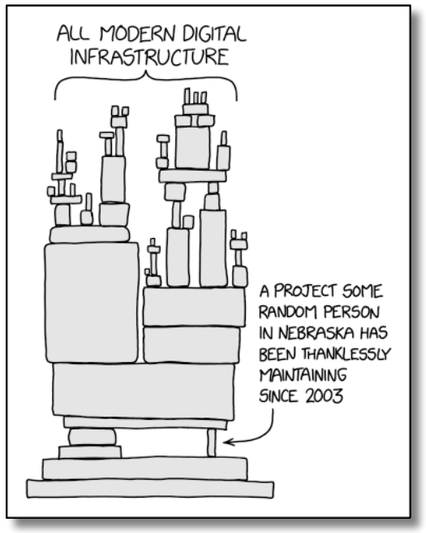


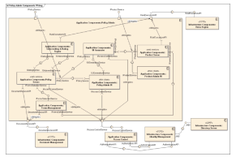
The University Of Sheffield.

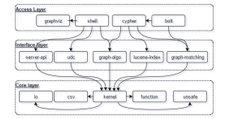
Change in Software Systems

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

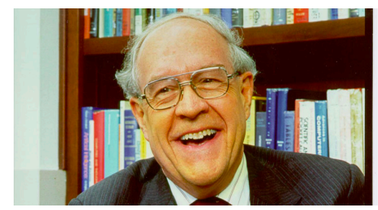
In many ways the most "malleable" part of a system.

- 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.

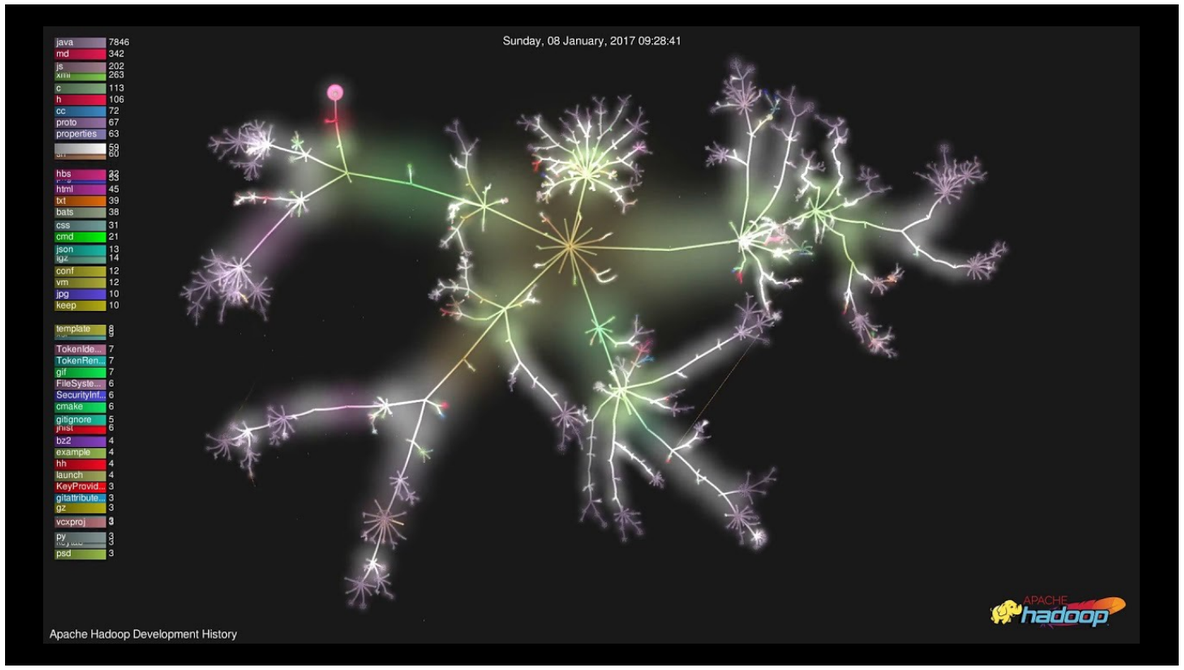
Invariable increase in scale and complexity.

- 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.

