



Software Development

Craft or Engineering?



Dr. Neil Roodyn

Dr. Neil Roodyn

- Developing software for too long!
- Founded 5 software companies
- Teaching and mentoring around the world since 1991
- Mission:
 - Increase the value of your Software Business

Dr. Neil Roodyn

Aims

- Review the discipline of software development
- Examine why software engineering exists
- Investigate the craftsmanship involved in developing software



Issues in Software Development

- Why do software projects fail?
 - And the majority of them still do!

Issues in Software Development

- Risks
 - Slipping deadlines
 - Cancellation
 - System goes out of date
 - Quality
 - Misunderstood business needs

The History of Software Development

- **The Pioneering Era (1955-1965)**
 - Machine rooms, development of high order languages, v. few software companies and no packaged software
- **The Stabilizing Era (1965-1980)**
 - Big O/S, large demand for programmers, structured programming, standards organisations got serious, a few software vendors emerged
- **The Micro Era (1980-1995)**
 - Massive drop in price of computing, GUI becomes prevalent, more tools than ever for development
- **The Internet Era (1996 - Present)**
 - Explosion in interconnectivity of computers, software must become internet aware, pace of change increases even more rapidly

Software Methodologies

- Chaotic
 - 1950 – Maths based
 - Code and fix
- Predictive
 - 1968 – Software Engineering
 - monumental methodologies
 - 1990 – Rapid Application Development
 - Lightweight
- Adaptive
 - 1997 – eXtreme Programming
 - Agile

Predictive

- Emphasis on planning before you build
 - Based on engineering disciplines such as civil or mechanical engineering
- Predictable schedule that can use people with lower skills
- Separate design from construction
- Design the software so that the construction can be straightforward once the planning is done

Planning before you build

- What does a software plan look like?
 - UML, Object Models, Class Diagrams etc..
- Does the design make the coding any easier?
 - Generally not, very hard to validate correctness
- Cost of design vs. coding
 - Civil engineering design cost is about 10% of total
 - Software engineering design is 50%+ of total cost
 - Construction is 15% of the total cost



So...

- In software:
 - construction is so cheap as to be nearly free
 - all the effort is design, and thus requires creative and talented people
- Creative processes are not easily planned, and so predictability may well be an impossible target.
- We should be very wary of the traditional engineering metaphor for building software.

Unpredictable Requirements

- “This project will never succeed the requirements keep changing” – anonymous developer
- I have never worked on a project with fixed requirements!
 - From day one to delivery
- Hardly a surprise that a method fails if it requires fixed requirements

Software Engineering

- 1968 NATO conference
 - *Software crisis*
 - Need for high quality, large software applications
 - Primarily for defence departments!

Software Engineering

- “...the application of a systematic, disciplined, quantifiable approach to development, operation, and maintenance of software; that is, the application of engineering to software.”

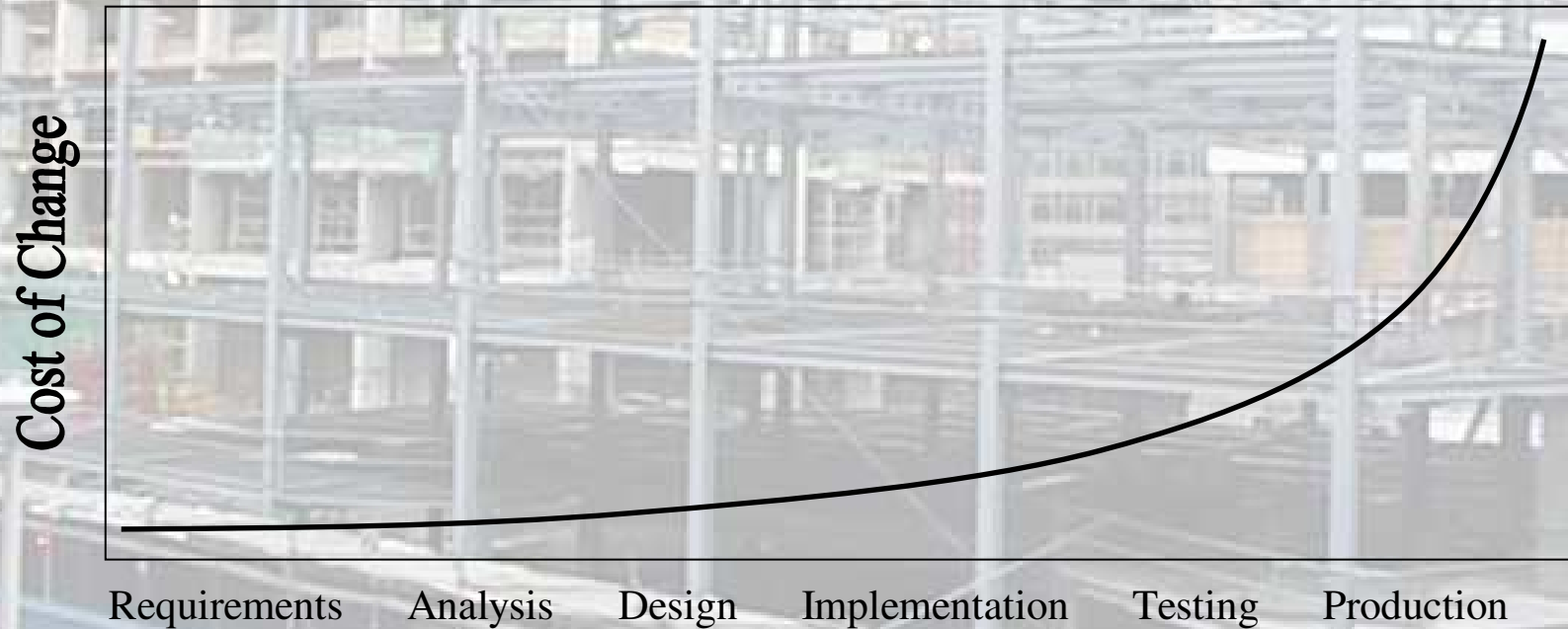
IEEE

Systems Engineering?

