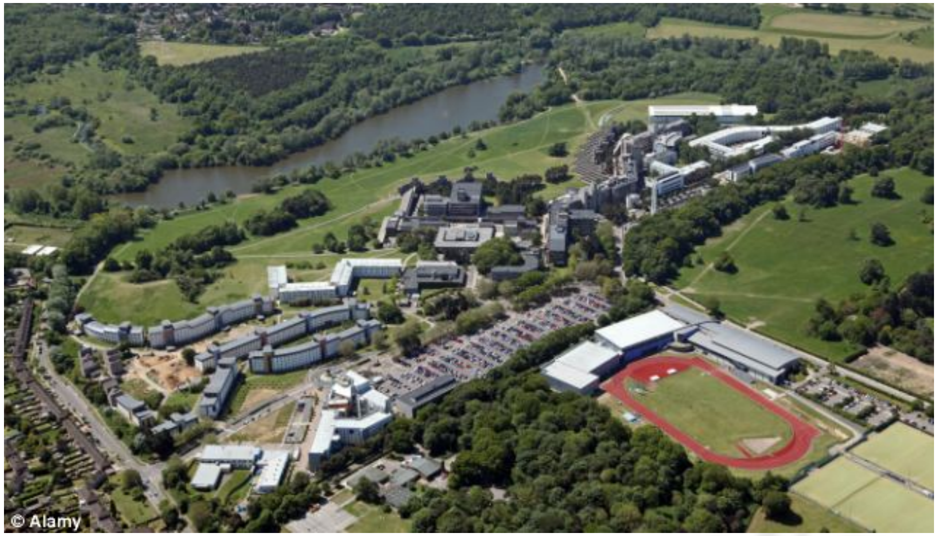
Mission Invitée supported by IUT Nantes

Traceability Forensics: Identifying Semantic Relations in Legacy Software Systems

