The Concepts Behind Software Design

Elena Punskaya, elena.punskaya@eng.cam.ac.uk

Course Objectives

- **Object-Oriented Software Design**
  - Understand the benefits of object-oriented analysis and design, its concepts and processes
  - Be familiar with formal design tools for object orientated design and analysis
  - Recognise and understand some frequently used design patterns
  - Be aware of the process involved in user interface design
- **Software Systems and Engineering**
  - Appreciate the basic design issues and concepts in distributed, real time and concurrent systems
  - Understand software development methodologies
  - Understand the main issues and processes necessary to achieve effective software product development
- **The course is not about becoming a Code Ninja**
  - so we are not going to learn programming in Scala, Ruby on Rails, Go!, Java or Thumb
  - but we might discuss them
- **It is neither about becoming a Project Management Guru**
  - so we are not going to make Gantt charts and milestones
  - but we could talk about priorities, teams and metrics

Additional Books

- Ian Sommerville, Software Engineering, Addison-Wesley
- Roger Pressman, Software Engineering: A Practitioner's Approach
- Fred Brooks, The Mythical Man Month, Addison-Wesley
- Martin Fowler, UML Distilled: A Brief Guide to the Standard Object Modeling Language, Addison-Wesley
- Andrew Hunt and David Thomas, The Pragmatic Programmer
- Don Norman, Design of Everyday Things
- Kent Beck with Cynthia Andres, Extreme Programming Explained: Embrace Change
- Donald Norman, Living with Complexity
- Jez Humble and David Farley, Continuous Delivery: Reliable Software Releases Through Build, Test and Deployment Automation

Definitions

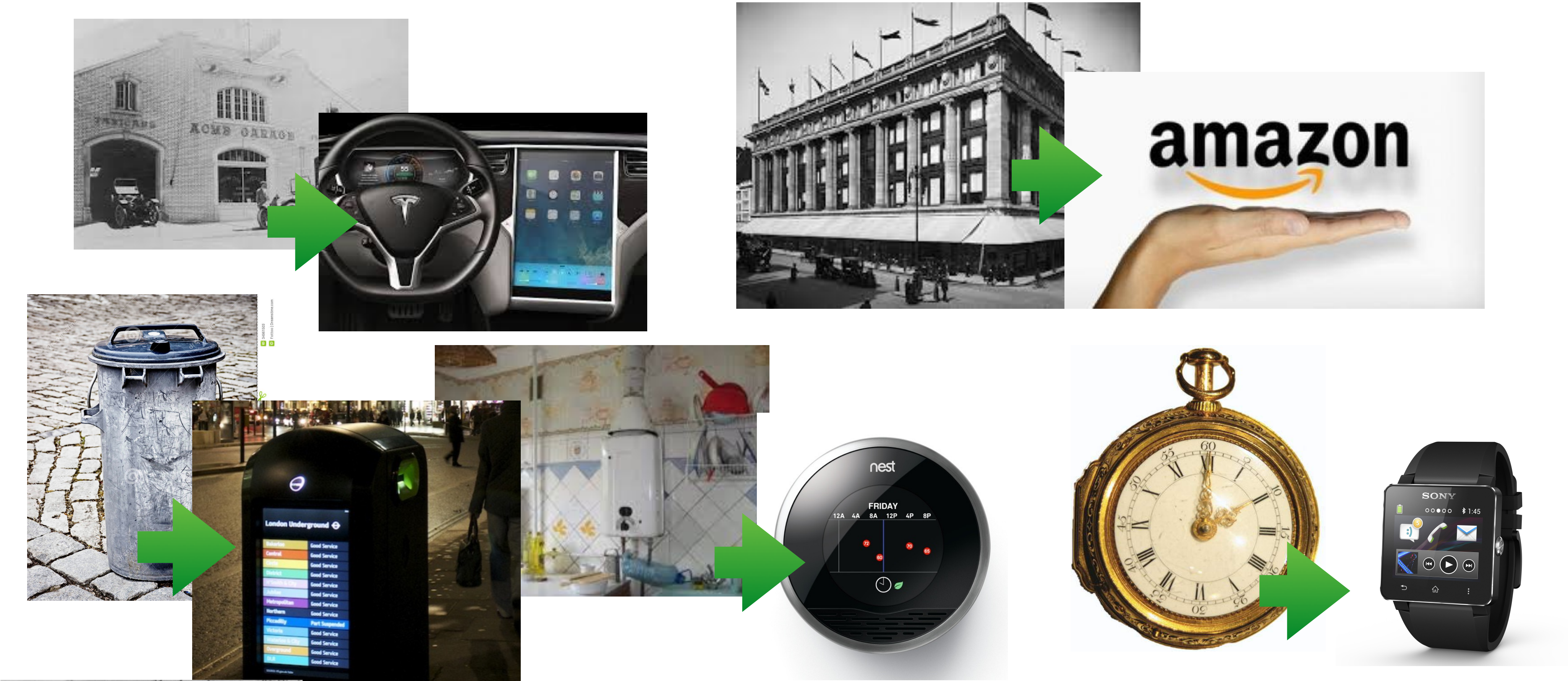
- **Software**
  - *is a collection of **computer programs** and related **data** that provide the instructions for telling a **computer** what to do and how to do it.* (Wikipedia)