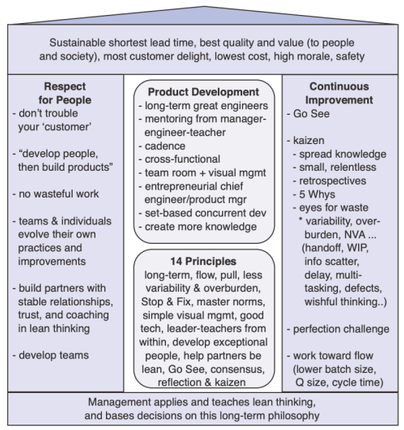
Business processes

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.

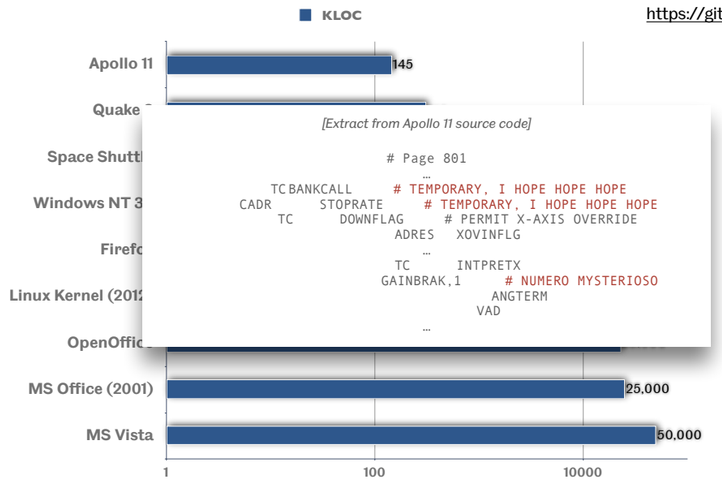
- E.g. ABS systems in cars, inertial guidance systems in aircraft, etc.

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

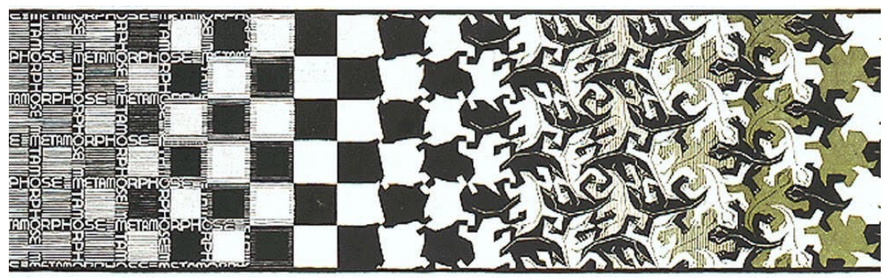
Margaret Hamilton next to the 1969 Apollo 11 moon lander code.

Continuous change

Code structure is continuously changing.

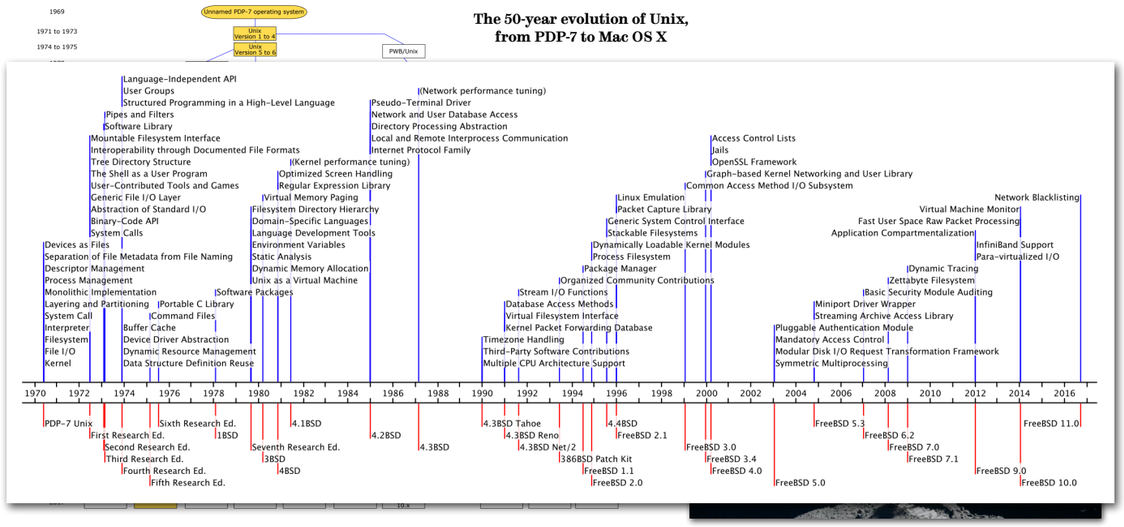
Impossible to maintain a fixed, reliable document of architecture or design.

