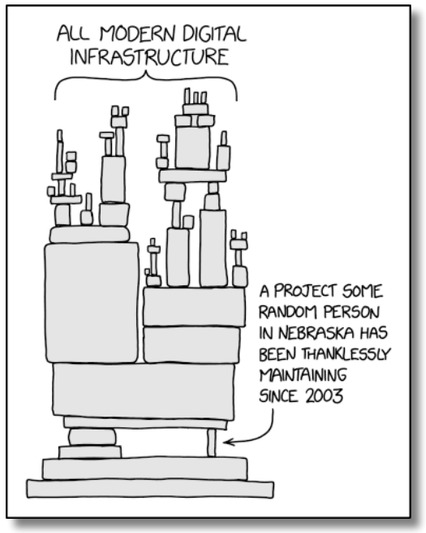





The University Of Sheffield.

# Change in Software Systems

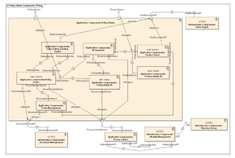
Software Reengineering
(COM3523 / COM6523)

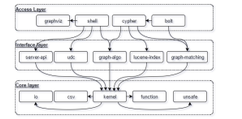
The University of Sheffield



# Why do Systems Change?







# Hardware

Hardware is constantly improving.

- E.g. Rise of GPUs for parallel computing.
- Moore's law: ICs double transistors every two years
- Games consoles, mobile phones, etc.

Difficult to replace.

- Can be physically difficult to extract.
  - E.g. circuitry that is built into aircraft.
- Embedded components can include hidden logic.
  - Difficult to reverse-engineer if not documented.


**IBM 5MB HD in 1956 - weighed > 1 ton.**

**America's nuclear arsenal runs on 45 year-old floppy disks.**



# Operating Systems

Constantly emerging to incorporate changes in hardware.

- Especially common for games consoles and mobile devices.

Vendors eventually remove support for older versions.

- Can force the issue when OS support is relied upon as part of a complete solution.

Can be challenging to adapt legacy code to:

- Often involve changes to file systems.
- Security considerations.





# Compilers, Virtual Machines, Interpreters

Tailored to suit different platforms.

Often associated with extensive development kits / libraries.

Can require change for many reasons:

- Deprecation of language features.
- Removal of support.
- License changes.
- Changes in application needs.
  - E.g. Older compilers cannot handle Unicode text.

**License Rights and Restrictions**
Oracle grants You a nonexclusive, nontransferable, limited license to internally use the Programs, subject to the restrictions stated in this Agreement and Program Documentation, only for the purpose of developing, testing, prototyping and demonstrating Your Application and not for any other purpose. You may allow Your Contractor(s) to use the Programs, provided they are acting on Your behalf to exercise license rights granted in this Agreement and further provided that You are responsible for their compliance with this Agreement in such use. You will have a written agreement with Your Contractor(s) that strictly limits their right to use the Programs and that otherwise protects Oracle's intellectual property rights to the same extent as this Agreement. You may make copies of the Programs to the extent reasonably necessary to exercise the license rights granted in this Agreement.

Further, You may not:
- use the Programs for any data processing or any commercial, production, or internal business purposes other than developing, testing, prototyping, and demonstrating your Application;

**Oracle's license for Java 11 prohibits the use of compiled programs for "commercial, production, or business purposes"**



# Source Code

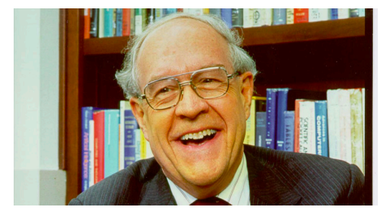
In many ways the most "malleable" part of a system.

- Changes to source code can work-around changes further down the stack.
- Successful code is changed on a daily basis by full-time developers for decades.
  - E.g. the Linux kernel.

