Repository Analysis

Software Reengineering
(COM3523 / COM6523)

The University of Sheffield

Version Repositories

Software development can involve hundreds or thousands of developers.

- Often working asynchronously, from different parts of the globe.

Version repositories manage these changes.

Every clone of a repository includes entire history of code changes.

- **A valuable data-set for exploring the evolution of the software system.**
- **Often come with powerful command-line interfaces.**

Patches

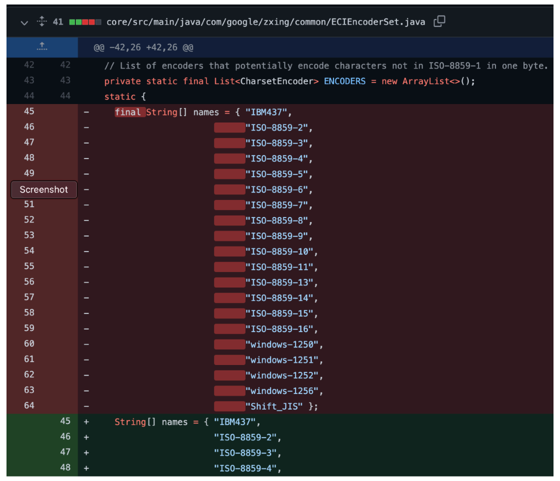
The contents of a commit in Git.

Each patch can affect one or more files.

A set of lines of code that are either added or deleted.

- A change to a line is achieved by deleting it, and adding the changed version.

Can include the creation of new files, or the removal of files.



```
41 core/src/main/java/com/google/zxing/common/ECIEncoderSet.java
@@ -42,26 +42,26 @@
    // List of encoders that potentially encode characters not in ISO-8859-1 in one byte.
    private static final List<CharsetEncoder> ENCODERS = new ArrayList<>();
    static {
-     final String[] names = { "IBM437",
-                              "ISO-8859-2",
-                              "ISO-8859-3",
-                              "ISO-8859-4",
-                              "ISO-8859-5",
-                              "ISO-8859-6",
-                              "ISO-8859-7",
-                              "ISO-8859-8",
-                              "ISO-8859-9",
-                              "ISO-8859-10",
-                              "ISO-8859-11",
-                              "ISO-8859-13",
-                              "ISO-8859-14",
-                              "ISO-8859-15",
-                              "ISO-8859-16",
-                              "windows-1250",
-                              "windows-1251",
-                              "windows-1252",
-                              "windows-1256",
-                              "Shift_JIS" };
+     String[] names = { "IBM437",
+                        "ISO-8859-2",
+                        "ISO-8859-3",
+                        "ISO-8859-4",
```

Useful information about the system

Which files do developers work on most frequently?

- Tells us which areas are particular important, or problematic.

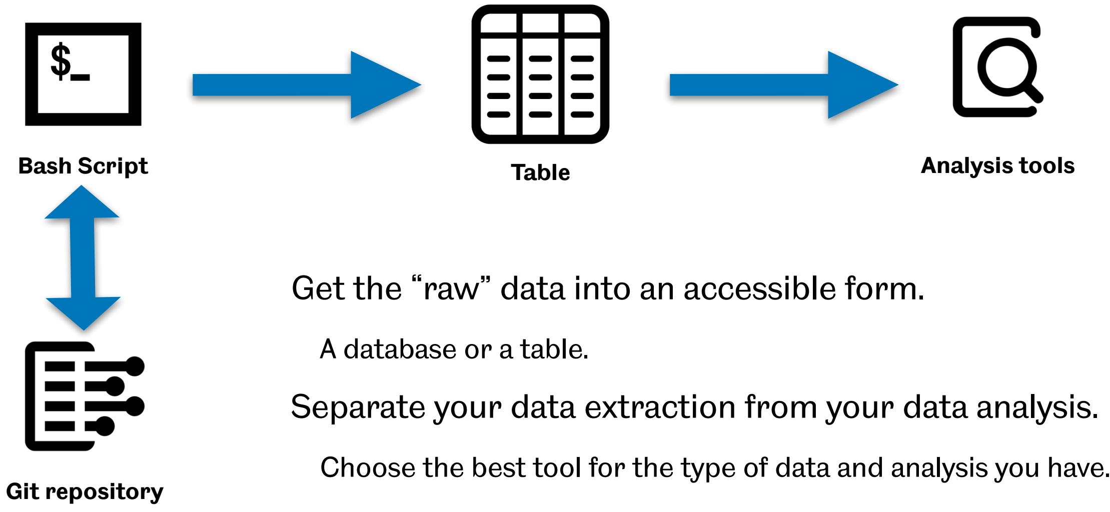
Which files were most associated with bug fixes?