- These big projects are not purely software based
- Hardware was often developed specifically for the job
 - Programmers had to wait for the hardware
 - So they start the design process
- When the hardware is complete the software needs to be available ASAP
 - So the software guys get as much done up front as possible
 - Attempt to make coding automated so: more programmers = more code

Cost of Change

- Software Engineering
 - Exponential



Dr. Neil Roodyn

Suitable for...

- Large systems involving hardware
- Mission critical
 - Life support
 - Air craft control
 - Etc...
- Predictable core requirements

Issues

- Software Engineering
 - Treats all developers as equals
 - Almost as automatons
 - Ignores the pace of change in the business world
 - Believes that more people = more productivity
 - Focuses on the process of development not the actual problems being solved
 - Many developers merely pay lip service to the methods while not actually using them

The world has changed

- 1968 – Software was the cheapest part of the system being developed
- Now software is often the most expensive part of a system being developed
- The majority of the software development cost is the people costs
- Systems need to be developed in weeks (or months) not years

What if?

- I want a feature rich application
- On the shelves ASAP
 - I need to beat the competition to market
- I want to upgrade the application quarterly-yearly
- I need to change it as my business needs change
- It needs to be affordable

Adaptive

- Alternative approaches to the predictive model of software development
- Understand that the requirements will change
- Adapt to those changes
- Focus on people solving problems not processes

Cost of Change

- Needs to be lower
- So we can adapt to the changes

Cost of Change



Time

Dr. Neil Roodyn

Craftsmanship

- Places the craftsman at the centre of the software development
- Software development is not easy
 - A skilled craftsman is required
- Just knowing how to program a computer is not the same as being able to develop high quality software

Craftsmen

- A professional craftsmen
 - Dedicates their working life to getting better at the craft
 - Learns their skills through apprenticeship to a master craftsman
 - Aims for mastery of the craft
 - Teaches others through apprenticeship
 - Builds a reputation based on what they deliver
 - Not what exams they have passed

Working with the customer

- A Craftsman
 - Works closely with the customer to create what is required
 - Discusses the implications with the customer
 - Has a vested interest in ensure the customer is happy
 - Reputation is at stake

Software: Easy to Copy

- In the physical world
 - Craftsmanship is too expensive because each item has to be hand crafted
- In the digital world
 - The craftsman's work can be easily duplicated

Accountability

- A huge issue in my experience
 - Developers hide behind documents
 - Refuse to be accountable for their work
- Craftsmanship resolves this
 - A craftsman is proud of their work
 - Signs their name on their work

Building a tiered culture

- Exploit the difference between developers
 - productivity and experience
 - Do I consider myself a master craftsman?
 - I have over 20 years experience solving software problems
 - I have run 5 software companies
 - Been the development director for 3 other companies
 - Does my earning potential reflect this?

What makes a good developer?

- Portfolio
 - Like any other craft
 - A history of success stories
 - Some failures in the past
 - You need to 'earn your spurs'

Customer chooses

- Make a decision of quality vs. cost
 - This is not easy to do right now
- Customers build longer term relationships with craftsmen

Alignment of Interests

- With engineering the customer is often pitted against the development team
 - “We delivered what you wanted”
 - “You never said you were going to do that!”
- With craftsmanship the interests are aligned
 - The craftsman wants a happy customer
 - Does whatever is required to ensure this

Enjoy work

- Craftsman pursue their craft from a love of what they do
 - It is more than a job
- Choose your attitude

Apprenticeship

- Once you leave school/college you still have everything to learn
- The best way to learn is from someone who has done it before
- Encourages good practices to be passed on
- Instils lifelong learning
- Not teaching how to program!

What about eXtreme Programming?

- It's an engineering discipline?
 - You must have testable requirements
 - Aims for predictable high quality software
- It's also a craft
 - You must communicate ideas and discuss the system metaphor
 - You work closely with the customer
 - Pair programming works as apprenticeship

XP – best of both worlds

- Agile methods such as XP focus on:
 - The people doing the development
 - The quality of the software developed
 - Customer happiness
 - Lowering cost of change
 - Increasing use of best practices

Bibliography

- Robert L. Glass - "In the Beginning: Recollections of Software Pioneers"
- The New Methodology by Martin Fowler
- Rapid Development: Taming Wild Software Schedules by Steve C McConnell
- What is Software Design? by Jack W. Reeves
- eXtreme Programming Explained by Kent Beck
- Software Craftsmanship by Pete McBreen
- Software Engineering: A Practitioner's Approach by Roger S. Pressman
- The Mythical Man-Month by Fred Brooks