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

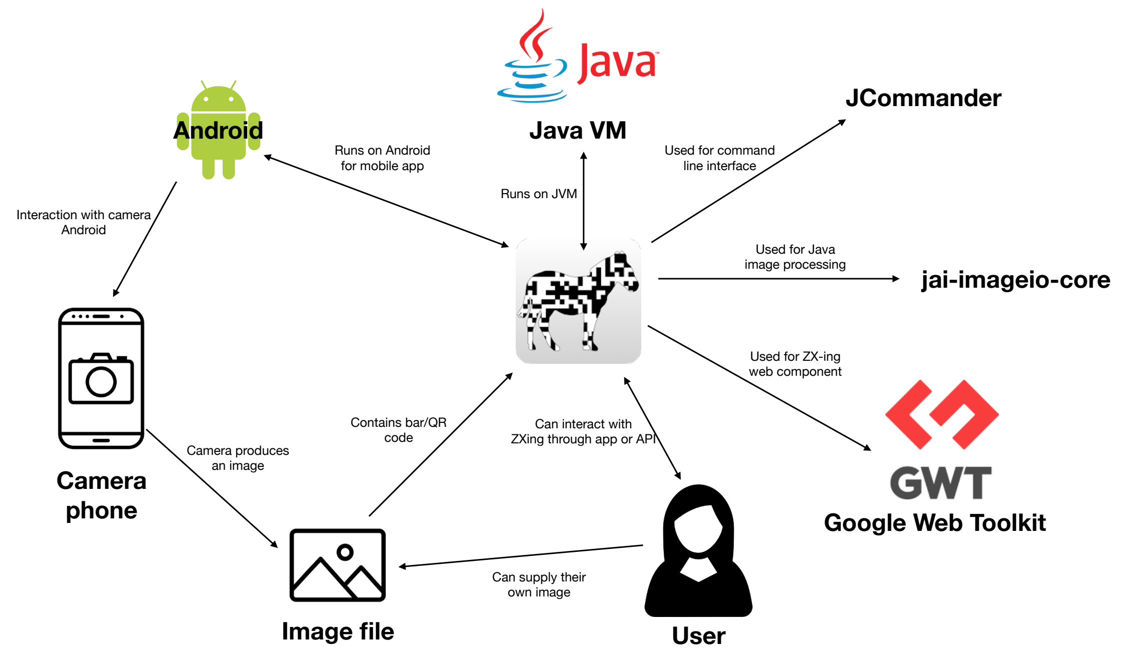
Key responsibilities (taken from GitHub webpage, and "Getting Started" docs)

- 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.

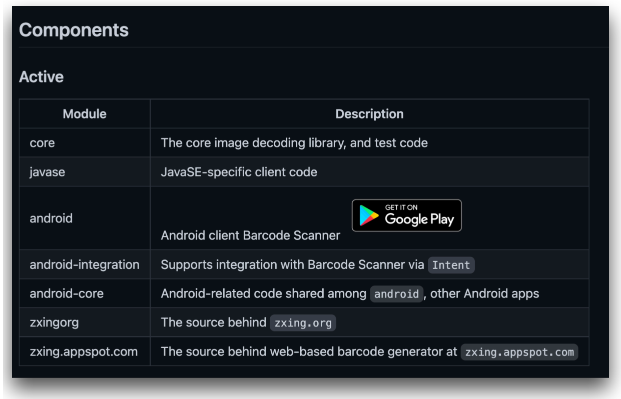
Identities of external entities



Java
Java VM
Android
JCommander
Runs on Android for mobile app
Runs on JVM
Used for command line interface
Interaction with camera Android
Used for Java image processing
jai-imageio-core
Used for ZX-ing web component
GWT
Google Web Toolkit
Camera phone
Camera produces an image
Contains bar/QR code
Can interact with ZXing through app or API
Can supply their own image
Image file
User

Development Viewpoint

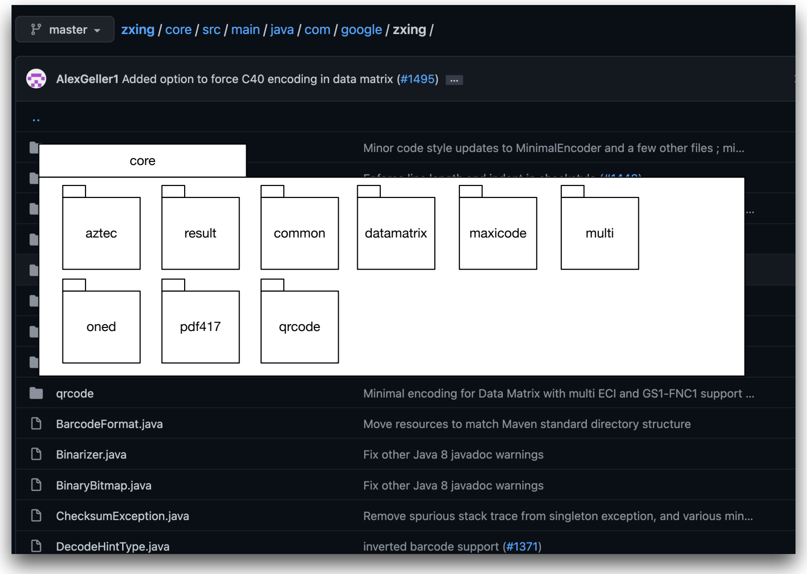
Module Structure



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`



master zxing / core / src / main / java / com / google / zxing /

AlexGeller1 Added option to force C40 encoding in data matrix (#1495)