- **Engineering**
  - *the way that something has been designed and built.* (Cambridge Business English Dictionary)

- **Software Engineering**
  - a collection of methods, techniques and tools that could be applied to design, build and maintain the "instructions for telling a **computer** what to do and how to do it"

What is a computer?

Everything, Everywhere



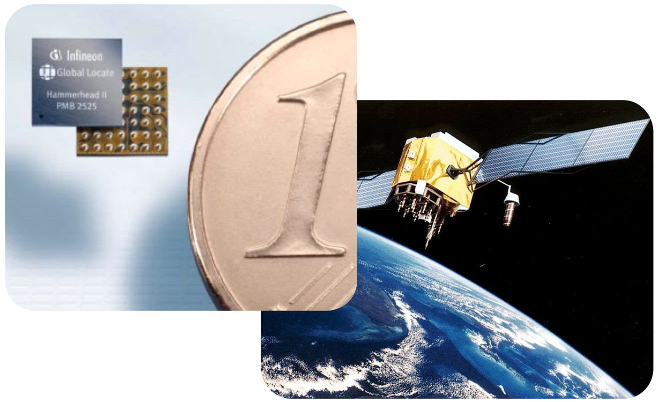
- **Everything is a COMPUTER (or needs one)**
- **A COMPUTER needs SOFTWARE**
- **Everything needs SOFTWARE :)**

What is a Computer?

Example: Tesla Car

- **Tesla iPhone App**
  - “The Tesla Motors app puts Tesla owners in direct communication with their cars anytime, anywhere”
- **Tesla In-Car software**
  - provides functionality to control the car functions and to access car’s location data

- **Tesla Application server**
  - manages the car, location and user data
- **GPS Receiver chip in the car**
  - processes GPS signals
- **Navigation Satellite**
  - broadcasts navigation messages



Everything is a COMPUTER (or needs one)

Software Types

Example: Tesla Car

- **Tesla iPhone App – consumer software application**
  - puts Tesla owners in direct communication with their cars anytime, anywhere

- **Tesla Application server – distributed concurrent software system**
  - aggregates and correlates user and location data

- **Tesla In-Car software – operating system, provides the platform for car applications**
  - provides functionality to control the car functions, including accessing car's location data acquired by the GPS receiver chip

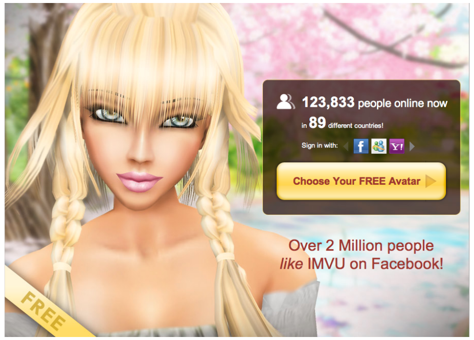
- **GPS Receiver chip – implements digital signal processing software**
  - processes GPS signals