Joost Noppen
University of East Anglia

A Short Introduction



Joost Noppen

Lecturer in
Software Engineering

University of
East Anglia
Norwich, UK

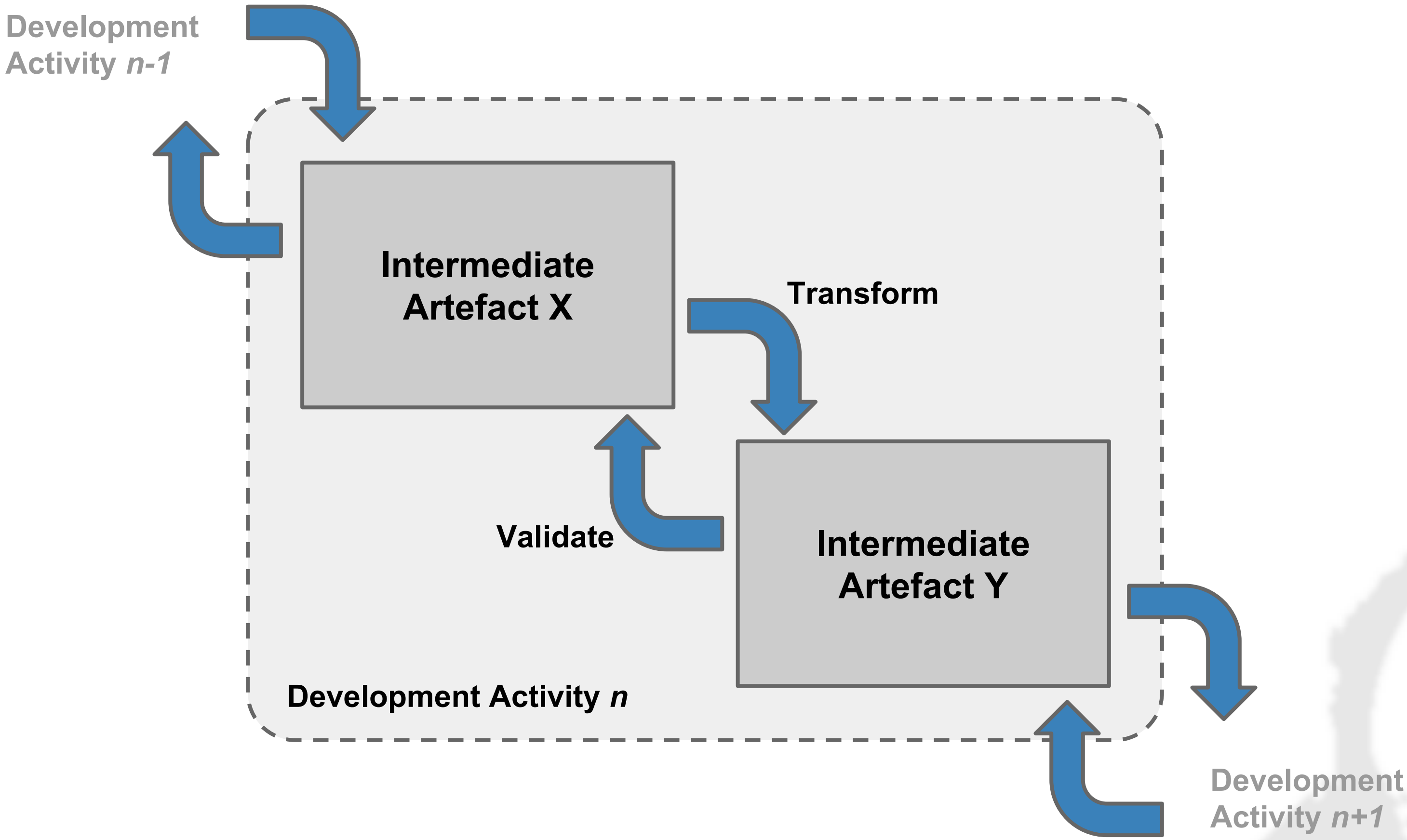
j.noppen@uea.ac.uk

A Short Introduction

Research at the School of Computing Sciences at UEA

Research is divided into three laboratories:
- Computational Biology
- Machine Learning and Statistics
- Graphics, Vision and Speech

Software Engineering is a recent addition:
- Started in 2011
- Aligned with the Machine Learning and Statistics Group
- Currently consisting of 4 members (1 lecturer, 3 PhD students)
- Related research being done in systems analysis and usability



The overall interest and goal at UEA is to create systematic analysis methods that can offer accurate support for these activities, realised in software tools.

Software Engineering Research at UEA

Decision Support for Software Engineering

Our research focusses on decision support approaches for software development activities, such as architecture analysis, change impact, fault prediction, etc.

Our approaches use historical data and mathematical analysis approaches to achieve this:

- Historical data mining and pattern recognition
- Probabilistic and Fuzzy models to represent uncertainty
- Goal functions and optimisation methods to determine the best option to choose
- Realised in software tools that can be used by developers

Forensic Analysis of Legacy Software Systems

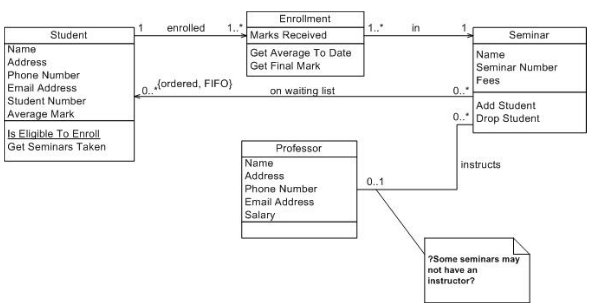
Modern software systems remain in service for a long time. In most cases their creators are gone before the systems themselves.

Green field Software Engineering does not really exist in the real world...

Developing in the real world typically involves building on existing software systems …

- … that have complex interactions with other software systems …
- … that are built with technology that no longer is in fashion …
- … that have limited documentation and detail …
- … that have designers that have long left the company ...
- … that were never designed to do the things you want to do
- … but have become too important and costly to redevelop or replace.