- Which areas of the system are weak, perhaps need some re-design?

Which files are most frequently changed at the same time?

- Which areas are probably related to each other?

Process



Get the “raw” data into an accessible form.

- A database or a table.

Separate your data extraction from your data analysis.

- Choose the best tool for the type of data and analysis you have.

git show

`git show` will extract *any* information you need about a commit in git.

Documentation available at: https://git-scm.com/docs/git-show

Can extract a single piece of data as follows:

```
git show -s —format=‘placeholder’ commit_hash_code
git show -s —format=‘%ci’
```

Shows the date as a Unix timestamp

Can also extract statistics for numbers of lines added / removed:

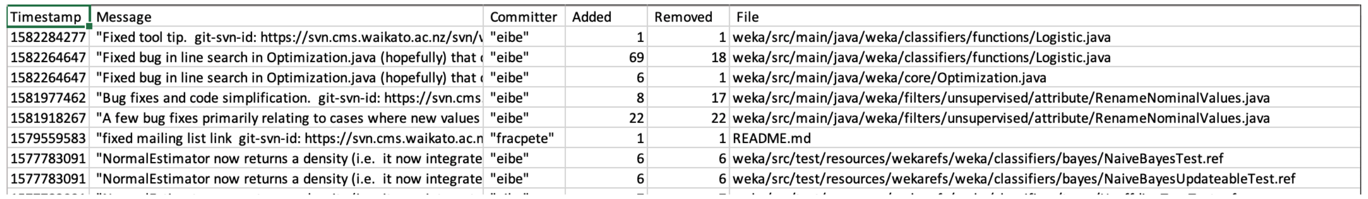
```
git show —numstat commit_hash_code
```

git show

Storage in a table

Attributes in the columns.

Each entry is a row.



Timestamp	Message	Committer	Added	Removed	File
1582284277	"Fixed tool tip. git-svn-id: https://svn.cms.waikato.ac.nz/svn/v	"eibe"	1	1	weka/src/main/java/weka/classifiers/functions/Logistic.java
1582264647	"Fixed bug in line search in Optimization.java (hopefully) that (	"eibe"	69	18	weka/src/main/java/weka/classifiers/functions/Logistic.java
1582264647	"Fixed bug in line search in Optimization.java (hopefully) that (	"eibe"	6	1	weka/src/main/java/weka/core/Optimization.java
1581977462	"Bug fixes and code simplification. git-svn-id: https://svn.cms.	"eibe"	8	17	weka/src/main/java/weka/filters/unsupervised/attribute/RenameNominalValues.java
1581918267	"A few bug fixes primarily relating to cases where new values	"eibe"	22	22	weka/src/main/java/weka/filters/unsupervised/attribute/RenameNominalValues.java
1579559583	"fixed mailing list link git-svn-id: https://svn.cms.waikato.ac.n	"fracpete"	1	1	README.md
1577783091	"NormalEstimator now returns a density (i.e. it now integrate	"eibe"	6	6	weka/src/test/resources/wekarefs/weka/classifiers/bayes/NaiveBayesTest.ref
1577783091	"NormalEstimator now returns a density (i.e. it now integrate	"eibe"	6	6	weka/src/test/resources/wekarefs/weka/classifiers/bayes/NaiveBayesUpdateableTest.ref

Summarising combinations of variables

Our “raw” CSV file is big.

- Every “atomic” change to a file has its own row.

Need to group and summarise changes to obtain useful summaries.

Lots of tools to do this - pick your favourite!

- Python - framworks such as Pandas can aggregate and summarise.
- R - Plyr, reshape2, etc.
- Excel - Pivot tables...

Key steps:

- (1) Select your “grouping” variables.
- (2) Select your “summary” operation to carry out on the grouped variables - to sum, to average, etc.

Key take-aways

Version repositories contain an extensive history of source code change.

- Can identify frequently changed files, active developers, co-changes, etc.

Can be particularly powerful when combined with other data sources.

- Information about file-sizes, speculative design documents, etc.

Often have powerful command-line interfaces.

- Relatively easy to mine with Bash scripts.

Static Analysis

Software Reengineering
(COM3523 / COM6523)

The University of Sheffield

Source Code

The **definitive record** of software structure and behaviour.

The primary component to be changed when the system is reengineered.

Source Code

The **definitive record** of software structure and behaviour.

The primary component to be changed when the system is reengineered.

Source Code

The **definitive record** of software structure and behaviour.

The primary component to be changed when the system is reengineered.

