Topic 6

Testing and Usability

Contents

Learning Objectives

- *Appreciation of the role of quality assurance departments*
- *Appreciation of the importance of system testing*
- *Awareness of the sorts of things that can and should be tested*
- *Familiarity with the different types and levels of testing possible*
- *Appreciation of the importance of usability issues*
- *Awareness of some key concepts in user interface design*

God doesn't play dice with the universe. (Albert Einstein)

6.1 Making the system usable

You should have read Articles 6, 7 and 8 of "The Case of the Killer Robot" before starting on this topic. They describe some pretty appalling practices at Silicon Techtronics Inc. The resulting Robbie CX30 system was playing dice with its users' safety; it had not been properly tested and very little account had been taken of the users in its design.

Poor testing and lack of regard to usability are the two most serious charges that can be brought against a system. We shall now look into these two aspects of system development. We shall treat the former in a bit more detail than the latter because you should already be acquainted with the basics of usability from assigned topics 5 and 6 which were researched and/or discussed in your tutorial group.

6.2 Testing

We have already met the notions of **unit testing** and **integration testing** in some of the life cycles described in the previous topic. Unit testing, you will recall, involves the testing of individual modules in isolation from each other to ensure that they are correct and integration testing involves testing that the modules communicate with each other correctly and achieve the desired overall goals correctly.

Two other ideas are useful to bear in mind. These are **black box testing** and **white box testing** (Sommerville 2000). Black box testing considers only the inputs and outputs to a system. Its inner details are not scrutinised during such testing, the system is treated as a black box into which one can't see. White box testing, on the other hand, pays considerable attention to the internal details of a system.

The correct functioning of a system can be assured using a number of different techniques:

1. Quality Assurance (QA) procedures and "House" styles
2. Verification and validation
3. Static and dynamic testing

The conscientious developer will often use more than one of these.

Quality assurance procedures and "House" styles

Large companies will generally have Quality Assurance (QA) departments whose functions are to prescribe procedures that should maintain quality levels and to provide a testing regime which is undertaken independently of the development team.

The QA department will lay down "House" styles which are ways of doing things that must be adhered to throughout the company. These styles will cover everything from the way documentation is formatted to the style of the comments required in program

code. Metrics will be employed by the QA department wherever possible to quantify those aspects of quality which are amenable to such things, such as the proportion of a piece of code which is devoted to comments.

Most companies will also standardise on one or more of the methodologies for analysis, design and development. International standards, such as ISO 9000 for Total Quality Management, should inform the QA procedures employed.

Verification & Validation

These two concepts are exceedingly useful and it is important to differentiate between them properly.

Verification is the determination of whether a product satisfies the initial conditions. I.e. whether it does the job it *claims* to do. This is an objective test which could make use of formal methods such as proofs of correctness

Validation is the determination of whether a product satisfies the specified requirements. I.e. whether it does the job it was *supposed* to do. This can be a more subjective judgement than verification.

Static and dynamic testing

Static testing involves inspection and analysis of the program code and its supporting documentation. This can be undertaken by computer-based checking in some areas. Syntax checking can be performed by a compiler for instance but identifying semantic errors is less readily achieved automatically. The checking of logic can be assisted by computer-based methods as long as a formal specification is available.

Dynamic testing involves running the program to ensure that the input-output mappings are correct. This can rarely be done in an exhaustive fashion because of the number of possible combinations but equivalence partitioning and boundary points can reduce the work involved.

Equivalence partitioning is the process of identifying classes of input data and classes of output data which have common properties. For example, consider a program which is required to perform an action A if an input lies between 1 and 10 inclusive and an action B otherwise. Three input equivalence classes can be created, one for all inputs less than 1, one for all inputs greater than 10 and one for the inputs between 1 and 10 inclusive. Sample inputs from each class can then be tested rather than testing all of the possibilities

Clearly the values 1 and 10 themselves, along with 0 and 11 are particularly critical. They are the extremes of the three ordered sets used to define the partition classes. They are boundary points and should definitely be tested to make sure that the transitions from action A to action B and vice versa are, indeed, occurring at the correct places.

A further dynamic testing technique is **structural testing**. This requires that each branch of the code is executed at least once during testing.

Finally, **acceptance testing** is essential. This is testing the system in the environment in which it will operate.

How many of these tests appear to have been used in the Robbie CX30 project?

6.3 Usability

Probably the most succinct guidance you can receive on usability is the **Eight Golden Rules** (Shneiderman 1998):

1. Strive for consistency
2. Enable frequent users to use shortcuts
3. Offer informative feedback
4. Design dialogues to yield closure (the user should know when it has ended)
5. Offer simple error handling
6. Permit easy reversal of actions
7. Support internal locus of control (the user should feel in control)
8. Reduce short-term memory load (do not overload user)

These rules ought to be fairly self-explanatory but you should also have received some explanation of them in your tutorial group whilst you were working on Assignment 1.