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

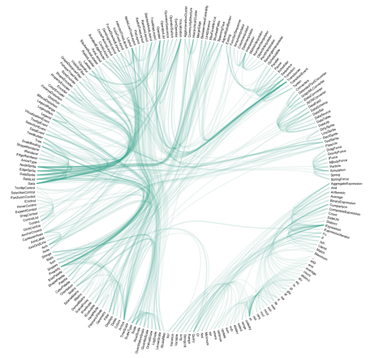
Complexity

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.

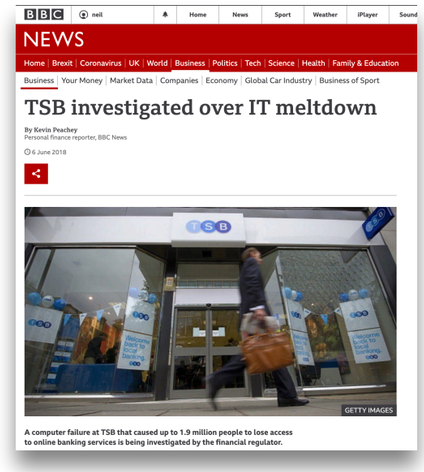
- Had used legacy system to manage bank accounts

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!

The University Of Sheffield.

Group work

Software Reengineering
(COM3523 / COM6523)

The University of Sheffield

Allocation into Groups

You have been allocated into groups of 5-6.

- We have done our best to respect the requests within the submitted Google Forms.

Your groups should all be on Blackboard now.

GitLab

This course will involve the use of GitLab.

- You’ll be using Git repositories for your course work.
- You will be learning how to analyse projects using Git as well.

We run a GitLab instance in the department.

- This enables you to log-in using your departmental credentials, instead of relying on a GitHub account.

It is important that you familiarise yourselves with how to use Git.

- For those of you who are new to Git, there are some excellent resources on LinkedIn Learning.
- LinkedIn Learning is subscribed to by the University, you should be able to access these courses through Muse.
- One good course is “Git Essential Training: The Basics”, by Kevin Skoglund (link on BB).

Maven

Maven is a popular framework within which to build an execute Java programs.

- Developers provide configuration details (e.g. links to external libraries) in a configuration file.
- Automatically manages class-path and dependencies.

You will not require an in-depth understanding of Maven, but a good overview of how it works would help.

- You can bring yourself up to speed with LinkedIn Learning.
- For example, “Introducing Maven” by Frank P Moley III (linked).

Choosing a group project.

As a group, you will study an open source Java project.

Your choice is significant.

- You will be analysing it every week, to learn how to implement the techniques.
- You will submit your group project on this system.

You will be asked to propose three Java projects as a group, which must fulfil the following criteria:

- (1) It is on GitHub
- (2) It is of a reasonable size. It should have more than 20 classes. Ideally many more.
- (2) It is primarily Java (we do not count Android or languages built on the JVM such as Kotlin.)
- (3) It is a Maven project - it should have a pom.xml file.
- (4) It should build (but its tests don’t need to pass). If you run

```
mvn package -DskipTests
```

, it should not throw any errors.

Before you meet your group…

There is a list of links to candidate projects attached to this item in Blackboard.

Use this as a starting point to identify one or more projects that fulfils the criteria on the previous slide.

- Keep a note of the GitHub URL.
- Make sure that you have actually tried to build it!

… when you meet your group

Introduce yourselves to each other!

Start off my making sure that you all understand the task.

- If there are any queries that you’re not sure of as a group, keep note of them so that you can ask a staff member when they join your group session.

Pick your systems.