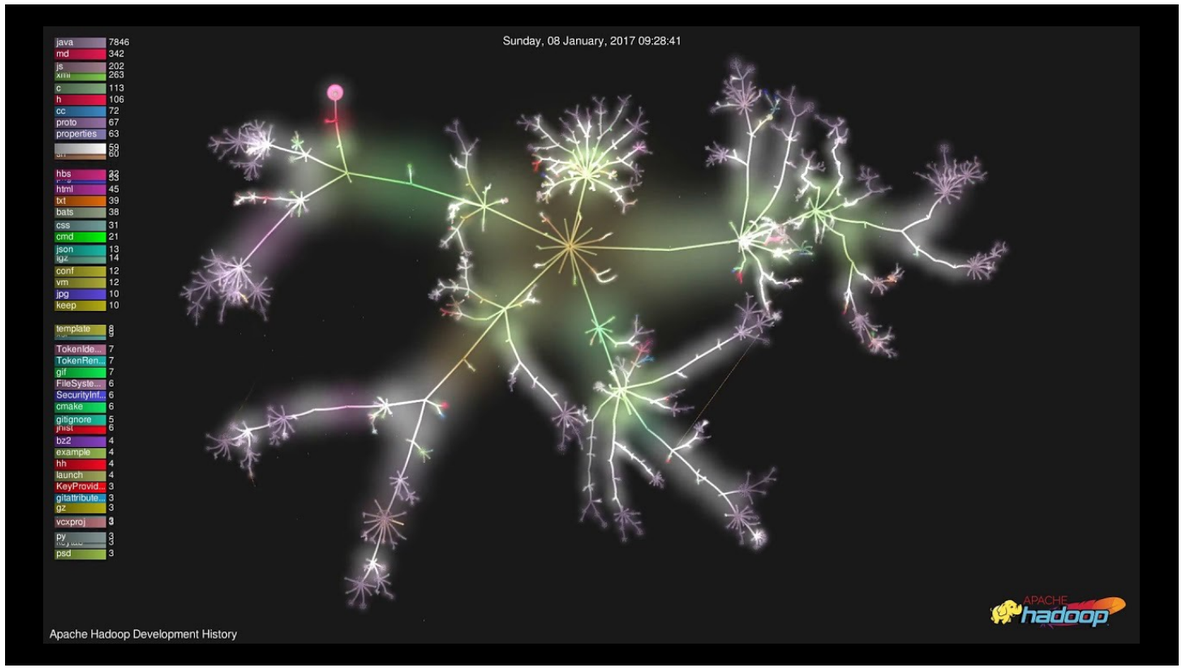
Invariable increase in scale and complexity.

- Some of which is **essential** to implement necessary features.
- Some of which is **accidental** - due to poor design.

*Fred Brooks, "No Silver Bullet: Essence and Accidents in Software Engineering", Computer (20),4, 1987*



**Fred Brooks - originator of the notions of "accidental" and "essential" complexity in software**




# Data Storage

Many drivers for change

- Change in legislation.
  - E.g. Change to rules about what types of data can be stored, for how long.
- Change in database system.
  - Differences in reliability, efficiency, ease of use, compatibility, etc.
- Change in schemas, storage formats

Migration can be extremely challenging.

- Data often always needs to be available.
- So do the systems that access and manipulate it.





# Front-end / UI

Several drivers for change

- Accessibility improvements
- Usability / UX
- New modes of interaction
  - Touch gestures
  - Motion sensing devices
  - Voice activation
- Changes in Browser / GUI technology



# Users

Subject to market forces

- Exciting new products come to market with innovative features.
- Broader eco-system might change.
  - Might demand compatibility with other applications and services.

Habits can change (fitness trackers, social media, etc.).

Privacy or security concerns.

Might seek to misuse the system - identify loop-holes, security vulnerabilities.

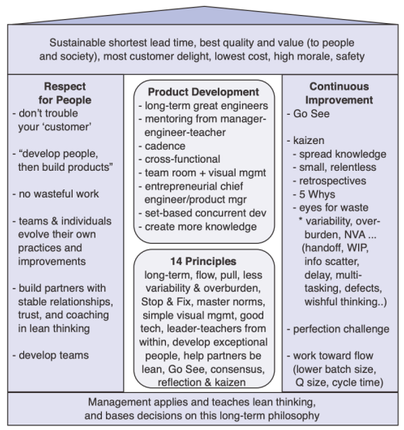
# Business processes

Businesses need to continuously tweak their processes to support new products and save costs.

- Referred to as **Process Improvement**
- E.g. Lean manufacturing, Six Sigma.

Leads to:

- Different use-cases and functionalities.
- Different classes of users.




**The Toyota "Thinking House" system that underpins its TPS process improvement system.**



# Business Structure

**Conway's Law**:

*"Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations."*

*Melvin Conway, "How do Committees Invent?", Datamation, 14 (5): 28–31, 1968*

Organisational changes tend to lead to changes in enterprise software.

