Software Project Management

Introduction

Why do Some Computer Systems Fail to Work as Expected

- We have all heard of computing systems that fail at enormous expense.
  - NHS Records System
  - Ariane 5 Rocket
  - The Heartbleed bug in OpenSSL
  - Bank IT systems.
- Smaller scale projects fail all the time, in a less visible manner. It is often said that 80% of computer systems fail in some way.
- Most modern software needs to be updated regularly to fix bugs.

Incorrect Code

- A bug in the code used to be a relatively rare cause of failure.
- The Heartbleed bug was a coding error, but inadequate management let it be added to OpenSSL.
- Most software seems to work well.
  - But needs regular updates.
- The most common form of failure is building the wrong system.
  - It is difficult to find out customer requirements.
  - This becomes more of a challenge the larger the system being built.

The Nature of Software

- Software consists of instructions on how to manipulate information.
- It is very flexible. There are a large number of different ways in which information can be processed.
- Contrast this with other aspects of engineering such as building a consumer device. The number of physical controls are limited (e.g. Microwave)

Software Costs

- Duplication of an existing piece of software is very cheap.
  - Contrast this with the mass production of a car.
- The cost of producing software is largely independent of the number of copies sold.
- The major expense in producing software is in the design and production of the first version, not the mass production.
- This makes it a high risk activity.
- The costs of making something are made up of:
  - Fixed costs, paid no matter how many copies sold
  - Costs per copy, the only way of earning money.

Software Creation

- Most of the work in the software industry is design, not manufacturing.
- This is labour intensive and is a craft activity.
- It is very difficult to automate the process of software creation.
  - Like automating other design activity.
  - It is possible to produce better tools that help the designer.

Software Complexity

- Most software is very complex
- It is difficult to understand exactly how it works.
- It is difficult to modify.
- It is relatively easy to create software that appears to work.
- It is hard to demonstrate that software does everything that it should do.
- The main way is by testing using different input values.
  - There are so many possible input values.
  - It is impossible to test all of them.
- Proving correctness by theorems works for small pieces of code.

Estimating Work Involved

- It is difficult to say in advance how much work it will take to build a particular piece of software.
  - But in many cases you will have to quote a price before you start work on the software.
  - When you know the least about how long it will take.
- The first 90% of the work takes the first 90% of the time.
  - The second 90% of the work takes the remaining 90% of the time.
- Fortunately, most customers realise that there will be uncertainty.
  - Quote your lowest price and increase it later.

Software Rot

- Software does not wear out, but . . .
- Bugs are found after it is released.
  - User experience.
  - Continued testing.
- The underlying operating system and supporting programs change.
- Corrections are made.
  - The consistency of the design deteriorates.
  - New defects are introduced.
- Eventually the software stops working.

Techniques for Creating Software

- It is possible to get small pieces software to work using almost any approach.
- As the size of software gets larger, most ways of developing software don’t scale up.
  - The software reaches an ‘almost working’ state, where it stays forever.
- It becomes impossible to understand and change.
- Software Project management and Software Engineering teach a process that scales up.

Time To Market

- In a competitive market
- The first company to get a new piece of software to market gets most of the sales.
  - The company with the ‘best’ software may not make any money if it is late to market.
- Software has to be ‘good enough’.
  - The ‘best’ is the enemy of the ‘good enough’
- Many bugs will still need to be fixed after a product is sold.
  - The patch cycle.

Types of Software

- Custom
  - Developed to solve a specific problem.
  - Air traffic control system.
- Generic
  - Developed to solve many different problems.
  - Word processor
- Embedded
  - Runs hardware devices.
  - Camera
  - Cars

Software Engineering

- Solving customers’ problems
- Systematic development
- Large high quality systems
- Within cost, time and other constraints.

Solving Customers’ Problems

- The customer will decide what needs to be done.
  - Avoid unnecessary features (gold plating).
  - Find out what the customers really want / need.
- Would it be more cost effective to buy existing software.
- Would it be more cost effective not to develop the software.
- It is a human activity and needs interaction with non-technical people.

Systematic Development

- Use well understood techniques.
  - Object oriented design with UML.
  - Scrum
  - Prince II
- Apply a disciplined approach.
- Continue developing software after it is delivered.
- Use a precise language to interact with non technical people.
  - Natural language is often ambiguous.

Large High Quality System

- A single highly skilled programmer can keep track of and understand about:
  - 50,000 lines of code written in an object oriented style.
  - 10,000 lines of code in a non-object oriented style.
- Many systems are larger than this and need a team of developers.
  - Need a consistent approach to documentation and coding style.

Teaching in University

- It is hard to teach this in university.
- You can do all of the exercises using a ‘bad’ technique because they are not large enough to need a good approach.
- Use our approach because it will work when you get a job and work on realistic systems.
  - You will have to adept to in-house ways of doing things.

Within Cost, Time and Other Constraints

- Someone will be paying your wages and needs to earn that money, and more, from what you do.
- It is not worth doing something if it won’t earn enough money.
- If someone else is cheaper or faster then they will make the money, rather than us.
- Create a budget and a plan.
- Monitor if you are sticking to the plan.
- Meet planned milestones.
- Check that the customer is satisfied.

Stakeholders

- Users
  - They will use the software to do more interesting things
- Customers
  - They pay for the software
  - They often employ users
- Developers
  - Create the software
- Managers
  - Keep the customer satisfied
  - Make the most money

External Software Quality

- Usabililty
  - Learnability, efficiency of use, error handling
- Efficiency
  - CPU, memory, disk space, network bandwidth
- Reliability
- Maintainability
- Reusability
  - Many different applications
- Can’t be excellent at all of these. Will have to make tradeoffs.

Internal Software Quality

- Not noticed by the customer!
- Effects Maintainability
- Amount of commenting in the code
  - More important as programs get bigger
  - The person trying to understand your code in a years time may well be you!
- Complexity of the code
  - Use the object oriented style to stop complexity spreading throughout the code.
  - Make individual code segments less complex.

Different Types of Software Engineering Projects

- Greenfield Project
  - Start from scratch
  - Not constrained by previous bad decisions
  - Don’t have to understand code written by others.
  - Common in university software courses.
  - Relatively rare in industry
- Evolutionary Project
  - Fix defects
  - Add features
  - Adapt to changing environments

Building a Framework

- Software reuse across projects will save money
  - Most projects are different, specific to individual customers
  - There will be common components across different projects.
- Creating a library of common components or a framework will pay off in the future
  - But they cost more now.
- It is an internal project
- Money comes from customer projects