Difficult to understand because it is:

- Big - hundreds of thousands or millions of lines of code.
- Complex - highly interconnected.
- Poorly designed - having deteriorated over decades.

Source Code

The **definitive record** of software structure and behaviour.

The primary component to be changed when the system is reengineered.

Difficult to understand because it is:

- Big - hundreds of thousands or millions of lines of code.
- Complex - highly interconnected.
- Poorly designed - having deteriorated over decades.

Loaded with latent information about what the system *should* be doing.

Source Code

The **definitive record** of software structure and behaviour.

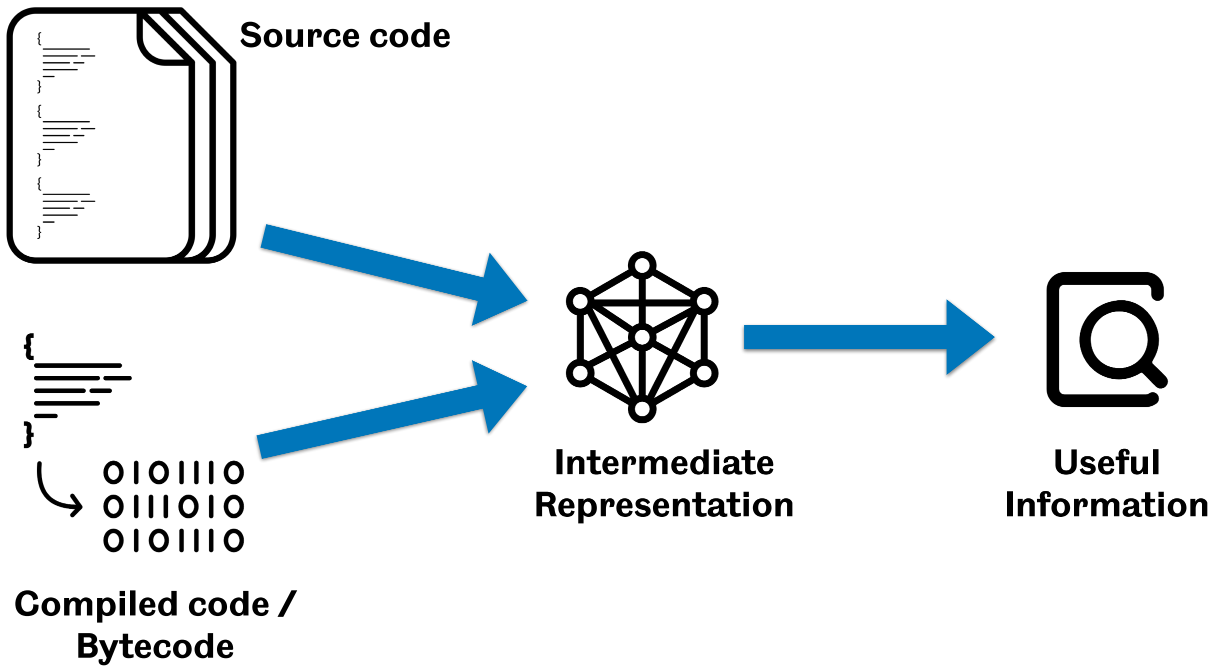
The primary component to be changed when the system is reengineered.

Difficult to understand because it is:

- Big - hundreds of thousands or millions of lines of code.
- Complex - highly interconnected.
- Poorly designed - having deteriorated over decades.

Loaded with latent information about what the system *should* be doing.