Two further ideas which can be very effective are **metaphors** and **mental models** (Preece et al. 1994).

A metaphor is a figure of speech in which the name of one thing (or some kind of representation of it) is applied to another thing. Probably the most famous metaphor found in user interfaces is the icon of a rubbish bin onto which files can be dragged and dropped in order to delete them. Without even thinking we know, from our everyday experience of real rubbish bins, just what that icon means. Metaphors do need to be obvious though. Straining to create metaphors is pointless. Brilliant flashes of inspiration lie behind good metaphors and rarely hours of hard work.

Mental models are models of the users themselves which are held in the system. They are generally built up over the course of many interactions with a user. Good mental models can help in the prediction of a user's needs and/or goals. Many benefits can result from this ranging from providing an appropriate level of help through to starting tasks up in advance of the user actually requesting them.

6.4 Summary and Assessment

At this stage you should be able to

- Outline the role of quality assurance departments
- Explain the importance of system testing
- Identify the sorts of things that can and should be tested
- Explain the different types and levels of testing possible

- Discuss the importance of usability issues
- Describe some key concepts in user interface design

End of topic test

5 min

Q1: One of the most serious charges that can be brought against a system is

a) Inefficient coding
b) Platform dependence
c) Poor testing
d) Too expensive

Q2: Which of the following was NOT cited as a means of assuring correct functioning ?

a) Carefulness
b) QA procedures
c) Static and dynamic testing
d) Verification and validation

Q3: ”House” styles do NOT cover

a) Code comments
b) Documentation formats
c) Dress sense
d) Metrics

Q4: A test of whether a system satisfies its requirements is

a) Black box
b) Validation
c) Verification
d) White box

Q5: Static testing CANNOT be used to check

a) Arithmetic
b) Logic
c) Semantics
d) Syntax

Q6: Equivalence partitioning is used in testing techniques which are

a) Black box
b) Static
c) Structural
d) White box

Q7: Which of the following was NOT one of Shneiderman’s 8 golden rules?

a) Enable frequent users to use shortcuts
b) Offer informative feedback
c) Offer customisable desktops
d) Reduce short-term memory load

Q8: A metaphor is

a) A figure of speech
b) An icon
c) A name
d) A rubbish bin

Q9: Metaphors are the result of

a) Artificial intelligence
b) Hard work
c) Inspiration
d) Plagiarism

Q10: Mental models CANNOT assist in

a) Modelling users
b) Providing help
c) Starting tasks
d) Visualisation

6.5 Assigned task

1. **Submit Assignment 2** if you have not already done so.
2. Read ”The Case of the Killer Robot” Article 9 (Epstein 1997 or Taylor 2002) before embarking on Topic 7.
3. Each member of your tutorial group is to research a *different* topic from the list below. They all relate to historical computing devices and this chronology will be brought up to the present day in Topic 8. If you are the first tutee on the group list then you are to research the first topic; if the second on the list then the second topic; and so on. You should prepare a **presentation** on your assigned topic in time for delivery at your next tutorial. The presentations will be spread over the last 2 tutorials in order to give each of you about **7 minutes** for your presentation followed by questions from the rest of your tutorial group. This presentation is **Assignment 3**. It will be assessed by your tutor and the mark will account for 33% of your final mark in the Praxis Unit.

Assigned Topics 19 - 27	
Napier’s Bones	*1st Tutee*
Pascaline	*2nd Tutee*
Multiplier Wheel	*3rd Tutee*
Difference & Analytical Engines	*4th Tutee*
Tabulating Machine	*5th Tutee*
Z1-Z4	*6th Tutee*
Colossus	*7th Tutee*
ENIAC/EDVAC	*8th Tutee*
EDSAC	*9th Tutee*

6.6 References

Epstein, R.G., 1997, *The Case of the Killer Robot*. John Wiley & Son.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S. & Carey, T., 1994, *Human-Computer Interaction*. Addison Wesley.

Shneiderman, B., 1998, *Designing the User Interface, 3rd edition*. Addison Wesley.

Sommerville, I., 2000, *Software Engineering, 6th edition.* Addison Wesley.

Taylor, N.K., 2002, The Killer Robot [online]. Heriot-Watt University (MACS), 16th December 2002 [cited 7th July 2003]. SHTML. Available from:

http://www.macs.hw.ac.uk/~nkt/praxis/epstein/index.sht

Answers to questions and activities

6 Testing and Usability

End of topic test (page 5)

Q1: c) Poor testing

Q2: a) Carefulness

Q3: c) Dress sense

Q4: b) Validation

Q5: a) Arithmetic

Q6: a) Black box

Q7: c) Offer customisable desktops

Q8: a) A figure of speech

Q9: c) Inspiration

Q10: d) Visualisation