- **Navigation Satellite – uses system control software**
  - broadcasts navigation messages

Engineering Approach: Horses for Courses

Not all SOFTWARE is created equal, choose an engineering approach accordingly

IMVU – one of the first companies to pioneer continuous deployment



Our tests suite takes nine minutes to run (distributed across 30-40 machines). Our code pushes take another six minutes. Since these two steps are pipelined that means at peak ***we're pushing a new revision of the code to the website every nine minutes****. That's 6 deploys an hour. Even at that pace we're often batching multiple commits into a single test/push cycle. On average* ***we deploy new code fifty times a day****.* [1]

Space shuttle – can there be a simple procedure to roll back to the previous version in the middle of the mission?

This software never crashes. *It never needs to be re-booted. This software is bug-free. It is perfect, as perfect as human beings have achieved. Consider these stats : the last three versions of the program – each 420,000 lines long-had just one error each.* ***The last 11 versions of this software had a total of 17 errors.*** [2]

A look back in history: Cambridge [3]

- **1948-9 Research on programming methods under M.V. Wilkes, including:**
  - *definition and refinement of Initial Orders (Wheeler);*
  - *closed subroutines (Wheeler);*
  - *building of a library of subroutines (all laboratory members interested in programming, plus Professor D. R. Hartree).*
- **6 May 1949, First logged program on EDSAC 1 (computing squares of 0-99) – World's first stored program computer**
- **1950 First Summer School on Programme Design for Automatic Digital Computing Machines, with 51 attendees**
- **1953 Diploma in Numerical Analysis and Automatic Computing began**
  - The Diploma "*would include theoretical and practical work ... [and also] instruction about the various types of computing-machine ... and **the principles of design** on which they are based.*"

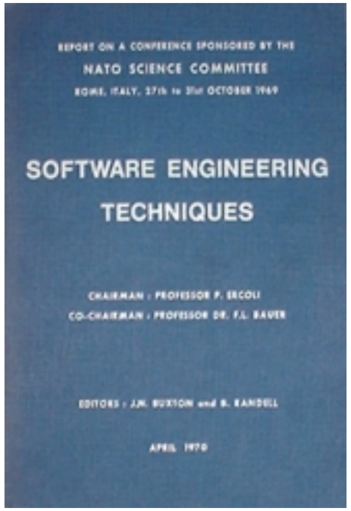
M.V.Wilkes, 1913-2010

EDSAC I, 1947/8 P.J.Farmer R.Piggott M.V.Wilkes W.A.Renwick

Software Engineering is Born

- **The NATO Software Engineering Conferences** - reached a crisis point
- **1968**: "*In Europe alone there are about **10,000 installed computers** — this number is increasing at a rate of anywhere from 25 per cent to 50 per cent per year. The quality of software provided for these computers will soon affect more than a quarter of a million analysts and programmers.*" [4]

P. Naur and B. Randell, (Eds.). Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. **1968**, Brussels, Scientific Affairs Division, NATO (1969) 231pp.



B. Randell and J.N. Buxton, (Eds.). Software Engineering Techniques: Report of a conference sponsored by the NATO Science Committee, Rome, Italy, 27-31 Oct. **1969**, Brussels, Scientific Affairs Division, NATO (1970) 164pp.

Typical Software Project (1968)