- You need to submit a list of three or more projects that you’d be willing to research.
- Take it in turns to talk about systems you came across that you were interested in.
- Enter them into the Google form attached to this presentation as you go.

Note: No two groups will be permitted to study the same project, they must all be different.

Engagement

Attendance at practical sessions will be monitored.

An individual's mark for the final group-work will factor in attendance at these weekly sessions.

Blogs

You are encouraged to chart your journey as a group using the "Blogs" facility in Blackboard.

For your weekly group work use the blogs to:

- Share the results of that week's work with other groups.
- Add any comments or surprise findings.
- Useful for turning into your final group report and for the final presentation.

We will not be assessing the blogs.

The University Of Sheffield.

Informal Analysis

Software Reengineering
(COM3523 / COM6523)

The University of Sheffield

Strategy

Work from the "top-down".

- What is the Context viewpoint?
  - Broader context, dependencies, scope, interfaces.
- What is the Development viewpoint?
  - How is the system organised in terms of code, tests, and development support infrastructure?

Objectives:

- Look for signs that might indicate problems.
- Develop a broad understanding of the system.

For this lecture, we're going to focus on ZXing.

Context Viewpoint

System scope and responsibilities

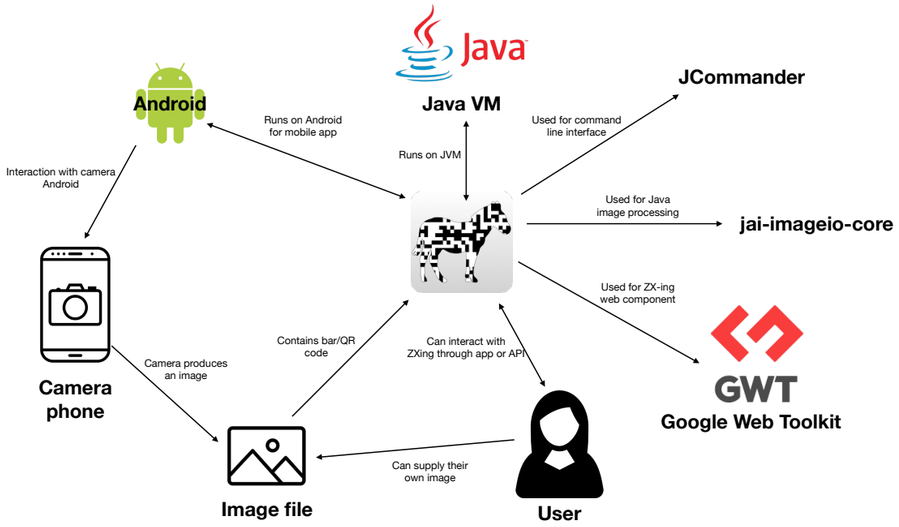
Key responsibilities (taken from GitHub webpage, and “Getting Started” docs)

- Scan a variety of different types of barcodes
  - 1D (for products and industrial usage), and 2D.
  - Supports a variety of different formats for each type.
- Can be used in Android (for use on mobile devices), Java, or web.
- Primarily an image-decoding app.
  - Does it also *encode*? Not clear from superficial view of GitHub.
- Lots of ports and wrappers for other languages (e.g. Python, Ruby, C++, etc.).

Exclusions

- It does not include functionality to take an image - assumes this is supplied from elsewhere.
- It does not include product information pertaining to barcodes.

