EEE4120F

High Performance Embedded Systems

Lecture 24
[Software] Quality Assurance

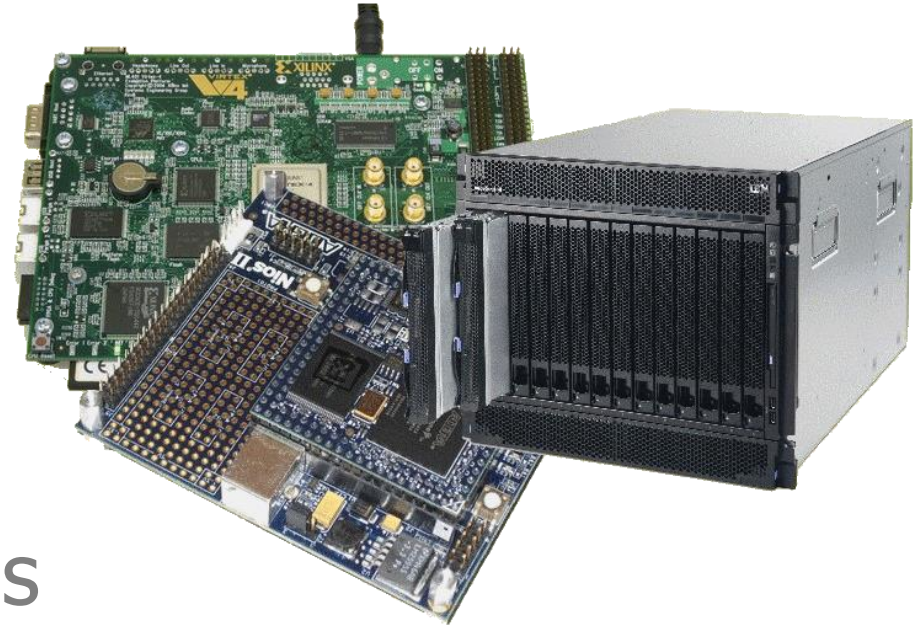
Kai-Zen Aspect parts optional

改善
ソフトウェア

Lecturer:
Simon Winberg

Lecture Overview

- What is quality?
- Software quality
- Software quality assurance (SQA)
- Software quality systems
- Consequences of bad SQA
- Evolutionary model of SQA in orgs
- Kaizen of software
- Formal Specifications



What is Quality?

- General definition of quality:
  - It is a products 'fitness of purpose'
  - A product is good quality if it does exactly what the users wants it to do!
- 'Fitness of purpose'
  - The system satisfies the requirements as specified in requirements document (agreed contract)

requirements

Consider this

A computer system (e.g. accounting program) that

- is functionally correct (e.g. accurately records debits/credits)
- meets all the requirements as laid out in the contract …

but one that

- is sometimes incredibly slow (takes 5 minutes or more to complete a transaction)
- the user interface is very difficult to use takes days for a new user to learn how to use it
- occasionally hangs for no reason (but doesn't loose/corrupt data) …

Is that Software Quality? NO!

Brief Brainstorm

Software and product quality is not just about satisfying requirements, it is also about…

Work with a buddy and decide at least 5 things besides satisfying requirements that a quality system – or your YODA project – should provide.

reflections on your

Brief Brainstorm

The answer to this question may also depend on *who* you ask…

Software and product quality is not just about satisfying product requirements and clients.

Not all the roles may have the same sense of what the top priorities are… but generally, regards a good product, there may be many dimensions to that.

Asked:

Software and product quality is **not just about satisfying requirements**, it is also about…

Work with a buddy and decide at least 5 things besides satisfying requirements that a quality system – or your YODA project – should provide.

Software (and generally product) Quality

- Software quality is not just about satisfying requirements, it is also about:
  - Correctness and
  - "The 7 desirable 'ilities" :
    1. Usability
    2. Maintainability (incl. documentation)
    3. Scalability / Extensibility
    4. Reliability
    5. Reusability
    6. Securability (more formally 'security')
    7. Portability

You should know what these 7 ilities are, read up if you don't

Software Quality Assurance (SQA)

- Software quality assurance is the:
  - Guidance/process for recognizing, defining, analysing, and improving the software production process.
  - It is achieved through a software quality system

Software Quality System (SQS)

- Organizational responsibility
- Managerial structures
- Individual SQA Responsibilities

Software Quality System Activities

- Auditing software projects
- Establishing:
  - Standards
  - Procedures
  - Guidelines, etc.