- **The need for feedback is already identified**

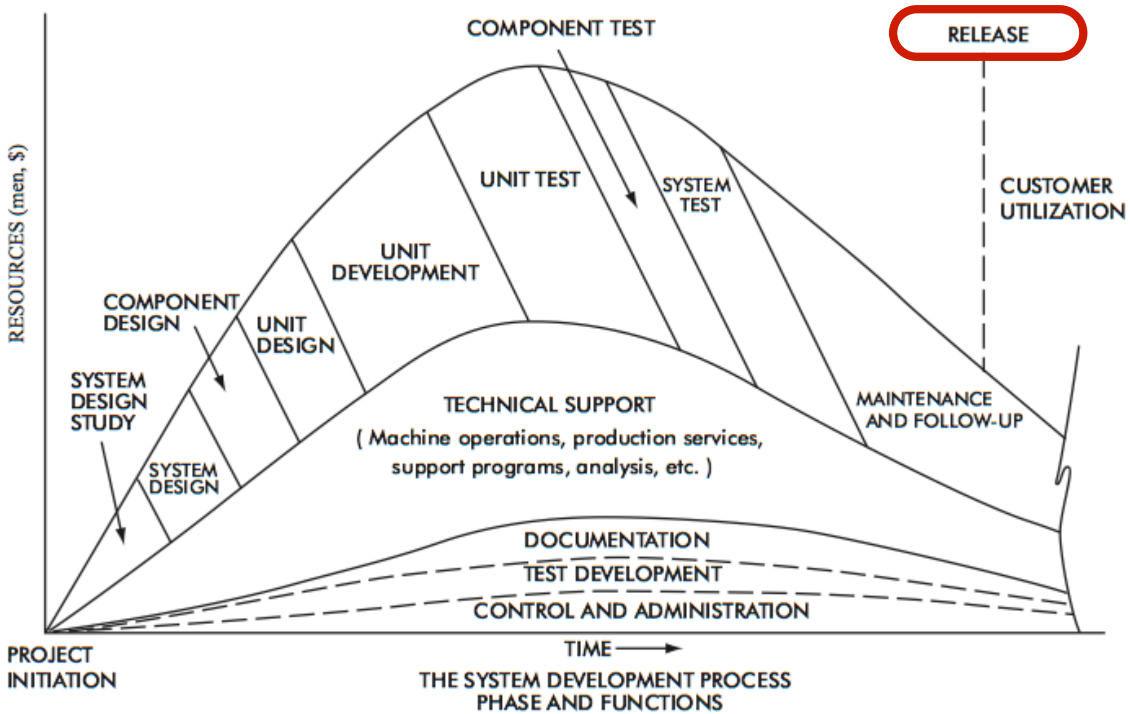


Figure 1. From Nash: Some problems in the production of large-scale software systems.

Selig: "External specifications at any level describe the software product in terms of the items controlled by and available to the user.

The internal design describes the software product in terms of the program structures which realize the external specifications. It has to be understood that **feedback between the design of the external and internal specifications is an essential** part of a realistic and effective implementation process.

Furthermore, this interaction must begin at the earliest stage of establishing the objectives, and continue until completion of the product." [4]

Learning From Mistakes

- **Traditionally, large scale project disasters are seen as the impetus for developing software engineering as a discipline in order to mitigate the risks**
- **Most referenced disaster projects include:**
  - 1991-1992, London Ambulance Service – an attempt to switch to a fully automated dispatch system (26 Oct 1992: went live, 02 Nov 1992: switched back to the manual system)
  - 1985-1987, Therac-25 – a computer-controlled radiation therapy machine massively overdosed 6 people
  - 1996, Ariane 5 – the first launch of a new rocket terminated in self-destruction due to a software bug in the control software
- **More recently**
  - 2003, Northeast Blackout – a power outage affecting est. 55m people in USA and Canada, an early alarm failed due to a "race condition" in the energy management system software
  - 2005-2006, Sensotronic Brake Control – almost 2m Mercedes cars recalled, the system is no longer used in production
  - April, 2011 – Amazon Elastic Compute Cloud (EC2) service disruption takes down FourSquare, Quora, Reddit
  - Oct 2011 – 3 days BlackBerry services outage affecting subscribers worldwide
  - Dec 2013 - Cyber Monday: IT outage leaves NatWest and RBS customers unable to use debit and credit cards on one of the busiest online shopping days of the year following systems meltdown in 2012

Complexity of the Software Systems

1986

> *Software entities are more **complex for their size** than perhaps any other human construct... Many of the classical problems of developing software products derive from this essential complexity and its nonlinear increases with size.*
>
> *Fred Brooks, "No Silver Bullet – Essence and Accident in Software Engineering"* [5]

2016

- **Hardware is more capable – has more and more software**
- **Software needs to support more features, use cases, devices, platforms**
- **Open source – more libraries, languages, frameworks to choose from**
- **Better communication channels – distributed teams**
- **Time to market is much shorter**
- **Overall complexity increased yet more manageable as smaller pieces**