Identities of external entities



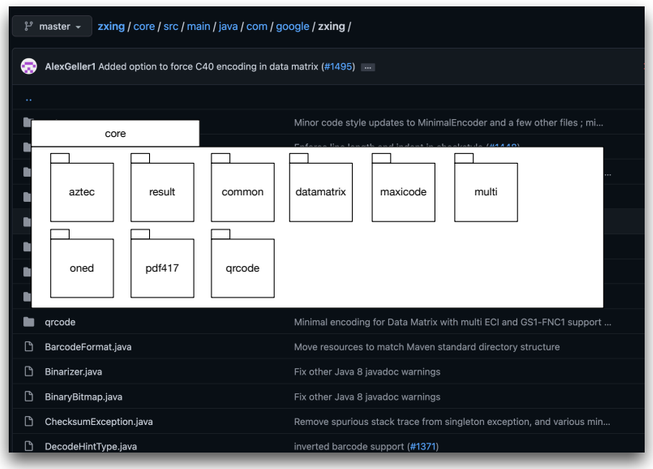
Development Viewpoint

Module Structure

Components

Active

Module	Description
core	The core image decoding library, and test code
javase	JavaSE-specific client code
android	Android client Barcode Scanner 
android-integration	Supports integration with Barcode Scanner via `Intent`
android-core	Android-related code shared among `android`, other Android apps
zxingorg	The source behind `zxing.org`
zxing.appspot.com	The source behind web-based barcode generator at `zxing.appspot.com`



Common Processing

3rd party libraries

- jai-imageio-core for image processing
- JCommander for command line processing

Noteworthy package - core.common

- Appears to have plenty of utility classes for use across system.

Very self-contained, most of the common processes (e.g. logging, exception handling etc.) managed through Java standard libraries.

Testing

Relatively comprehensive looking set of JUnit tests

- Stored in tests directory
- test/resources directory contains plenty of images of test barcodes / qr-codes

Appear to be limited to unit tests of the API, no integration tests?

Codeline Organisation

Code versioning managed via Git on GitHub.

Build is configured via Maven.

- Each module has its own Maven configuration file (pom.xml).

Adopts Apache 2.0 License

Latest version is 3.4.1 (September 2020)

- GitHub contains predecessors dating back to Barcode Scanner 4.5.1 (January 2014)

Operates on pull-based development.

- Use of pull-requests to accommodate changes.

In “Maintenance mode” - only refinements and minor feature enhancements accepted.

A Superficial Analysis of the Code Files

Identify “exceptional” entities.

Find unusual entities.

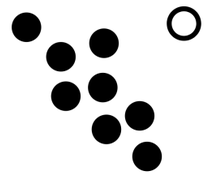
- e.g. methods or files that are especially large.

Can indicate importance.

- Can also indicate poor design - poor distribution of responsibilities.
- A potential starting point for identifying weaknesses.

Extract data using simple script commands.

Load into a spreadsheet or a data-processing tool.



“Identify Excepional Entities”, Chapter 4.3, “Object Oriented Reengineering Patterns”, Demeyer et al.

Extracting the relevant information via the CL

We will be doing this in the coming lab class.

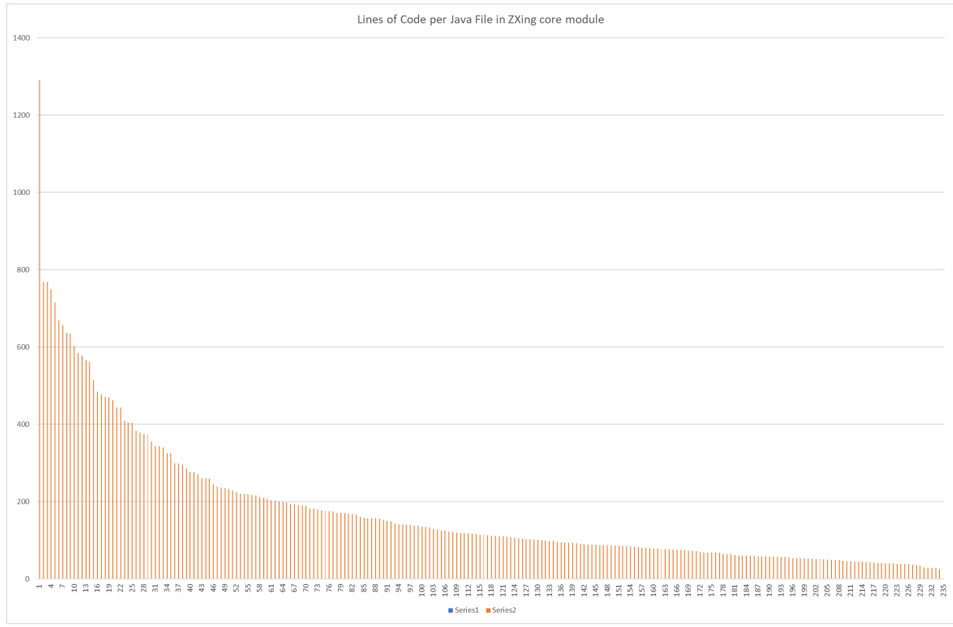
How many java files are there in the zxing core component?