Often quite specific to a particular organization but often based around published recommended practices

- Producing reports for high-level management
- Continuously review & refining the organization's quality system

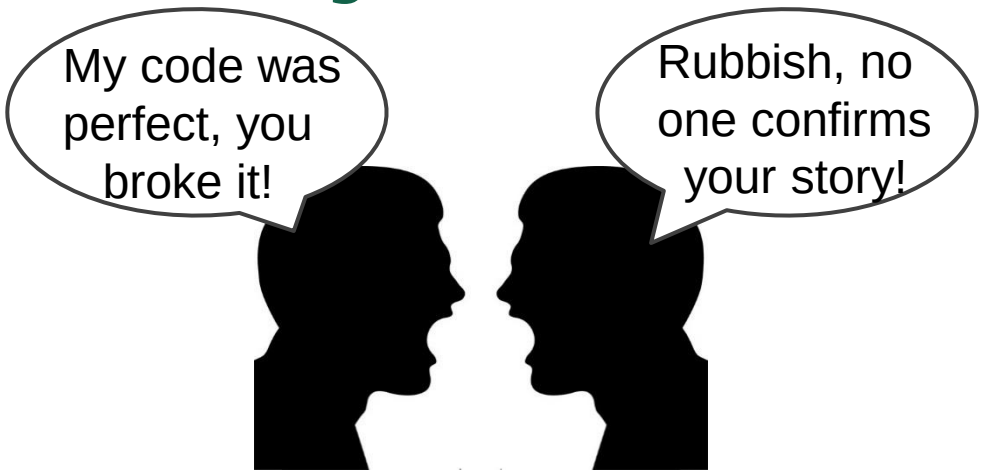
Software Quality System Activities

- Effective quality system are well documented
- Without a properly documented quality system, application of quality procedures become ad-hoc, results in large variations in the quality of the products delivered*.

* Sadly this is commonly what happens to most hons and postgrad projects... because there tends not to be enough time to learn or apply quality practices; imagine how impressive out projects could become if there was sufficient time for this!!

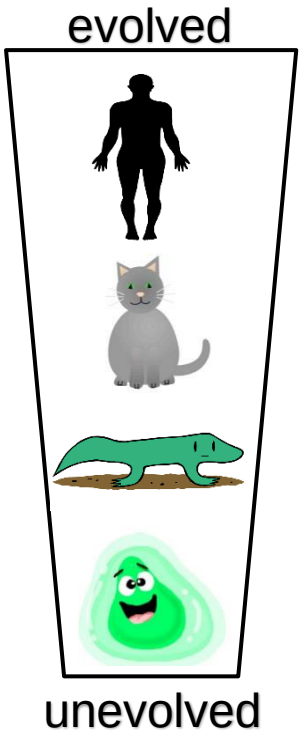
Bad/Inconsistent SQA → Bad Attitudes and Behaviour

- An undocumented/poor quality system:
  - Sends clear messages to the staff about the organization's attitude of quality assurance.
  - Can contribute towards project failure/delays
  - Can make maintenance impossible/expensive
  - Can leads to disagreements between staff



Evolution of Quality Assurance

- The SQA of a particular organization can usually be placed into the following continuum of Software Quality Systems



4. Total Quality Management (TQM)
3. Quality Assurance (QA)
2. Quality Control (QC)
1. Inspection

It's up to you to decide where programmers are the happiest!

The Levels

- Inspection
  - Check for defects and eliminate them
- Quality Control
  - Not just detect & eliminate defects…
  - Also determine causes behind defects
- Quality Assurance
  - Above + this basic premise:
    - If quality processes are good, and followed rigorously, the products are bound to be good!
  - Includes guidance for recognizing, defining, analysing, and improving the production process

The Levels

- Total Quality Management (TQM)
  - Continuous improvements to the quality process through measurement and refinement.
  - "Continuous Process Improvement"

There is a Japanize term for this … and people who strive toward this ideal in their work are considered something like Samurai Warriors (or like an 'Iron Chef') … the work is …

改善 Kaizen

OPTIONAL supplementary reading; a useful concept

改 KAI=Change

善 ZEN=Good

改善 KAIZEN (Continual Improvement)

Kaizen :
The **method of continuous incremental improvements** is an originally Japanese management concept for incremental (gradual, continuous) change towards improvement. Kaizen is actually **a way of life philosophy**, assuming that every aspect of our life deserves to be constantly improved.

Samurai warriors strove towards the kaizen of the warrior, that is to say they aimed to become perfection in all areas of combat, from the appearance of their armour, the way they moved, to the mental preparation, and physical preparation for strategy, defensive tactics, striking skill, etc.

Toyota the huge automotive giant proudly strives towards the kaizen of automobile manufacture

What would it take to strive towards the kaizen of HPEC system development?... Like a samurai, it takes much time & practice!

OPTIONAL
supplementary reading

Formal Specification

Towards precision of implementation meeting the requirements

Formal Specification

OPTIONAL supplementary reading

- In computer science Formal Specifications (FS) =
  - Mathematically based techniques the purpose of which is to help the implementation of a systems and its software meet its requirements.
- Formal specifications are used to
  - Describe a system
  - Analyse its behaviour
  - Aid the design by verifying key properties of interest through rigorous and effective reasoning tools.
- These specifications are formal in that they have a syntax and semantics that can be used to infer useful information and formulate proofs.