Measuring Complexity: Lines of Code

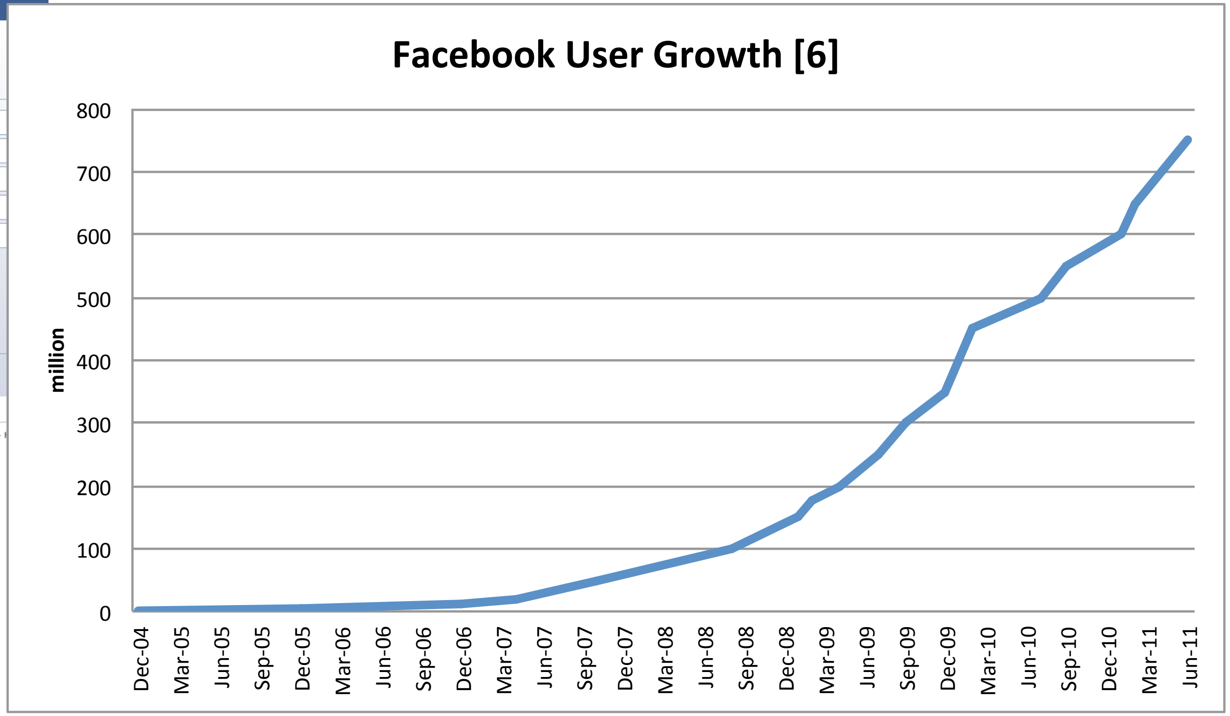
All data is as reported in 2011/2012, unless specified

- **F-22 Raptor: 1.7m**
- **Radio and navigation system in the current S-class Mercedes-Benz: 20m**
- **IBM OS/360 (1966): 1m**
- **Android Mobile OS: 1m**
- **Facebook main site: 9.2m**
- **Google Chrome browser: 4.5m**
- **Of course, the number lines depends on the programming language and less is not always better**

```
perl -MYAML -ne '$c{$_}++for split//;END{print Dump\%c}' data.txt
```