- Supported by an empirical study at Microsoft
- Demonstrated relationship between organisational and software metrics.

Nagappan, Murphy, Basili. "The influence of organizational structure on software quality.", International Conference on Software Engineering (ICSE), 2008.



# Case Study

Unix represents the date as the number of seconds since 1st Jan. 1970.

As a 32-bit integer, the maximum is hit on the 19th January 2038.

After that, it overflows to a negative value representing the 13th Dec. 1901.

Expected to affect **wide** range of systems, especially time-sensitive embedded ones.

- E.g. ABS systems in cars, inertial guidance systems in aircraft, etc.

All will need to be **reengineered** to work around the problem.

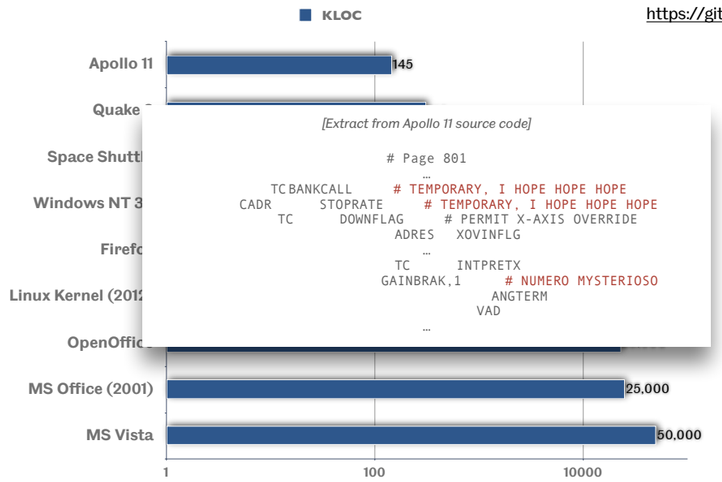



**Illustration showing how the date would reset if represented as a signed 32-bit integer.**



# Why is Reengineering Hard?

# Scale




*[Extract from Apollo 11 source code]*

```
                # Page 801
                        …
     TC BANKCALL      # TEMPORARY, I HOPE HOPE HOPE
CADR       STOPRATE      # TEMPORARY, I HOPE HOPE HOPE
     TC       DOWNFLAG      # PERMIT X-AXIS OVERRIDE
                   ADRES   XOVINFLG
                        …
                   TC      INTPRETX
                GAINBRAK,1      # NUMERO MYSTERIOSO
                             ANGTERM
                            VAD
                        …
```

https://github.com/chrislgarry/Apollo-11

**Margaret Hamilton next to the 1969 Apollo 11 moon lander code.**

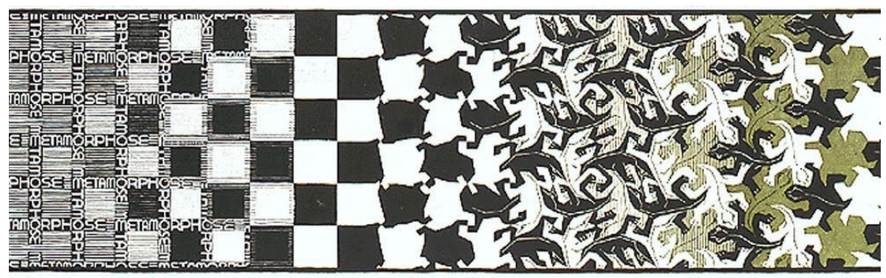


# Continuous change

Code structure is continuously changing.

Impossible to maintain a fixed, reliable document of architecture or design.