Source code analysis process



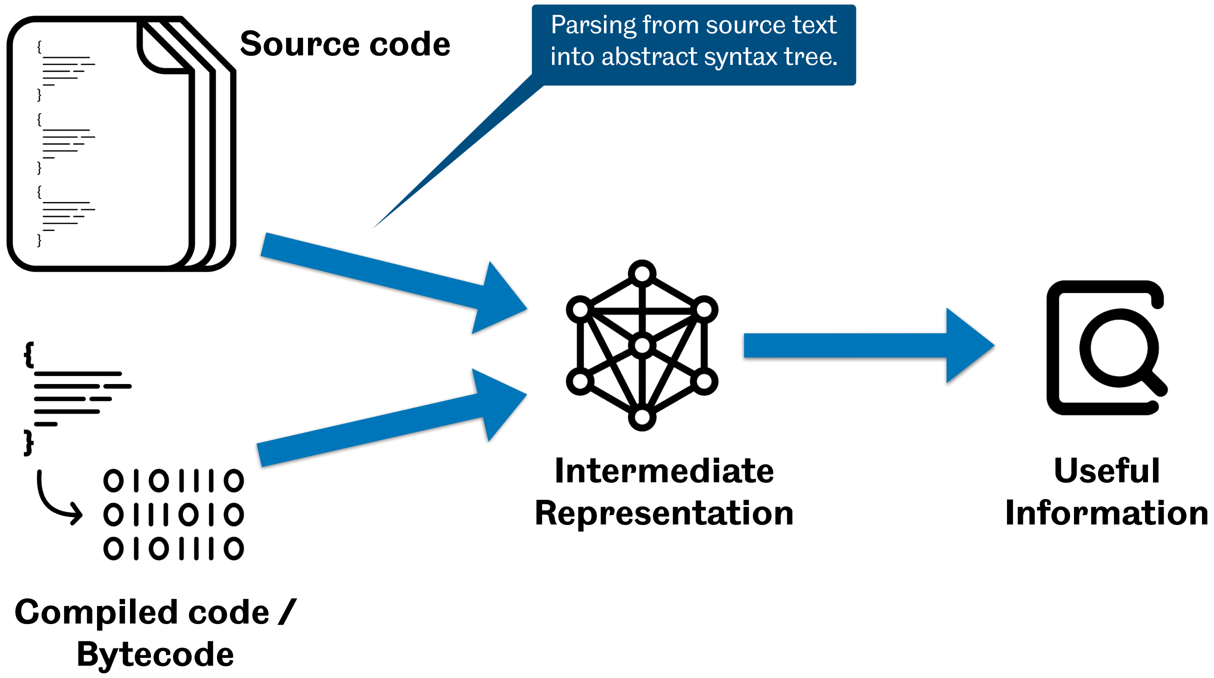
Source code

Compiled code / Bytecode

Intermediate Representation

Useful Information

Source code analysis process



Source code

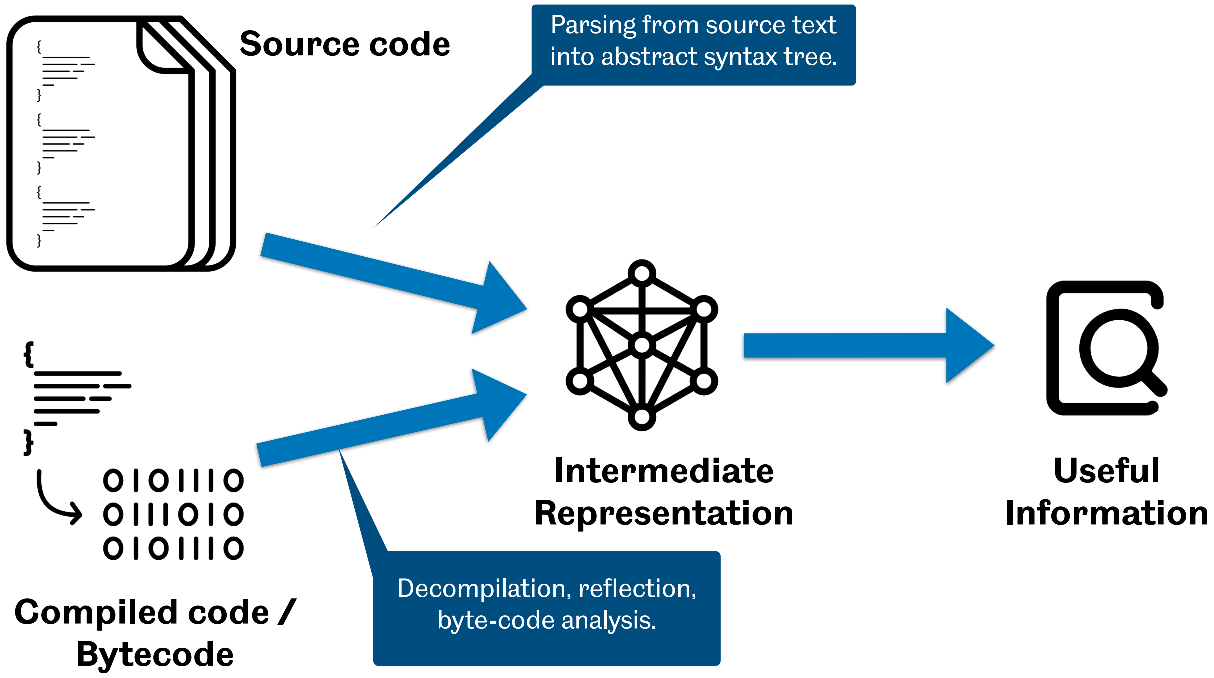
Parsing from source text into abstract syntax tree.

Compiled code / Bytecode

Intermediate Representation

Useful Information

Source code analysis process



Source code

Parsing from source text into abstract syntax tree.