Measuring complexity: another prospective

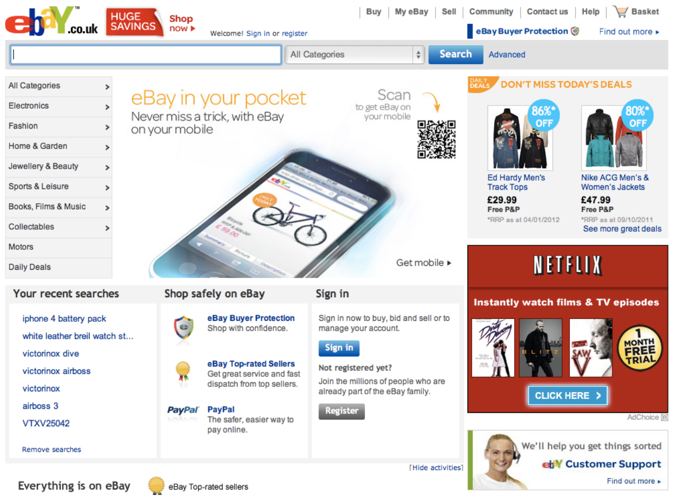
Facebook, data as reported 2011/2012



- 800m users [8]
- 500m visit daily
- 350m on mobile
- all data is stored in a database (MySQL)
- 60 million queries per second
- 4 million row changes per second

Measuring complexity: another prospective

eBay, data as reported 2011/2012



- **"The size of the problem"**
  - "The problem involves both the number of changes and the velocity of the changes. The eBay code base consists of hundreds of thousands of source elements (with total lines of code in the tens of millions). The applications – which comprise web, services, messaging, and batch style – number in the thousands. It is common for an application to use tens of thousands of source elements. The elements themselves are contributed by different teams and are shared with other applications." [8]

Code Type	Count
Source elements	100,000's
Source elements per application	10,000's
Applications	1000's
Features per month	100's
Source elements changed per feature	100's (sometimes 1000's)
Projects	1000's
Source elements per project	10-100's (sometimes 1000's)

Ever Increasing Complexity: Better Faster Stronger

End of 20th century

- Google, PayPal - only just founded (1998)
- Firefox/Chrome did not exist
- First smartphone with touchscreen and iPod are about to appear
- Social media is an unknown term (even MySpace didn't exist)
- "DotCom" was still a buzzword, no "wearables" no "Internet of Things"

Data from 2011/2012

- Instagram: 60 photos per second;
- Tumblr: 900 posts per second
- Twitter: 8,868 Tweets per second (during MTV Awards)
- Netflix: stream 1 billion hours in Q4 2011
- PayPal: 5 million transactions a day
- London Stock Exchange: 22.4 million trades in Dec 2011
- Google: over 1 billion searches per day
- Youtube: over 100 million views a day on mobile

All data is as reported in 2011/2012, unless specified

Ever Increasing Complexity Software Systems Drive World's Economic Growth

Software Engineering Aims

- **Managing complexity and Minimizing risks**

> *[T]here are **known knowns**; there are things we know we know. We also know there are **known unknowns**; that is to say we know there are some things we do not know.*
> *But there are also **unknown unknowns** – there are things we do not know we don't know.*
>
> *Former United States Secretary of Defense Donald Rumsfeld*

- **Designing systems that**
  - meet requirements ("known knowns")
  - can adapt to changes ("known unknowns")
  - and withstand any unexpected use cases ("unknown unknowns")

- **Software Engineering is also about:**

Making Better Mistakes Tomorrow!

Having Happy (Alive) Users!