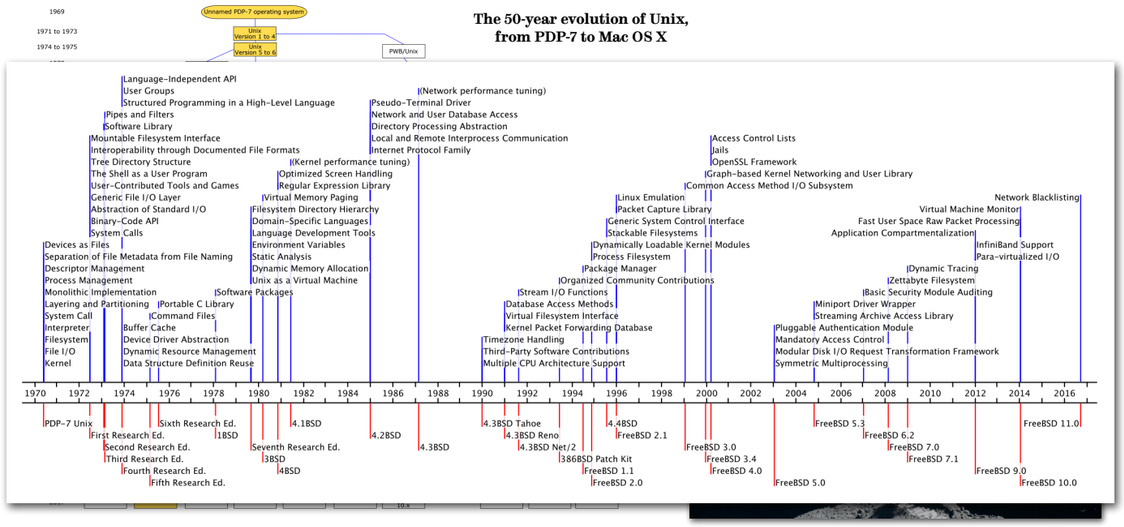
Developer knowledge of the system rapidly becomes outdated.



**Metamorphosis, M.C. Escher**



# An example: Unix




**The 50-year evolution of Unix, from PDP-7 to Mac OS X**



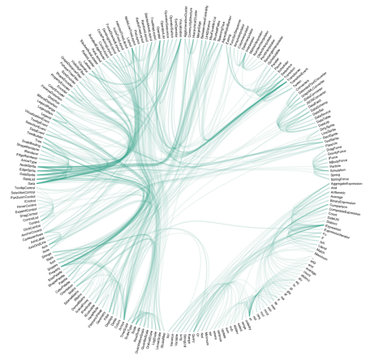
# Complexity

Functionality emerges from interactions.

An intricate network of dependencies and data-flows.

Difficult to anticipate from source code alone.

Behaviour is "invisible" - relies on the ability to execute code mentally.



**Calls between files in the Flare Actionscript visualisation library**

## Abstract and difficult to conceptualise

Legacy code is:

- Dynamic
- Built on abstractions that may no longer be appropriate.
- Have been gradually re-purposed over time.

Difficult to conceptualise.

Difficult to plan a re-engineering strategy…

- … figure out where to start, …
- … what to change,
- and how.



# Why not start from scratch?



## It's extremely risky.

A lack of requirements and specification.

- They are implicit, "baked in" to the source code of the legacy system.
- Extremely difficult to extract.
- Highly complex; they have evolved and grown, potentially over decades.

Difficult to anticipate cost and ensure reliability.

Difficult to ensure that the replacement is able to replace the old system.

High stakes for business critical systems.

- Failure at "switchover" could lead to calamity.



## Case study

TSB bank split from Lloyds Bank in 2013.

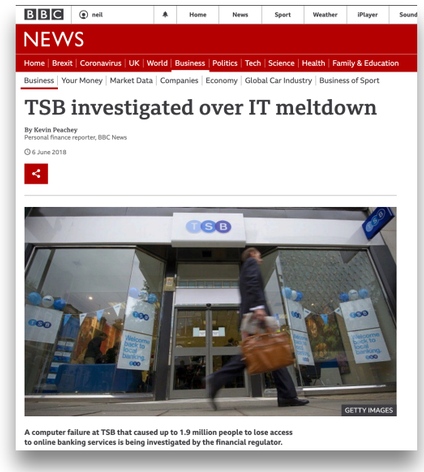
- Had used legacy system to manage bank accounts

TSB decided in 2018 to migrate accounts from Lloyds system to new platform to facilitate digital banking.

Led to disruption to millions of TSB customers, including:

- Loss of internet and banking services for >1 week.
- Customers being able to view each others' banking details.
- Money disappearing from accounts.

Led to estimated £366m loss for TSB.



# Take-aways

Many external drivers for change.

- Often happens by necessity, rapid bug fixes, added features.
- Lack of oversight and consideration of all of the various **architectural viewpoints and perspectives.**

Lots of “horizontal” and “vertical” dependencies.

- Changing one component leads to knock-on effects and can have unintended consequences.

Lots of intrinsic challenges

- Scale, continual change, dynamism, …

**Reengineering is an important skill-set to have!**