  - **Discussion phase** If some, or any issues needs discussion, this will be done during this phase.
  - **Decision phase** A decision on the document has to be made during this phase, sometimes based on the exit criteria.
- **Rework:** If the documents has too many defects during this phase, then the document has to be reworked.
- **Follow-up:** Check that defects have been addressed. Gather metrics and check exit criteria.

(c) – *A 'walkthrough' and an 'inspection' are two different types of reviews. Explain the difference between them; including their purpose and the roles of the participants.*

inspection is used to verify the compliance of the product with specified standards and requirements. It is done by examining, comparing the product with the designs, code, artefacts and any other documentation available. It needs proper planning and overviews are done on the planning to ensure that inspections are held properly. Lots of preparations are needed, meetings are held to do inspections and then on the basis of the feedback of the inspection, rework is done.
In a **walkthrough** the author presents their developed artifact to an audience of peers. Peers question and comment on the artifact to identify as many defects as possible. It involves no prior preparation by the audience. Usually involves minimal documentation of either the process or any arising issues. Defect tracking in walkthroughs is inconsistent.

4 Exercise 4:

(a) *Why are specification-based testing techniques also called 'black-box' techniques?*
Because in specification-based testing we often (mostly) look at the output from a system or component, based upon the input. E.g in an calculator-app, we look at the output from pressing (2 + 2). If the output is something other than whats expected (4), then we found a defect. We put a "black box" over the structure of the system/component.

(b) *Why are structure-based testing techniques also called 'white-box' techniques?*
Because here the testers required knowledge of how the software is implemented, how it works. In white-box testing the tester is concentrating on how the software does it. For example, a structural technique may be concerned with exercising loops in the software.
What is usually the purpose of these techniques?
Developers use structured based testing in component testing and component integration testing, especially where there is good tool support for code coverage. You test paths within a unit, paths between units during integration, and between subsystems during a system–level test.

Though this method of test design can uncover many errors or problems, it has the potential to miss unimplemented parts of the specification or missing requirements. The purpose of black-box is to examine the functionality of an application without looking into its internal structures or workings.

(c) *When do we sue experience-based testing specification-based techniques? What are the benefits and drawbacks related to these techniques?*
Usually on low-risk systems or under extreme time pressure. A benefit could be that due to previous experiences they'll have insight as to what might go wrong, and be aware of typical pitfalls. A disadvantage could be being fooled by previous experiences, that a similar, but not identical, situation occurs and you rely too much on previous experiences which does not yield the most satisfying result, whereas perhaps not relying too much on previous experiences would've been better.

(d) *Would you say that some testing techniques are better than others? What is your opinion on the use of the different techniques?*
It's all very situational, they all serve different purposes regarding to what and how they test, and there is a time and place for all of them respectively.

5 Exercise 5:

(a) The valid equivalence partitions would be the scopes from each of the different zodiac signs. E.g March 21 – April 19 is one valid parition, April 20 – May 20 would be another etc. Invalid partitions would be any dates that doesn't exists, e.g June 32, Julaiuis 4, or January -1.

(b) The boundary values would be the dates you see in the given picture from the assignment. E.g March 21 and April 19 for Aries, and April 20 and May 20 for Taurus.

(c)
1. December 21^{st} gives error about invalid date
2. June 31 st gives Cancer, even though the moth only has 30 days.
3. When entering letters instead of numbers (for both month and/or day of the month)

We would prioritize the defects in this order.

(d) *Use the template below to write an accident report about the defects / incidents you have discovered.*

Test incident report 1

Entered value 21 as day and 12 as month, received "invalid date".
Input: December 21^{st}.
expected results: Sagittarius.
Actual results: Invalid date.
Anomalies: None
Date and time: March 28^{th} 2017 13:37
Procedure step: Entered dates and clicked submit.
Environment: Mac OSX SIERRA 10.12.3
Attempts to repeat: 5
Testers and observers: Rune, David and Hans Henrik.

Test incident report 2

Entered value 31 as day and 6 as month, returned a horoscope (cancer) even though the month only has 30 days.
Input: June 31^{st}.
expected results: Invalid date
Actual results: Cancer
Anomalies: None
Date and time: March 27^{th} 2017 13:37
Procedure step: Entered dates and clicked submit.
Environment: Mac OSX SIERRA 10.12.3
Attempts to repeat: 7
Testers and observers: Rune, David and Hans Henrik.

Test incident report 3

Entered a letter as day and month, received an non-descriptive error.
Input: EAÅKFPOEJFAF as day and apojfeafe as month .
expected results: Invalid date
Actual results: Her burde det ha kommet en informativ melding om hva brukeren har gjort feil, men det gjør det ikke!!!!

Anomalies: None
Date and time: March 29^{th} 2017 13:37
Procedure step: Entered dates and clicked submit.
Environment: Mac OSX SIERRA 10.12.3
Attempts to repeat: 2
Testers and observers: Rune, David and Hans Henrik.

