

---

## Software Quality Assurance: SOFT3302

### Tutorial – Week 2

---

#### Objectives

By the end of this tutorial you should have an appreciation of whether or not you have been doing enough testing when developing software. You should start to gain an appreciation of the complexity of even small software systems. You should develop a desire to incorporate testing into your normal development processes.

#### Pework

Ensure you have read the definitions from Binder provided.

#### Labwork

1. Form small groups. Discuss, in the light of other assessment tasks you have been given while at University, the rôle of testing in your assessment. Compare how much time you have devoted to testing in the past compared with, say, the 50% breakdown mentioned in Brooks. Consider also at what point testing takes place in your development cycles.

**This is an important exercise not for any particular number which may be gained but because it should readily make apparent any underestimations concerning the importance of testing. It is to be expected that most programmers will have spent too little time on testing and have relegated testing to a final phase ‘polish’ step which both belittles testing and overestimates almost everyone’s programming ability.**

2. Vendors provide little guarantee when releasing software? Would you guarantee your software? If not, why not? What would stop you?

**Building software is something that occurs usually at the limits of current complexity. No one person comprehends the entire system and software systems have a myriad of states. It’s impossible to test all these states. The unknown saps confidence.**

**Anyone who says ‘yes’ to this needs to be careful. Such a goal is laudable but even Microsoft with its deep pockets doesn’t guaranty correct behaviour of its software (which is certainly wise in its case).**

3. Consider the different types of testing covered in the first lecture. Again in small groups consider which of these different types of testing would have been/would be appropriate when developing the solutions to your assessment tasks. If any are not appropriate, say why not.

**Clearly unit, integration and system testing are appropriate. User acceptance testing is the marking process! (There is usually only one shot at this.) Performance and load testing are probably not relevant in most cases (unless explicitly asked for) although I have been known, as part of marking suites, to stress test a programme which has been developed under a specification which doesn’t specify hard limits on things, *e.g.* test an input size of 100 000 records where a student might have unwarrantedly assumed 10 or 100 in order to uncover an  $O(n^2)$  algorithm.**

**Security testing is probably a non-issue as well. Backup testing may be relevant. Students don’t have access to the School’s backup tapes but are probably backing**

things up on CD-R or (worse) floppies and they should be aware that these don't last. Since there is no production of a product *per se* the relevance of alpha and beta testing is hard to see.

Regression testing would be appropriate and in group projects practices such as design and code reviews and pair-programming would help.

4. Consider the statement 'A tester is not a failed programmer.' Does this fit with your understanding of a tester?

**It shouldn't. Testing requires the same (if not more) skills. Ask if anyone would be happy to fly knowing an aeroplane was tested by someone without mechanical aptitude and then ask, 'What if the plane's software was tested by someone without programming aptitude?'**

5. Do you think testing will play a more central (and earlier) role in your development processes in the future?

**It had better! 😊**