Software Process

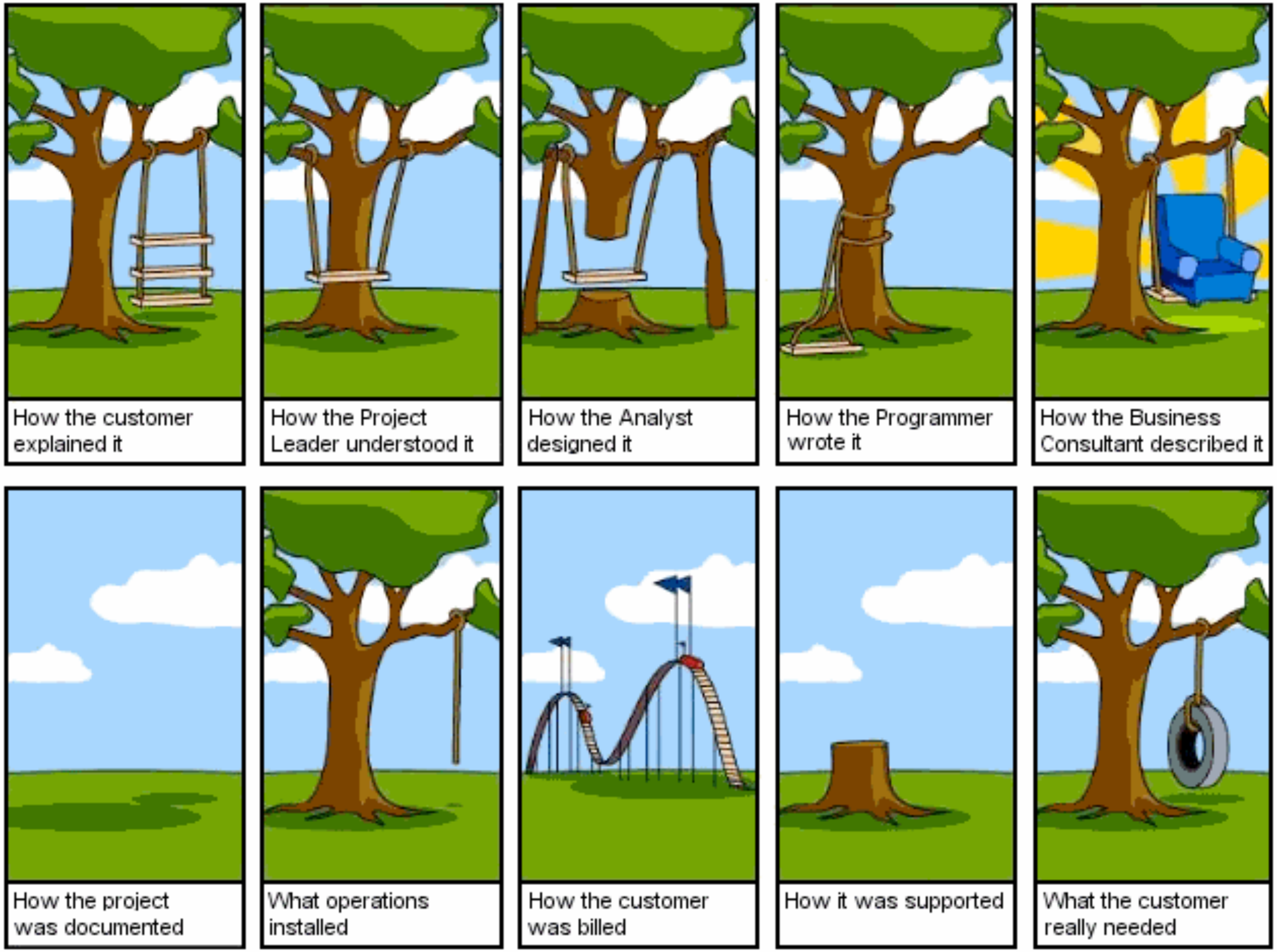
- "**Analysis** is discovering and describing those aspects of a software development about which there is no choice, that is, to which the project is already committed"
- **Design** is creating a definition of how the project goals are going to be achieved
- **Implementation** is the process of writing code, typically would be partitioned in many subprojects
- **Building** is creating a "complete" version of the software, i.e. putting all chunks of code together, including any custom configurations for target deployment
- **Testing** is about making sure that small independent parts of code work correctly (unit tests), that all code parts work together (integration tests), that the functionality meets requirements (acceptance), that any new code doesn't break the old (regression)

Software Process

- **Deployment** – actual release of the software into the end user environment, e.g. publishing the mobile app in the App Store or launching the service on bank's servers

- **Maintenance** – supporting the software during its lifetime, e.g. releasing compatibility updates when a new version of mobile OS is released or improving performance as user base grows
- Traditionally, **80-90%** of software system **Total Cost of Ownership** is attributed to **maintenance**. Once the system is operational, the cost of change is high (each new release requires a new full cycle: analyse, design, implement, build, test, deploy). Also, to achieve a better **Return on Investment**, the preference is naturally to extend the existing system than develop a new one.
- Nowadays, in some software systems (web apps), the line between maintenance and continuous development is less clear, consider Google and Facebook – is scaling up to *hundreds of millions of users* and *PetaBytes of data* maintenance or new development?