..	
	Minor code style updates to MinimalEncoder and a few other files ; mi...
qrcode	Minimal encoding for Data Matrix with multi ECI and GS1-FNC1 support ...
BarcodeFormat.java	Move resources to match Maven standard directory structure
Binarizer.java	Fix other Java 8 javadoc warnings
BinaryBitmap.java	Fix other Java 8 javadoc warnings
ChecksumException.java	Remove spurious stack trace from singleton exception, and various min...
DecodeHintType.java	inverted barcode support (#1371)

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?

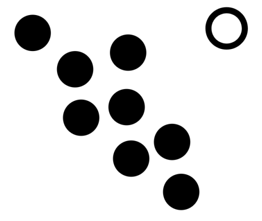
Codeline Organisation

- 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.

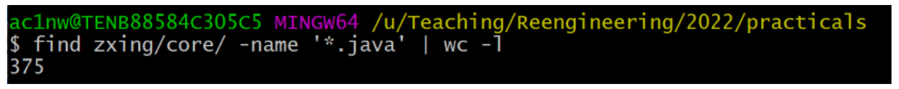


“Identify Excepional Entities”, Chapter 4.3, “Object Oriented Reengineering Patterns”, Demeyer et al.

Extracting the relevant information via the CL

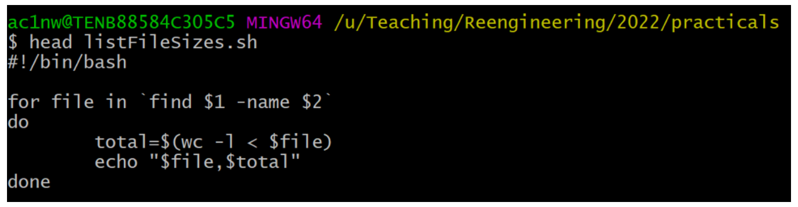
We will be doing this in the coming lab class.

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



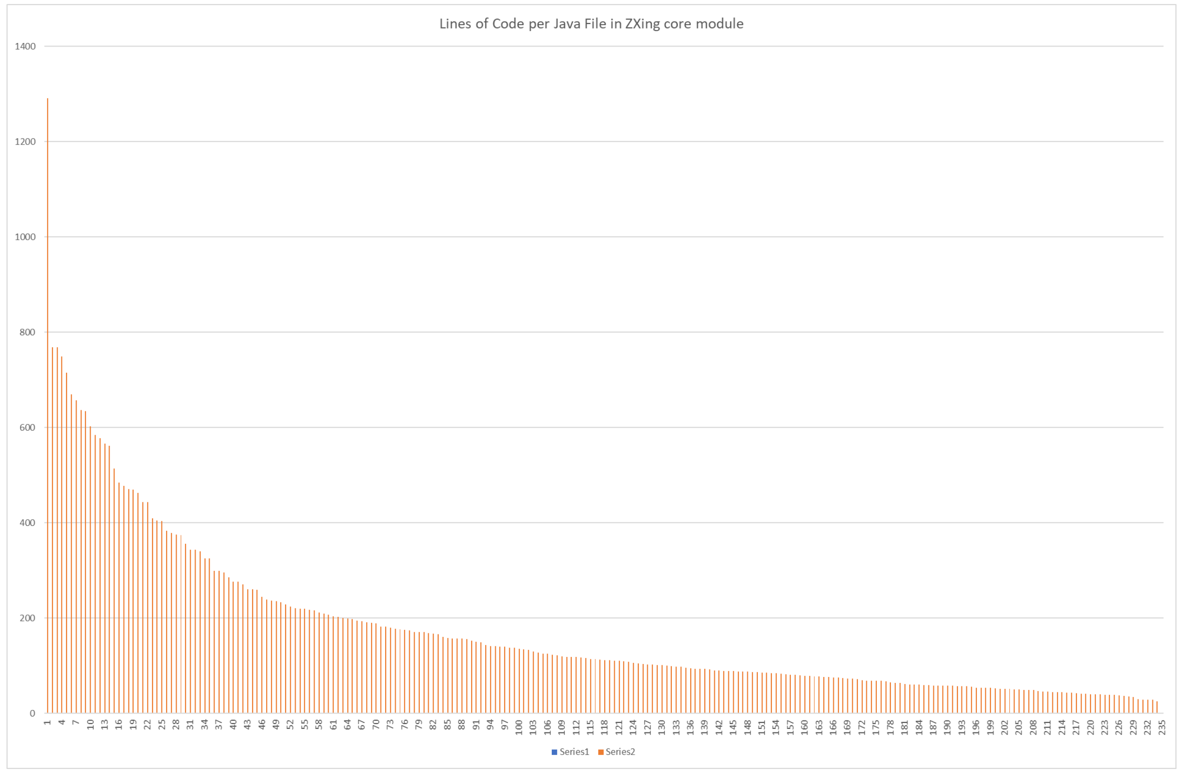
```
ac1nw@TENB88584C305C5 MINGW64 /u/Teaching/Reengineering/2022/practicals
$ find zxing/core/ -name '*.java' | wc -l
375
```

