


Software Engineering

Quality Management

Yvonne Dittrich
IT-University in Copenhagen
Design and Use of IT


© Dittrich February 2005 1



this lecture

- what is quality
- project related quality management
- organization wide project management
- reviews
- testing

© Dittrich February 2005 2



why do we want quality?

- we want to do a good piece of software
- we are tired of trial and error
- it is boring to debug.
- we want happy customers and users
- it is saving time and money

© Dittrich February 2005 3

The road to wisdom?
Well, its plain
and simple to express:
Err
and err
and err again
but less
and less
and less.

© Piet Hein, cited after Don Knuth

© Dittrich February 2005 4


quality or qualities

- quality in use
- product quality
- process quality

© Dittrich February 2005 5

quality management within a project

- defining quality goals for the project (e.g. part of requirements specification)
- application of defined methods and tools
- carrying out reviews
- testing systematically
- measurements - on the way and at the end
- making sure that the feedback is used for improvements
- systematically reporting quality status



© Dittrich February 2005 6

reviews

Formal technical review (FTR) is an umbrella term for review methods involving a structured encounter where a group of technical personnel analyzes an artifact in order to improve both the quality of the product and the review process. © Johnson

phases

- planning
- information meeting
- preparation
- review meeting
- adjusting
- closing

roles

- review leader
- recorder
- producer
- reviewer

© Dittrich February 2005

7

quality management across projects



- do we develop good software?
do we develop software in a good way?
 - how do we develop software?
 - is the software we develop of a satisfying quality?
- can we develop better software?
can we develop software in a better way?
 - what is 'better'? which of the qualities do you want to improve?
 - what are the possibilities for improvement

© Dittrich February 2005

8

PLAN – DO – CHECK – ACT

- PLAN for changes to bring about improvements
- DO changes on a small scale first to trial them
- CHECK to see if the changes are working and to investigate selected processes
- ACT to get the greatest benefit from changes



© Dittrich February 2005

9

Quality Management

- quality planning
 - specification of required quality
 - decision on the development processes
- quality control
 - measurements and recording of product qualities
 - evaluation and design of correcting actions
- quality assurance
 - measurements and recording of the process qualities
 - evaluation and planning of correcting actions

© Dittrich February 2005

10

Testing is meant to

- control the quality of the program
does the thing really behave as it is meant to?
- increase the quality
find and correct bugs
- be part of normal software development
when programming we make assumptions,
e.g. about the bounds of a loop.
these assumptions might be wrong.

© Dittrich February 2005

11

what is the difference between a test and a trial run?

- every developer is trying a trial run as soon as something can be executed
- users might do a trial run
- in a trial run, you see whether the class/module/program behaves as you expected it.
- you do not test all possible input combinations
- you do not test all paths in the program
- you do not test all error possibilities and how to recover from them

© Dittrich February 2005

12

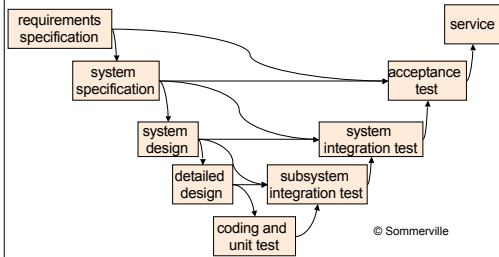
Testing is the process of exercising a program with the specific intent of finding errors prior to the end-user

testing requires:

- a systematically developed set of test cases
- that the result of the test for a given input is computed before running the test!

almost all inexperienced developers ignore the latter!

The V-model: looking from a quality perspective



systematic testing, test case definition

- **systematic:** test the whole system according to some strategy.
- **statistic:** concentrate the tests at the most used inputs. (needs a statistical model of the use of the system)
- **crash test:** test extreme input to check that the system does not crash. should be used to complement to statistic testing.

systematic testing
black box - white box

- black box testing
treat the system as a black box. generate the test cases from the specification
 - often done by someone not involved in the development
- white box testing
look at how the program is constructed. generate the test cases from the structure of the program
 - often done by the developers themselves

Black box testing example

equivalence testing

- partition the input space in equivalence classes, use one test case from each class (and possibly also one from the border)
- e.g. $\text{abs}(x)$
the two classes of the input are positive and negative numbers. test with 1, -1, 0
- the problem often is how to find this classes.

white box test

- path testing. cover all possible paths through the code. (not possible in practice)
- code coverage: test all the code. this is the most commonly used systematic test method.
 - a problem is exceptions in the code that sometimes are impossible to test without changing the code.
 - loops must be treated in a special way
 - 100% coverage is often not possible in practice.

?

when to use black box testing
when to use white box testing

© Dittrich February 2005 19

integration test

- putting the system together bitwise
- testing the interaction of different units
- in order to locate errors in the behavior that only get visible in the interaction

© Dittrich February 2005 20

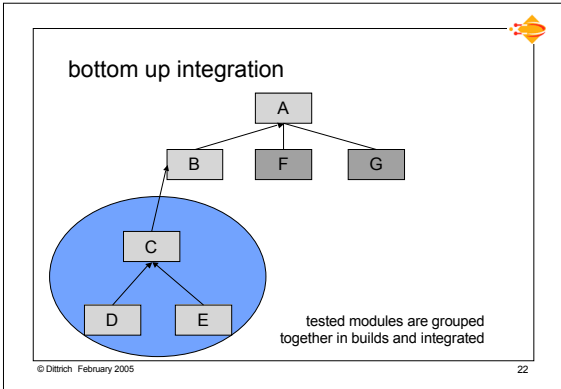
top down integration

the central node is tested first with the help of stups

stups are exchanged by the unit-tested modules one by one, according to the depth first strategy

you have to repeat all the previous tests when you exchange a stub.

© Dittrich February 2005 21



- specificities of object oriented testing**
- unit tests are normally tests of each class (instead testing each procedure)
 - testing of independent classes is often not easily possible, when the classes they use are not yet there, or not yet tested.
 - inheritance complicates testing
 - side effects and errors due to interaction of different classes are more difficult to find
- © Dittrich February 2005 23

- remedies**
- class-test based on the behavioral model of each object
 - a well partitioned design with clear interfaces pays off
 - integration test based on
 - use relations (bottom up)
 - on control flow (integration according to the sequence diagrams)
 - regression test becomes even more important: when integrating a new class you have to make sure that all the old test cases still work
 - use the use cases to define system test and acceptance test.
- do not underestimate code reviews
- © Dittrich February 2005 24

regression testing!!!

- small changes in the software may affect some seemingly unconnected parts of the program
- the solution is to always retest the whole program
- this type of testing is called regression testing. the goal is that the new version should at least not have more errors than the last version.

references

- P. Hein, cited in D. Knuth 'Learning from errors.' In: C. Floyd, H. Zuellighoven, R. Budde, R. Keil Slawik Software Development and Reality Construction. Springer Verlag 1992
- P. M. Johnson 'Reengineering Inspection.' Communication of the ACM, Volume 41, 1998, pp. 49 - 52.
- Y. Dittrich, O. Lindeberg 'How Use-oriented Development Can Take Place' Information and Software Technology 46 (2004) 603-617.
- Ian Sommerville Software Engineering Addison Wesley 2004
- R. Pressman Software Engineering- A practitioner's approach. 6th edition 2005.