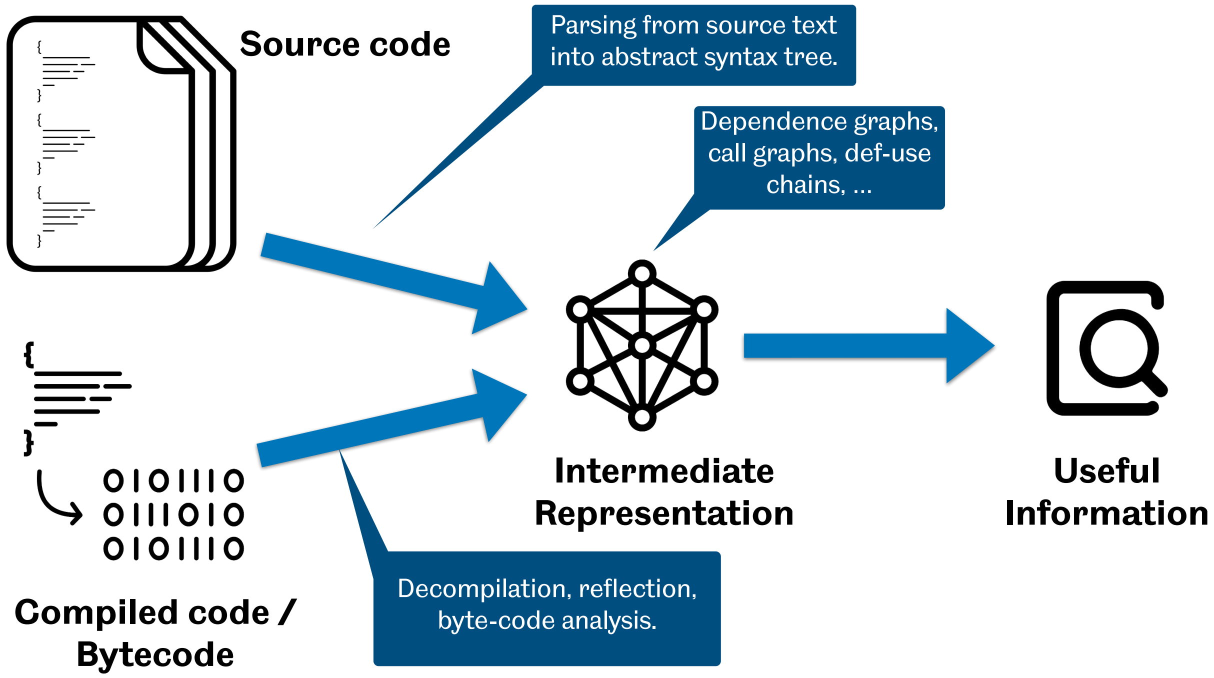
Compiled code / Bytecode

Decompilation, reflection, byte-code analysis.

Intermediate Representation

Useful Information

Source code analysis process



Source code

Parsing from source text into abstract syntax tree.

Dependence graphs, call graphs, def-use chains, ...