Forensic Analysis of Legacy Software Systems



The Challenge of Understanding Legacy Software Systems (with David Cutting)

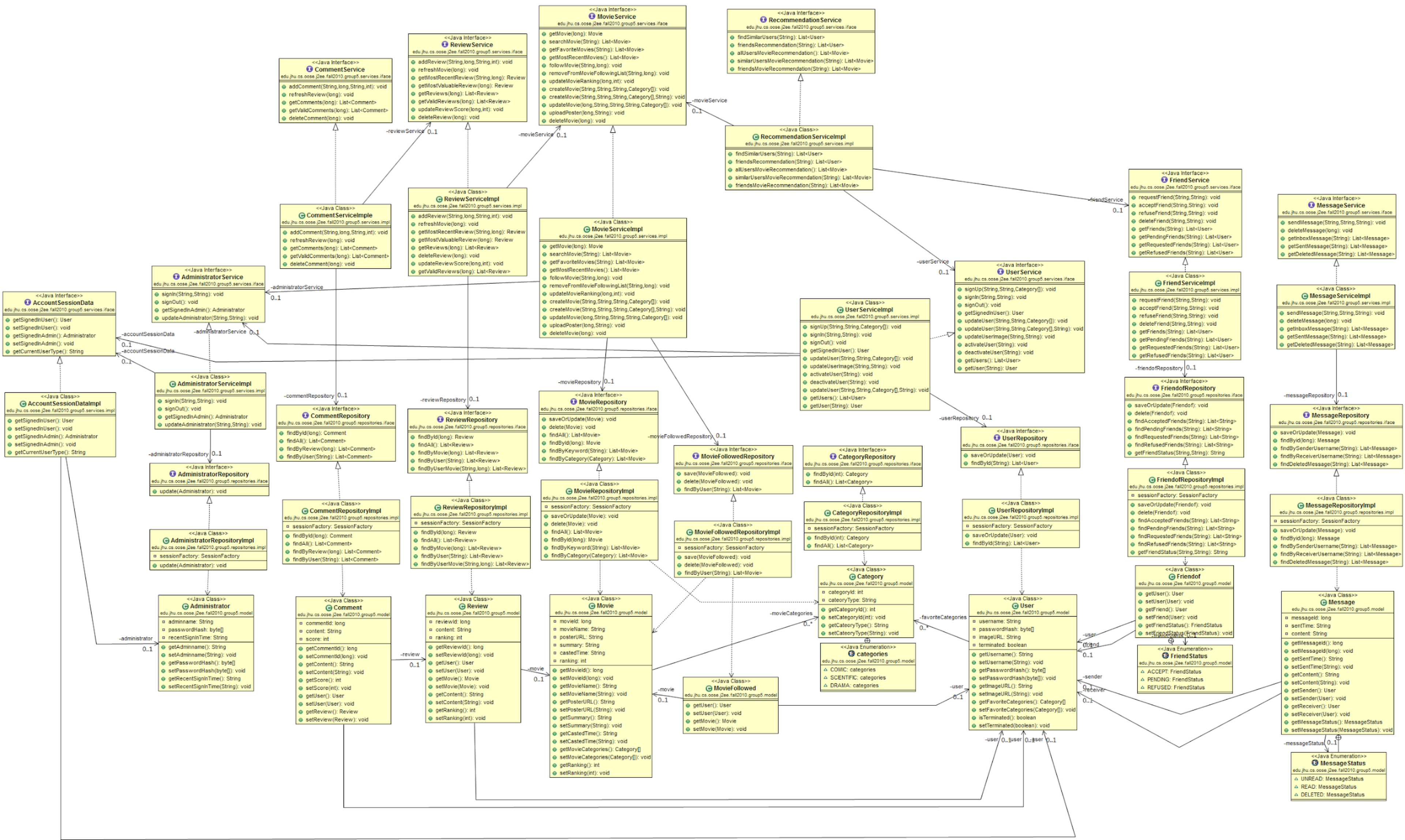
Legacy systems can be considerable in size, up to thousands of classes or more. Often they are without documentation to speak off, and no experts to consult

Traditional reverse engineering tries to reconstruct a class diagram from source code. At best this leaves the developer with a massive class diagram of potentially thousands of classes.