Example


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Requirements Analysis not Analysis Paralysis

> *The **hardest single part** of building a software system is **deciding precisely what to build**. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong.*
>
> *Fred Brooks, "No Silver Bullet – Essence and Accident in Software Engineering"* [5]

- **It is impossible to know in advance everything required to build the software system. Why?**
- **Users don't know 100% what it should do**
- **Developers don't know 100% how users would use it**
- **The only 100% certainty is that things will change:**
  - the deployment environment change, e.g. new version of hardware/software platform
  - the business requirements change, e.g. adding a new method to a payment system
  - the scale changes, e.g. from 1m users to 800m

Cambridge University Engineering Department

The Answer?

- **Almost 50 years since The 1968 NATO Software Engineering conference, the issues discussed are as much present as then**
- **No Silver Bullet – there is no single answer/process/method**
- **but we can ...**

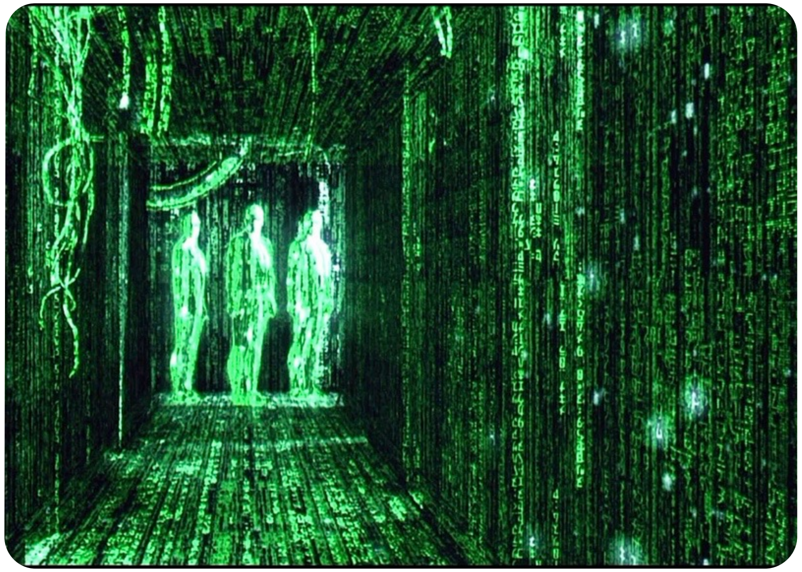
✓ **Build for Change**

✓ **Communicate Clearly**

✓ **Test Continuously, Test Everything**
  - problem understanding, user behaviour assumptions, code, integration...

✓ **Choose the Tools for the Job**
  - Object-Oriented Analysis and Design
  - Extreme/Agile Programming
  - Continuous Integration/Deployment
  - IDEs, Source Control, UML ...

> *Grow, don't build, software.*
> *Fred Brooks, "No Silver Bullet – Essence and Accident in Software Engineering"*



References and Further Reading

1. http://timothyfitz.wordpress.com/2009/02/10/continuous-deployment-at-imvu-doing-the-impossible-fifty-times-a-day/
2. http://www.fastcompany.com/magazine/06/writestuff.html
3. http://www.cl.cam.ac.uk/conference/EDSAC99/history.html
4. http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF
5. http://www.cs.nott.ac.uk/~cah/G51ISS/Documents/NoSilverBullet.html
6. http://www.facebook.com/press/info.php?timeline
7. http://www.facebook.com/press/info.php?statistics
8. http://www.ebaytechblog.com/2011/12/15/rapid-development-setup-in-large-environments/
9. http://www.securityfocus.com/news/8016
10. http://aws.amazon.com/message/65648/
11. http://www.autoweek.com/article/20051216/FREE/51216010
12. http://www.computerworlduk.com/news/mobile-wireless/3310790/we-do-not-know-why-system-failed-says-blackberry-ceo/
13. http://spectrum.ieee.org/green-tech/advanced-cars/this-car-runs-on-code/0
14. http://highscalability.com/
15. http://martinfowler.com/bliki/UseCases.html