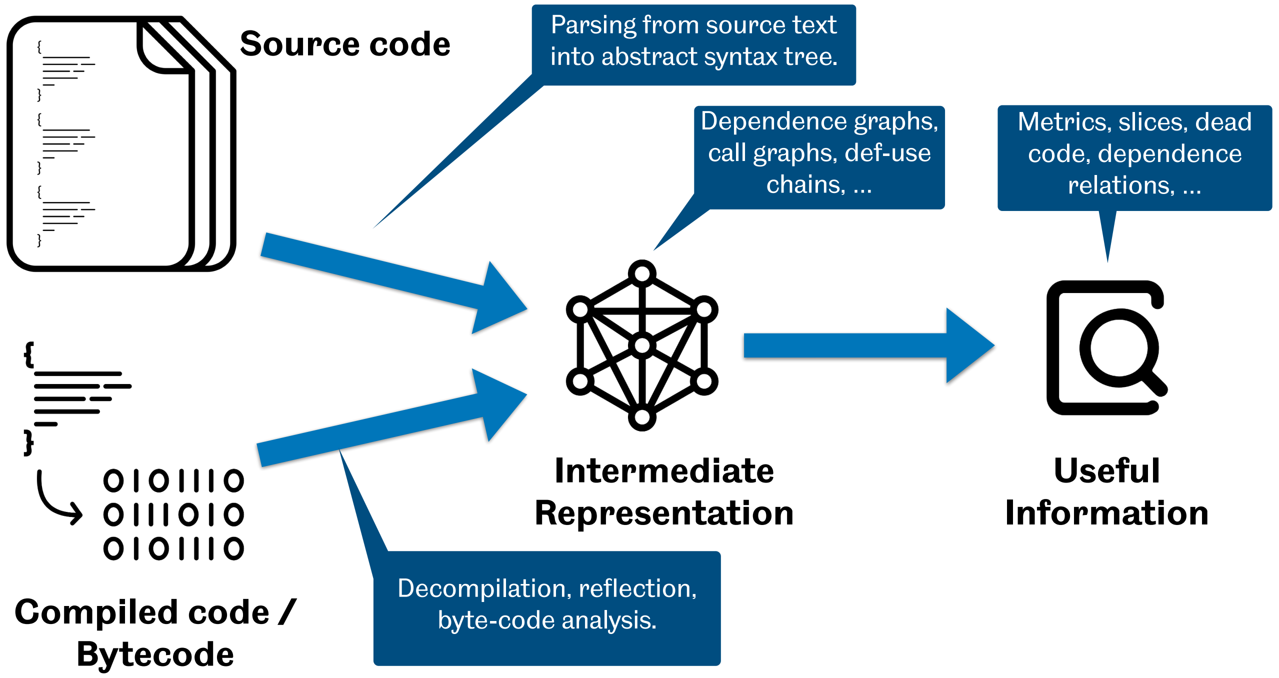
Compiled code / Bytecode

Decompilation, reflection, byte-code analysis.

Intermediate Representation

Useful Information

Source code analysis process



Source code

Parsing from source text into abstract syntax tree.

Dependence graphs, call graphs, def-use chains, …

Metrics, slices, dead code, dependence relations, …

Intermediate Representation

Useful Information

Compiled code / Bytecode

Decompilation, reflection, byte-code analysis.

Reverse Engineering Class Diagrams via Reflection

Reflection

The ability of a program to inspect and modify its own structure and behaviour.

Terrible idea to use as a primary programming mechanism.

But - very useful for debugging, inspection, hot-swapping, and reverse-engineering.

Class
getAnnotations():Annotation[]
getConstructors(): Constructor<?>[]
getDeclaredFields(): Field[]
getDeclaredMethods(): Method[]
getInterfaces(): Class<?>[]
getName(): String
getPackage(): Package
getSuper(): Class<?>
…

Some reflection methods in java.lang.Class

Reflection

The ability of a program to inspect and modify its own structure and behaviour.

Terrible idea to use as a primary programming mechanism.

But - very useful for debugging, inspection, hot-swapping, and reverse-engineering.

Lots of languages have this ability - especially dynamically-typed ones.

Java, C# (and other .NET languages), Go, Julia, Lisp, Perl, Python, R, Ruby, **Smalltalk**, …

Class
getAnnotations():Annotation[]
getConstructors(): Constructor<?>[]
getDeclaredFields(): Field[]
getDeclaredMethods(): Method[]
getInterfaces(): Class<?>[]
getName(): String
getPackage(): Package
getSuper(): Class<?>
…

Some reflection methods in java.lang.Class

Reflection

The ability of a program to inspect and modify its own structure and behaviour.

Terrible idea to use as a primary programming mechanism.

But - very useful for debugging, inspection, hot-swapping, and reverse-engineering.

Lots of languages have this ability - especially dynamically-typed ones.

Java, C# (and other .NET languages), Go, Julia, Lisp, Perl, Python, R, Ruby, **Smalltalk**, ...

Built-in to languages, easy to use.

Class
getAnnotations():Annotation[]
getConstructors(): Constructor<?>[]
getDeclaredFields(): Field[]
getDeclaredMethods(): Method[]
getInterfaces(): Class<?>[]
getName(): String
getPackage(): Package
getSuper(): Class<?>
…

Some reflection methods in java.lang.Class

Reflection

The ability of a program to inspect and modify its own structure and behaviour.

Terrible idea to use as a primary programming mechanism.

But - very useful for debugging, inspection, hot-swapping, and reverse-engineering.

Lots of languages have this ability - especially dynamically-typed ones.

Java, C# (and other .NET languages), Go, Julia, Lisp, Perl, Python, R, Ruby, **Smalltalk**, ...

Built-in to languages, easy to use.

Useful if you want structure, and don't need to know anything about instructions within a method.

Class
getAnnotations():Annotation[]
getConstructors(): Constructor<?>[]
getDeclaredFields(): Field[]
getDeclaredMethods(): Method[]
getInterfaces(): Class<?>[]
getName(): String
getPackage(): Package
getSuper(): Class<?>
…

Some reflection methods in java.lang.Class

Class Diagram

Classes are boxes.