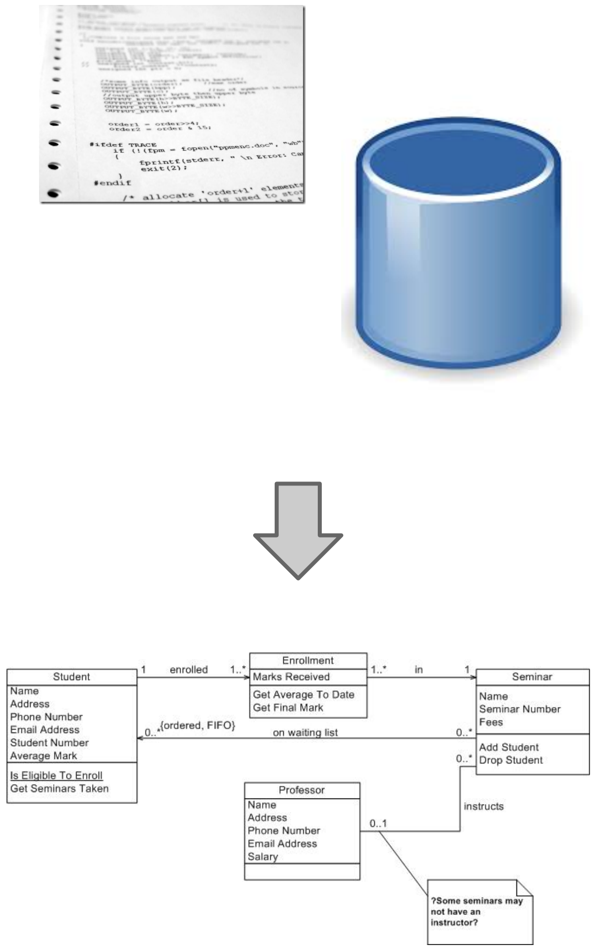
Which of these belong to the same semantic concepts (e.g. MVC, or related model classes)?

And how do these classes correlate to the requirements description we might have for this?

Forensic Analysis of Legacy Software Systems



Forensic Analysis of Legacy Software Systems



Automatic Identification of Semantic Concepts

Sometimes classes are related but they do not have explicit relations in a class diagram.

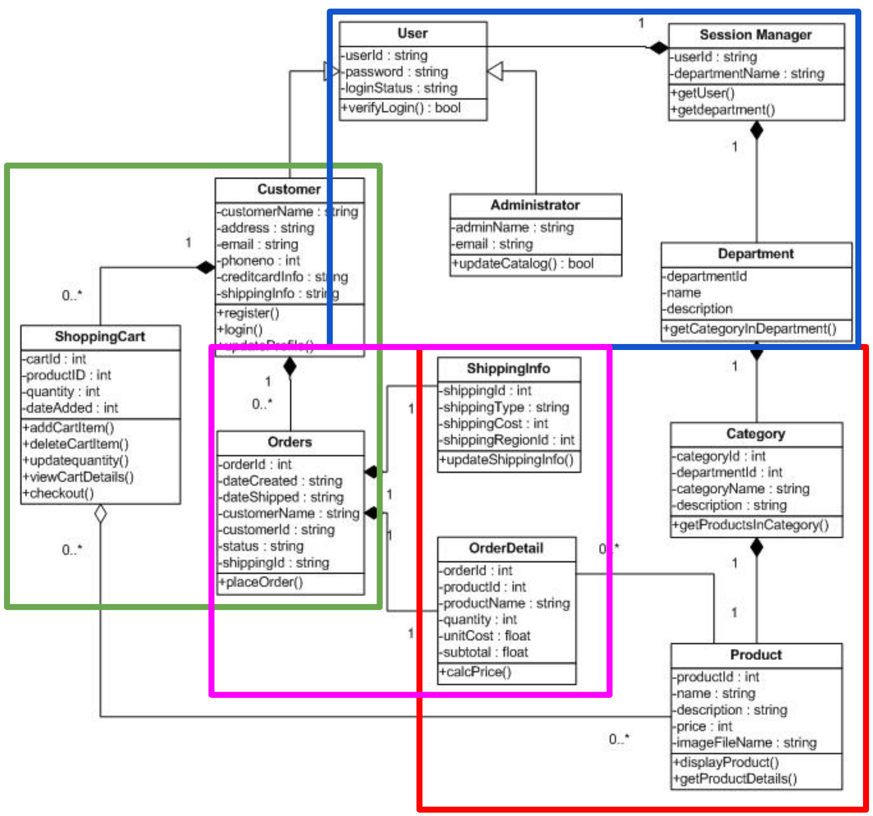
Rather than structural, the relations are *semantic*, ie they are the result of the intention of classes