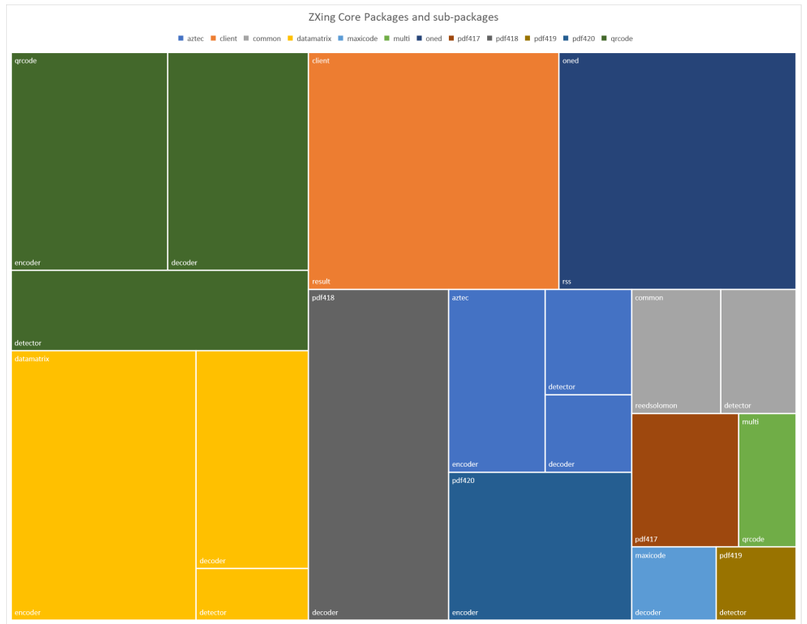
```
ac1nw@TENB88584C305C5 MINGW64 /u/Teaching/Reengineering/2022/practicals
$ find zxing/core/ -name '*.java' | wc -l
375
```

Which (source) files are the biggest one - what is the distribution?

```
ac1nw@TENB88584C305C5 MINGW64 /u/Teaching/Reengineering/2022/practicals
$ head listFileSizes.sh
#!/bin/bash

for file in `find $1 -name $2`
do
        total=$(wc -l < $file)
        echo "$file,$total"
done
```



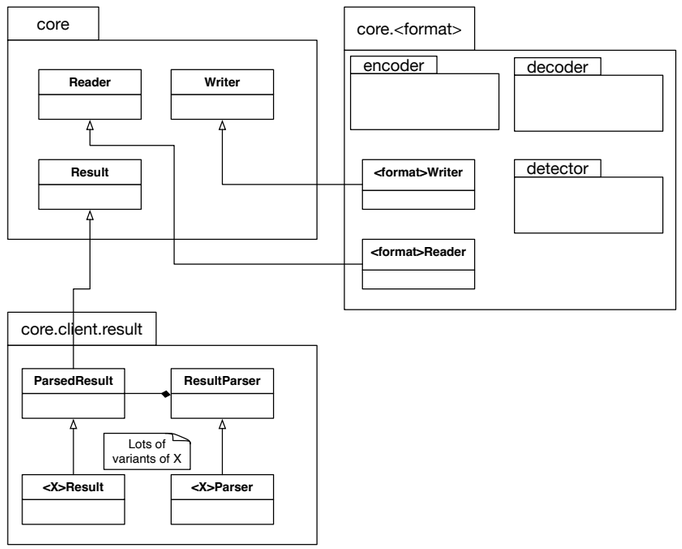
Speculating about the design

Take a guess at the key elements of the system functionality.

Start by identifying potential key interfaces or abstract classes.

Look at naming patterns.

"Speculate about design", Chapter 4.2, "Object Oriented Reengineering Patterns", Demeyer et al.



Conclusions and next steps

Conclusions

Important to consider the wider context before diving into the source code.

Start from a contextual view, and move down into the detail.

Can use an iterative approach, starting from a hypothesis and checking the code.

This can be supported by some simple automated tools, starting with Bash commands, which brings us to…

… **Next steps:** Watch the two tutorials on Bash Analysis.