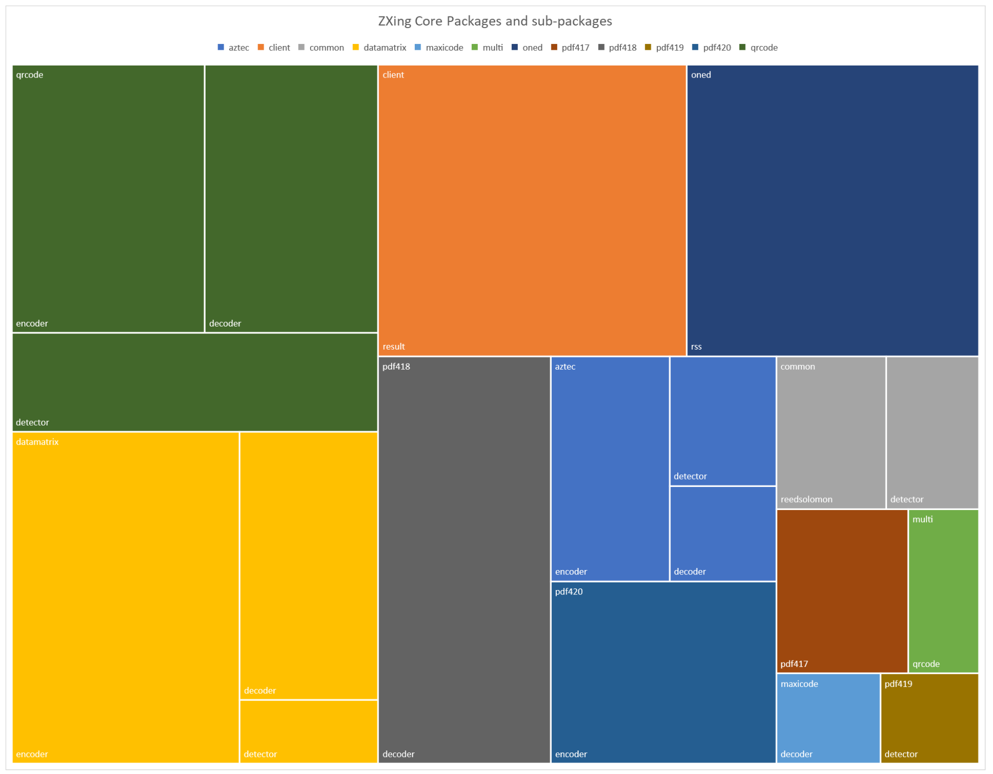
Which (source) files are the biggest one - what is the distribution?



```
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
```





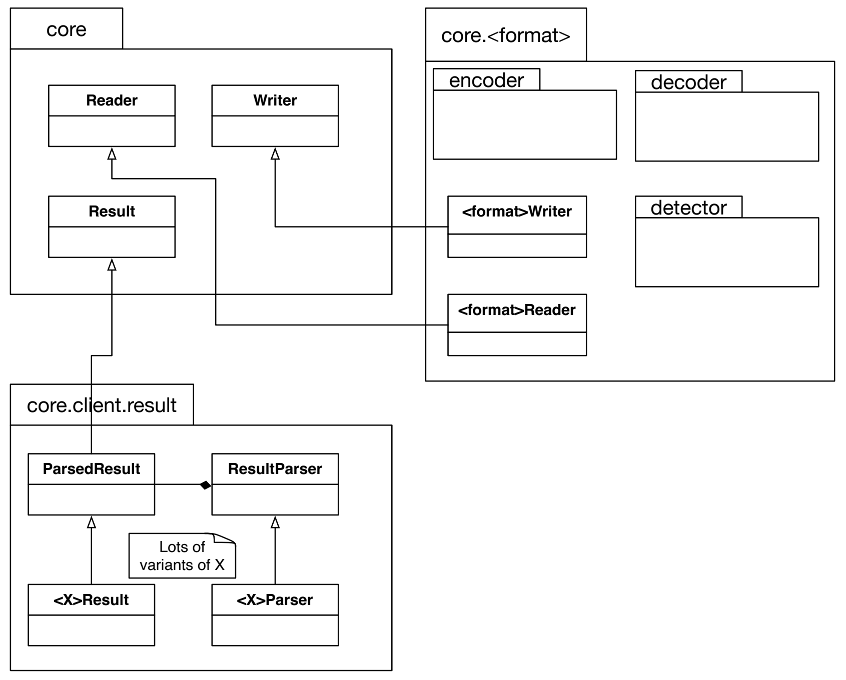
Speculating about the design

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.