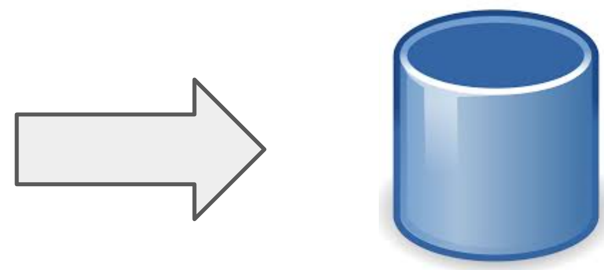
These semantic relations are as useful as structural ones, for example to identify all classes related to the GUI

Another scenario is in case of determining change impact. Classes without a specific relation in a class diagram can frequently collaborate to perform a specific task (share a semantic concept). Pure structural reasoning would miss this out.

We try to extract this information from
Version Management Systems

Forensic Analysis of Legacy Software Systems





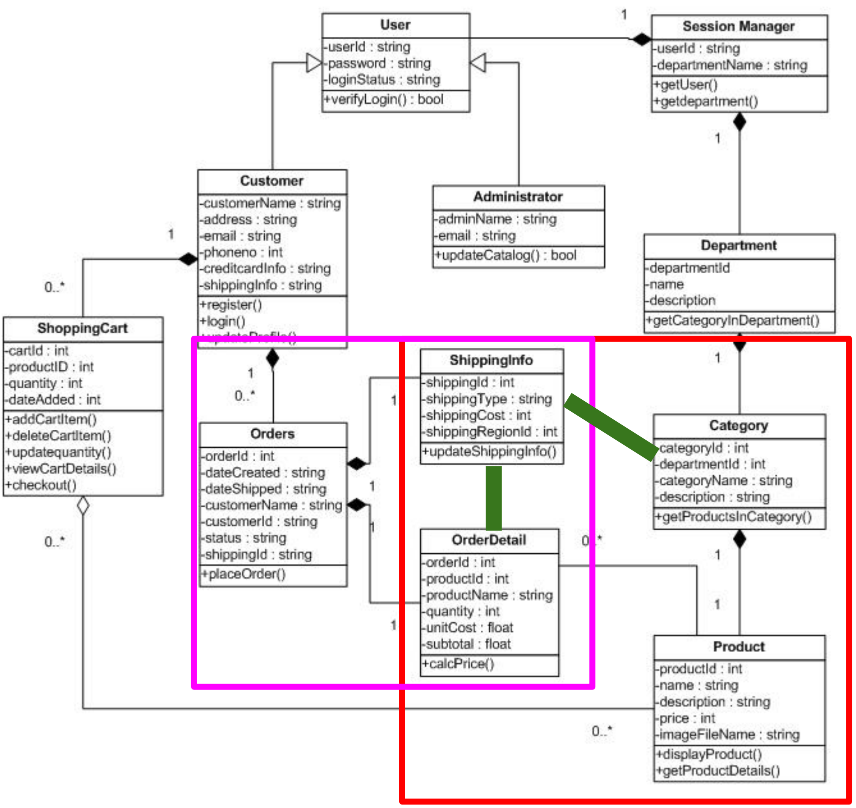
Repositories capture human behaviour

The classes that developers commit in a single transaction are related to the problem they are working on.

If classes are committed together more frequently than not, there is a semantic relation there. We call this a *co-commit*.

If a set of classes are co-committed frequently in a short period of time, this suggests a semantic relation even more.

Forensic Analysis of Legacy Software Systems



Quantifying the Relations

Every pair of classes will have a different frequency of co-committence.

We model this by attaching every relation between classes with a number on the domain [0..1].

Relations in the (reverse engineered) class diagram always have a value of 1, which we call *ground truth*.