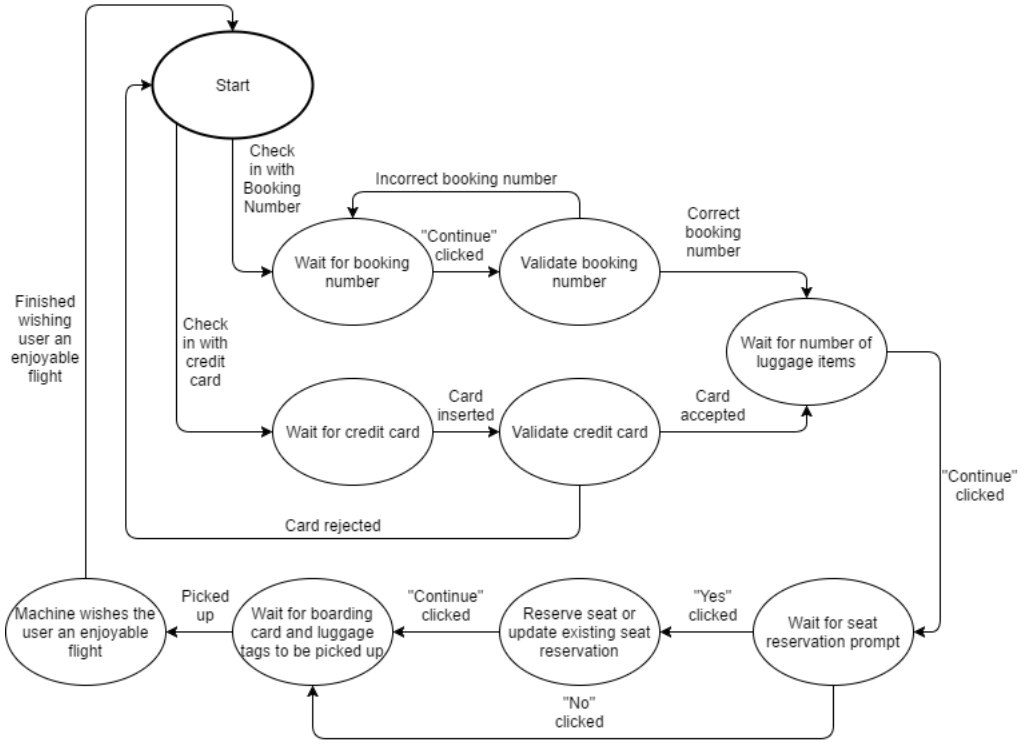
6 Exercise 6:

Tabell 1: Exercise 6

Conditions	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**
Rush hour	T	T	F	F
Electric vehicle	T	F	T	F
Actions/outcome				
Amount to pay	30	100	0	50

7 Exercise 7:

(a) *Draw a state transition diagram that shows how the machine works*



To save ourself from having a clustered picture, we did not add that if you press 'cancel' in any of the states, it would bounce back to the 'start'-state

(b) *Set up a state transition table showing every legal and illegal (impossible) state*

Tabell 2: 7b

	Valid booking number	**Invalid booking number**	**Valid credit card**	**Invalid credit card**	**Cancel**
S1) Start state	S1.A	S1.A	S1.B	S1.B	S1
S1.A) Wait for booking number	S2	S1.A	–	–	S1
S1.B) Wait for credit card number	–	–	S2	S1.B	S1
S2) Wait for num of luggage items	S3 (Wait for seat prompt)	–	S3 (Wait for seat prompt)	–	S1

Tabell 3: 7b

	Yes(seat)	**No(seat)**	**Cancel**
S3) Wait for seat prompt	S4	S5	S1
S4) Reserve or update seat	S5	–	S1
S5) Wait for boarding card	S6	S6	S1
S6) Good flight	S1	S1	S1

(c) *How do you measure statement coverage?*

- To measure statement coverage, you divide the number of statements exercised by the total number of statements, then multiply the answer by 100% to get it in percentage.
  How do you measure transition coverage?
- To measure transition coverage, you can do something similar. Divide the number of transitions exercised by the total number of transitions,

then multiply the answer by 100%.

(d) 1.(*start*) $\rightarrow$ *a*.(booking number)$\rightarrow$ *I*.(validate booking)$\rightarrow$ *III*. (incorrect booking)$\rightarrow$ *a*. $\rightarrow$ *I*. $\rightarrow$ *II*.(correct booking) $\rightarrow$ 2.(luggage items) $\rightarrow$ 3.(seat reservation) $\rightarrow$ *a*.(confirm seat reservation) $\rightarrow$ 4.(change/choose seat) $\rightarrow$ 5.(print card and tag) $\rightarrow$ 6.(best wishes) $\rightarrow$ 1. $\rightarrow$ *b*.(credit card) $\rightarrow$ *I*.(validate card) $\rightarrow$ *III*.(rejected) $\rightarrow$ 1. $\rightarrow$ *b*. $\rightarrow$ *I*. $\rightarrow$ *II* $\rightarrow$ (*accepted*) $\rightarrow$ 2. $\rightarrow$ 3 $\rightarrow$ *b*.(seat reservation not confirmed). State coverage = 100% Transition coverage = 100%

It should be impossible to reach 100% transition coverage without getting 100% state coverage, as there shouldn't be more states than there is transitions. If there is more states than transitions, then one or more states aren't connected to the others.

8 Exercise 8:

Our main focus will be testing that you cannot proceed without a valid card/booking number.

(a) Our main focus will be testing that you cannot proceed without a valid card/booking number.

(b) Tabel on next page.

Tabell 4: 8b

Main success scenario S = System A = Actor	Step	Description
	1	A) Choose credit card
	2	S) Wait for credit card
	3	A) Insert credit card
	4	S)Validates card
	5	S) Approves card
	6	A) Enters luggage items and clicks continue
	7	A) Clicks Yes to update seat reservation
	8	A) Updates seat reservation
	9	S) Prints out boarding card and luggage tag
	10	A) Picks up card and tag
	11	S) Wishes you a good flight
	12	A) Smiles and cries at the same time
	Exertions	
	4A	S)Invalid card, spits out card, shows message 'invalid card', goes to step 1
	7A	A) Clicks no to update seat.
		S) Goes to step 9.