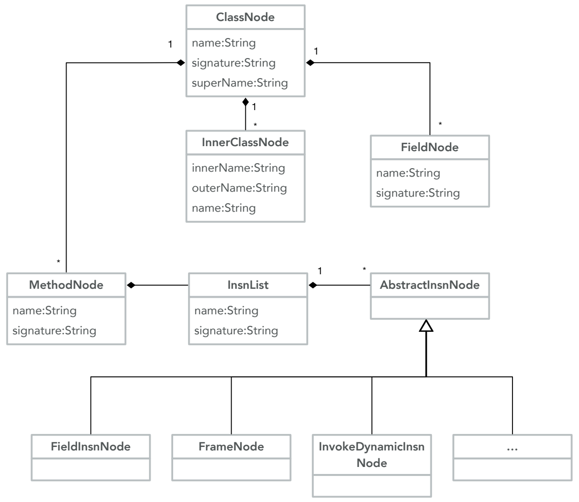
Class names at the top, field and method names below.

Edges represent associations:

Inheritance (large, hollow arrow).

Composition (a class is an attribute within another class).

All of this can be obtained via reflection.



Class Diagram Pseudocode

To create a class diagram via reflection:

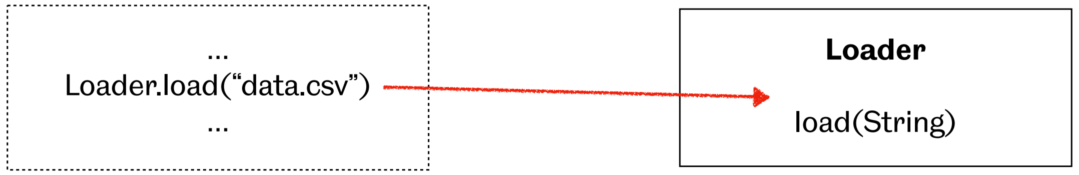
```
Iterate through classes in the system and for each class X:
Create a “class” node corresponding to X.
Load X via reflection.
For each type of relationship from X to some other class Y:
    Create a “class” node for Y if it doesn’t exist already.
    Create an edge X → Y (using the appropriate edge notation for the relationship type).
```

Reverse Engineering Class Diagrams via Decompilation / Bytecode Analysis

Call graphs

…
Loader.load(“data.csv”)
…
Loader
load(String)

Call graphs

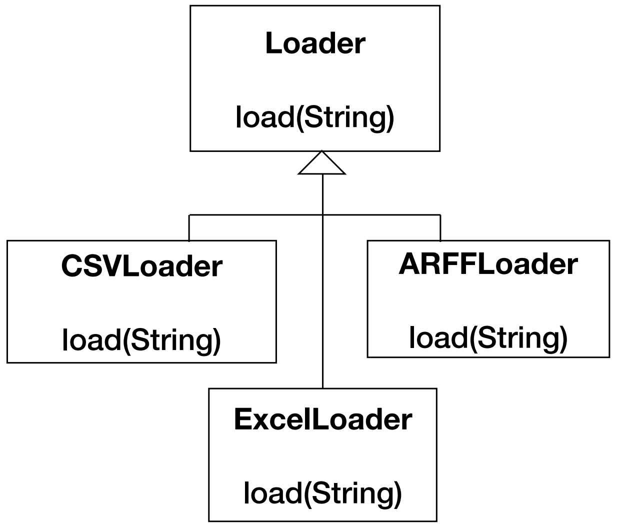


Call graphs

```
public static void main(String[]
args){
    Loader l = new CSVLoader();
    loadFile(l);
}

public void arffLoad(){
    Loader l = new ARFFLoader();
    loadFile(l);
}

protected void loadFile(Loader l){
    l.load(“data.csv”);
}
```

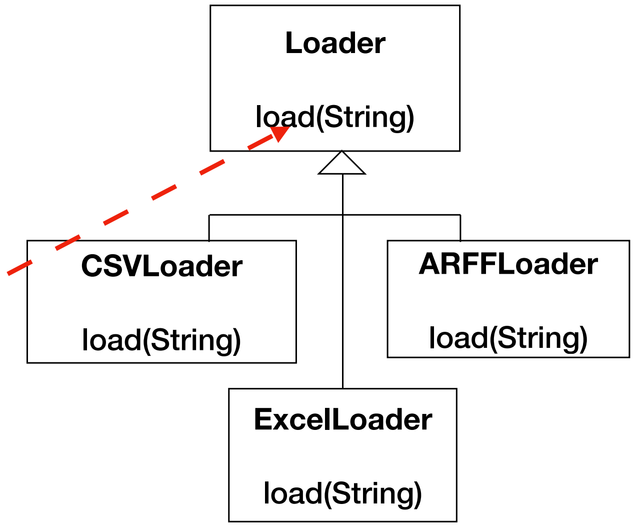


Call graphs

```
public static void main(String[]
args){
    Loader l = new CSVLoader();
    loadFile(l);
}

public void arffLoad(){
    Loader l = new ARFFLoader();
    loadFile(l);
}

protected void loadFile(Loader l){
    l.load("data.csv");
}
```