The value attached to detected semantic relations between classes from repositories is calculated using the co-commit data harvested:

$$\frac{\textit{\#Co-commits of C1 and C2}}{\textbf{\textit{Max}}\textit{(\#Commits of C1, \#Commits of C2)}}$$

Forensic Analysis of Legacy Software Systems

	1	2	3	4	5	6	7	8	9	10	11
1. User	1	**1**		**1**		**0.5**					
2. SessionManager		1	**0.2**		**1**				**0.3**		
3. Customer			1			**1**	**1**				
4. Administrator				1			**0.6**				
5. Department					1					**1**	
6. ShoppingCart						1		**0.4**			**1**
7. Orders							1	**1**	**1**		
8. ShippingInfo								1	**0.7**	**0.8**	
9. OrderDetail									1		**1**
10. Category										1	**1**
11. Product											1

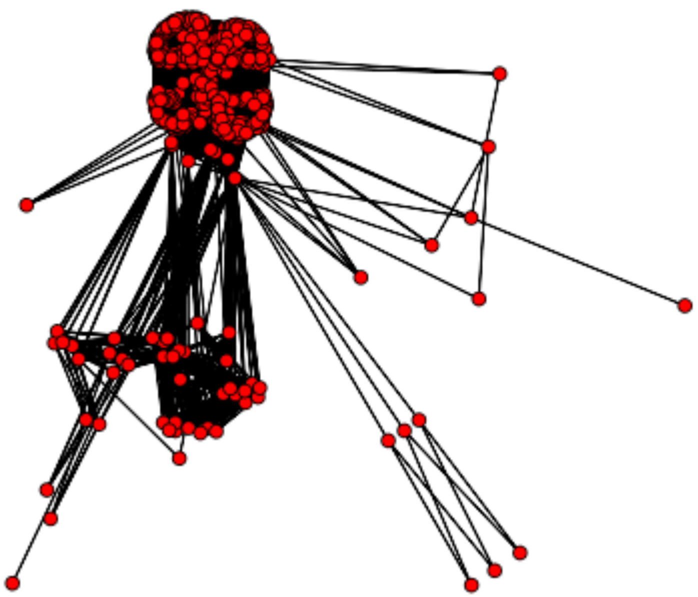
Relationship Matrix

We can now describe the full set of class relations using a relation matrix.

This matrix combines the ground truth of the structural diagram with the detected semantic relations.

This matrix now is akin to similarity matrices used for clustering in pattern recognition.