Formal Specification

OPTIONAL supplementary reading

- *Why FS*
  - Computer systems have become increasingly more powerful and integrated into society over the years, have impacted society and the environment to an ever greater extent – people have become more dependent on their computer systems
  - Due to their increasing complexity and ubiquity, they have also become more difficult to analyse, both their workings and their impact on other systems.
  - For these reasons methods are needed to more decisively, if not undisputedly, reason and know that these systems will do what they are supposed to do.
  - Formal specifications are one such way to achieve this in software engineering, particularly for applications where reliability and predictability is paramount, and regimes of testing may not be adequate – such as in medical systems and rocket control modules where you want an unequivocal proof that the system satisfy its requirements.

Formal Specifications

OPTIONAL supplementary reading

- To do FS properly, we would probably need a whole course on it. So in this course the concept is just introduced so that you know about it, as FS is indeed highly relevant to HPEC where these types of systems are often used in safety-critical applications.
- One thing I do want to delve into is the concepts of pre- and postconditions…

Preconditions and Postconditions

OPTIONAL supplementary reading

- Often a programmer must needs to describe exactly what a function needs to accomplish, without any indication of how the function does its work.
- i.e. you need to describe what the result of the function will be, and conditions in which it is expected to work, without explaining in details (of a particular programming language) what it does.

Preconditions and Postconditions

OPTIONAL supplementary reading

- One method to explain the operation of a function without explaining its implementation is using
  - Preconditions
    - A statement indicates what must be true before the function is called.
    - Programmer needs to ensure the **precondition is valid** when function is called
  - Postconditions
    - A statement indicates what will be true when the function finishes its work.

Preconditions and Postconditions Example

OPTIONAL
supplementary reading

Consider this disp_sqrt function.

```
void disp_sqrt ( float x )
{
   // calculate the square root and print the result to the stdout
}
```

Preconditions and Postconditions Example

OPTIONAL
supplementary reading

Consider the sqrt function.

```
void disp_sqrt ( float x )
{
   // calculate the square root r and print this result to the stdout
}
```

We know that for negative x it should give imaginary numbers, but assume that use of imaginary numbers are not possible by the specification. So in that case, we can specify the post condition for the function: that it will only work for non-zero numbers…

Preconditions and Postconditions Example

OPTIONAL
supplementary reading

Consider the sqrt function.

We could add →

```
void disp_sqrt ( float x )
{
// PRECONDITION:   (x>=0)

    // calculate the square root r and print this result to the stdout
}
```

We know that for negative x it should give imaginary numbers, but assume that use of imaginary numbers are not possible by the specification. So in that case, we can specify the post condition for the function: that it will only work for non-zero numbers…

Preconditions and Postconditions Example

OPTIONAL
supplementary reading

Consider the sqrt function.

```
void disp_sqrt ( float x )
{
   // PRECONDITION:   (x>=0)

 // calculate the square root r and print this result to the stdout
}
```

What should happen after the function? It should come out with a result *r* such that r*r = x, and print this result to the screen…

Preconditions and Postconditions Example

OPTIONAL
supplementary reading

Consider the sqrt function.

```
void disp_sqrt ( float x )
{
   // PRECONDITION:   (x>=0)

  // calculate the square root r and print this result to the stdout

   // POSTCONDITION: (r * r = x)  & has send data to stdout
}
```

What should happen after the function? It should come out with a result *r* such that r*r = x, and print this result to the screen…

Pre- and post-conditions in C

OPTIONAL supplementary reading

- The **assert** function can be used to test conditions in a simple way

```
void disp_sqrt ( float x )
{
   assert(x>=0); // check precondition

 // calculate the square root r and print this result to the stdout

  assert(r*r==x); // check postcondition
}
```

How assert(*val*) works: if *val* is true, does nothing, if *val* is false then prints an error that indicates the module file name, the line number and what *val* is.
e.g.
Assertion violation: file mysqrt.c, line 12: x <= 0

Any Questions?

End of Slideshow

Disclaimers and copyright/licensing details

I have tried to follow the correct practices concerning copyright and licensing of material, particularly image sources that have been used in this presentation. I have put much effort into trying to make this material open access so that it can be of benefit to others in their teaching and learning practice. Any mistakes or omissions with regards to these issues I will correct when notified. To the best of my understanding the material in these slides can be shared according to the Creative Commons “Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)” license, and that is why I selected that license to apply to this presentation (it’s not because I particulate want my slides referenced but more to acknowledge the sources and generosity of others who have provided free material such as the images I have used).

Image sources:

– Wikipedia open commons

Businessman juggling - Pixabay

Samurai, Kaizen Japanese lettering – sources include flickr and openclipart.org