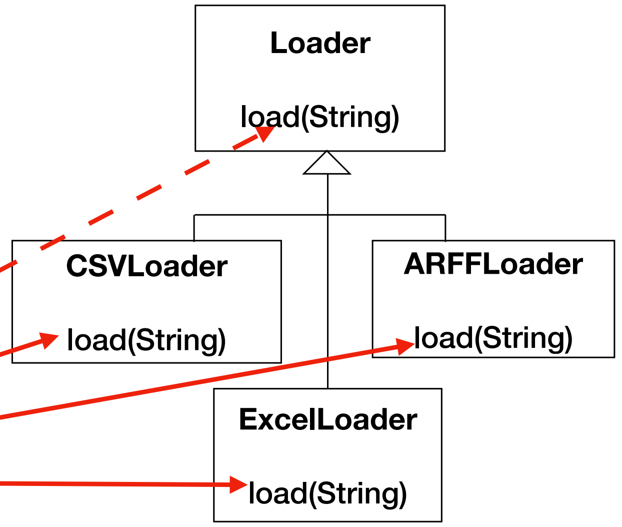


Call graphs

```
public static void main(String[]
args){
    Loader l = new CSVLoader();
    loadFile(l);
}

public void arffLoad(){
    Loader l = new ARFFLoader();
    loadFile(l);
}

protected void loadFile(Loader l){
    l.load("data.csv");
}
```



Points-To Analysis

Identify the possible destination(s) of a reference.

Lots of possible algorithms.

- Tend to trade-off efficiency against accuracy.

Class Hierarchy Analysis (CHA)

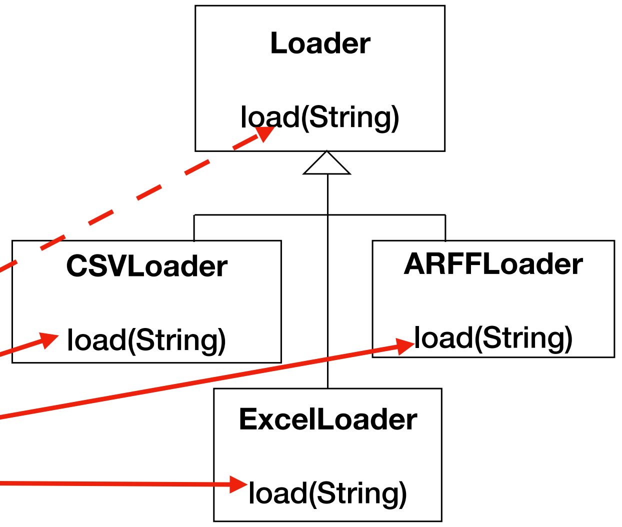
- For any class that is the target of a call, identify any sub-classes with overriding methods.
- Make these methods potential targets.

Call graphs

```
public static void main(String[]
args){
    Loader l = new CSVLoader();
    loadFile(l);
}

public void arffLoad(){
    Loader l = new ARFFLoader();
    loadFile(l);
}

protected void loadFile(Loader l){
    l.load("data.csv");
}
```



The “Fan-In” and “Fan-out” metrics

Call graph can be used to analyse inter-dependencies within a system.

Can be used to quantify this interconnectedness via **metrics**.

Fan-in:

- Number of incoming calls to a method or a class.
- Provides an idea of how “critical” or “useful” a class or method is.

Fan-out:

- Equivalent of fan-in with outgoing edges.

Can be computed from the call graph:

- For a method - number of incoming call edges!
- For a class - sum of number of incoming edges for all methods, where source of the edge lies in a different class.

Static analysis is conservative

Returns *everything* by default.

- Every single class or method in a system.
- Every single *potential* call (even calls that are infeasible in practice).

How useful is a class diagram with >700 classes?

Key strategies:

- For visual outputs (e.g. class diagrams) - **focus** on specific packages / classes.
- For non-visual outputs (e.g. call graphs) - summarise data into key metrics.

Key take-aways

Two useful technologies for source code analysis: Reflection and Bytecode analysis.

Reflection is useful for structural analysis - e.g. class diagrams.

Byte code analysis is more useful for detailed analysis - e.g. call graphs.

Overarching challenge: Information overload - static analysis is conservative.