Forensic Analysis of Legacy Software Systems



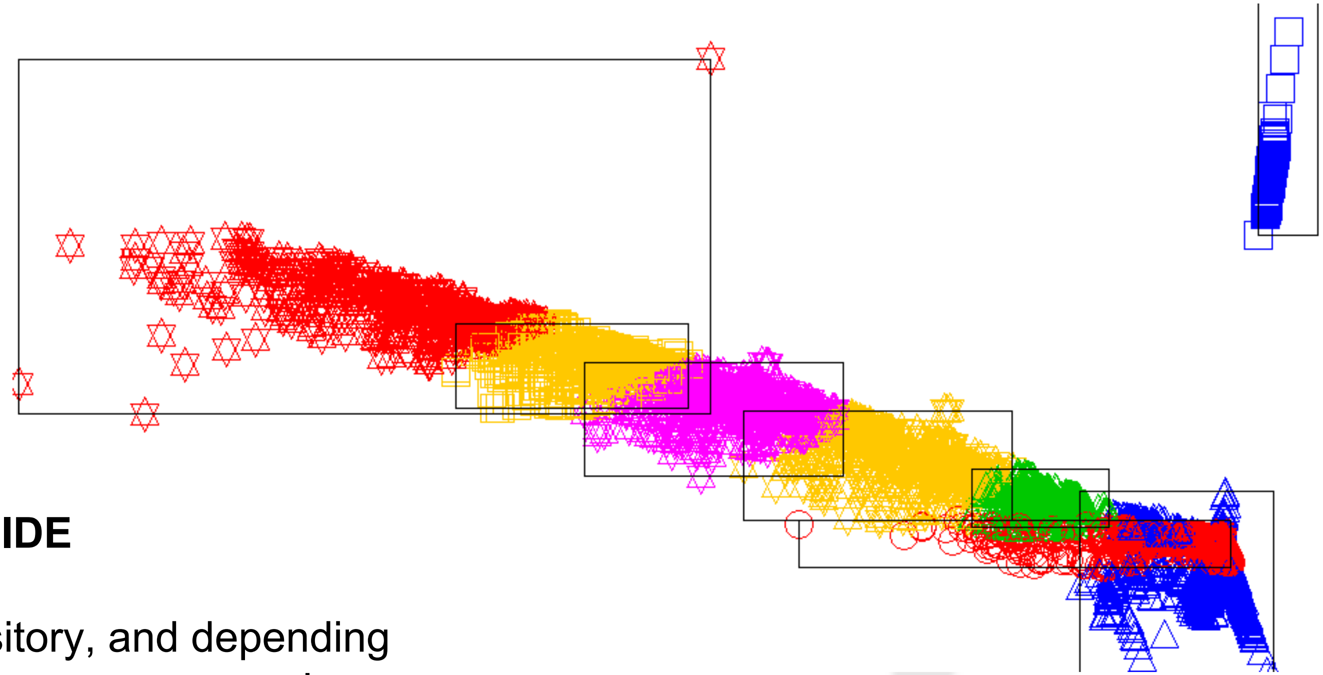
Clustering the D-UEA-ST System

When we group together the classes with the strongest relationships using clustering algorithms, we can generate graphs such as the one on the left.

Interesting aspects and semantic concepts can already be observed:

- A large cloud of classes clumped that represent the initial commit
- Another large clump is an extension by an MSc project student
- Six structured classes are the GUI abstraction mechanism

The last two definitely are semantic concepts that are hard to spot with a diagram only.

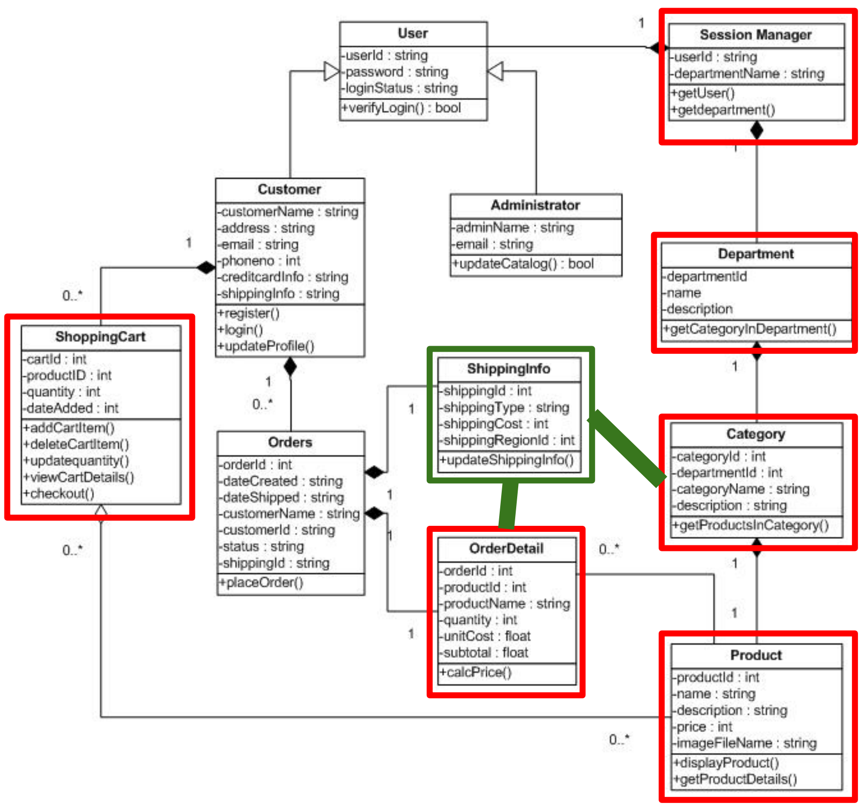


Clustering the Eclipse IDE

A far more mature repository, and depending on the scope you choose, you can see clear semantic concepts emerge:

- The progression of colours are the various releases (not highlighted in the repository itself)
- The blue element at the right top is the compile, the most stable element

Forensic Analysis of Legacy Software Systems



Change Impact Analysis

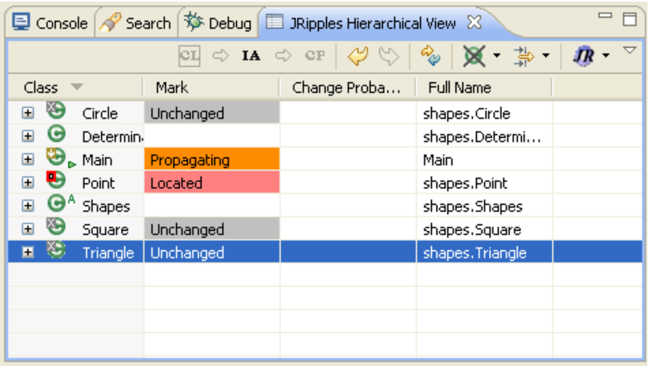
Used to determine how changing one class affects the rest of the system.

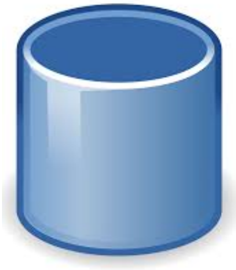
Traditional approaches rely on structural information combined with source code analysis to determine the affected areas.

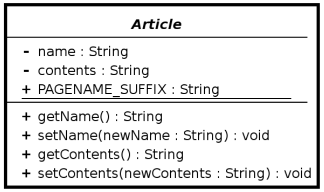
This typically creates a flooding algorithm that can miss out on semantic relations.

We have combined traditional flooding algorithms with our semantic relation information, and this improves change impact analysis.

Forensic Analysis of Legacy Software Systems



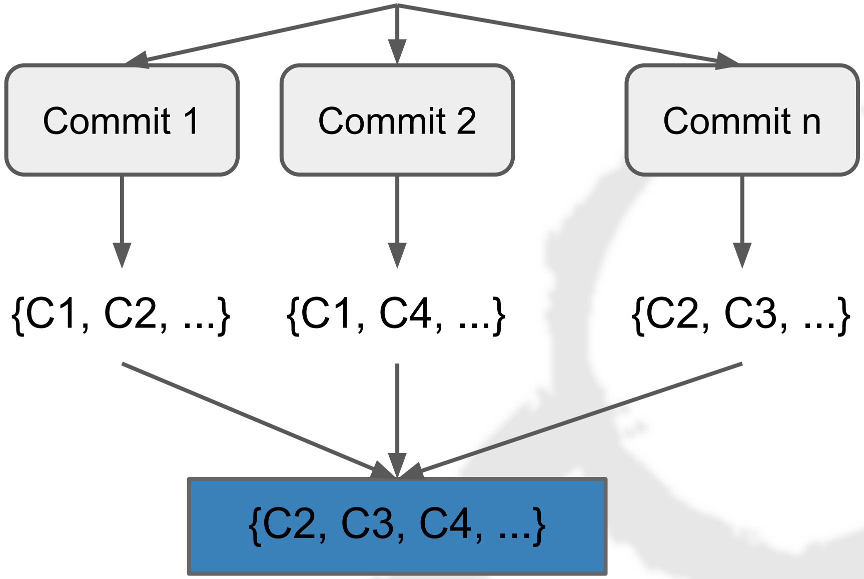


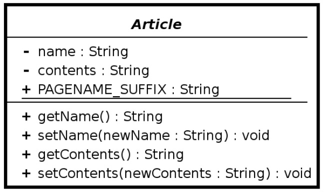


Change Impact Analysis Evaluation

To assess the effectiveness of our approach, we have compared performance to JRipples, one of the most well known CIA tools.

As we needed independent data to run the tools on, we leveraged our knowledge of repositories.





Forensic Analysis of Legacy Software Systems

Base: {C2, C3, C4, ...}

Approach: {C2, C4, C5, ...}

Precision: $$\frac{\#(\text{Approach} \cap \text{Base})}{\#\text{Approach}}$$

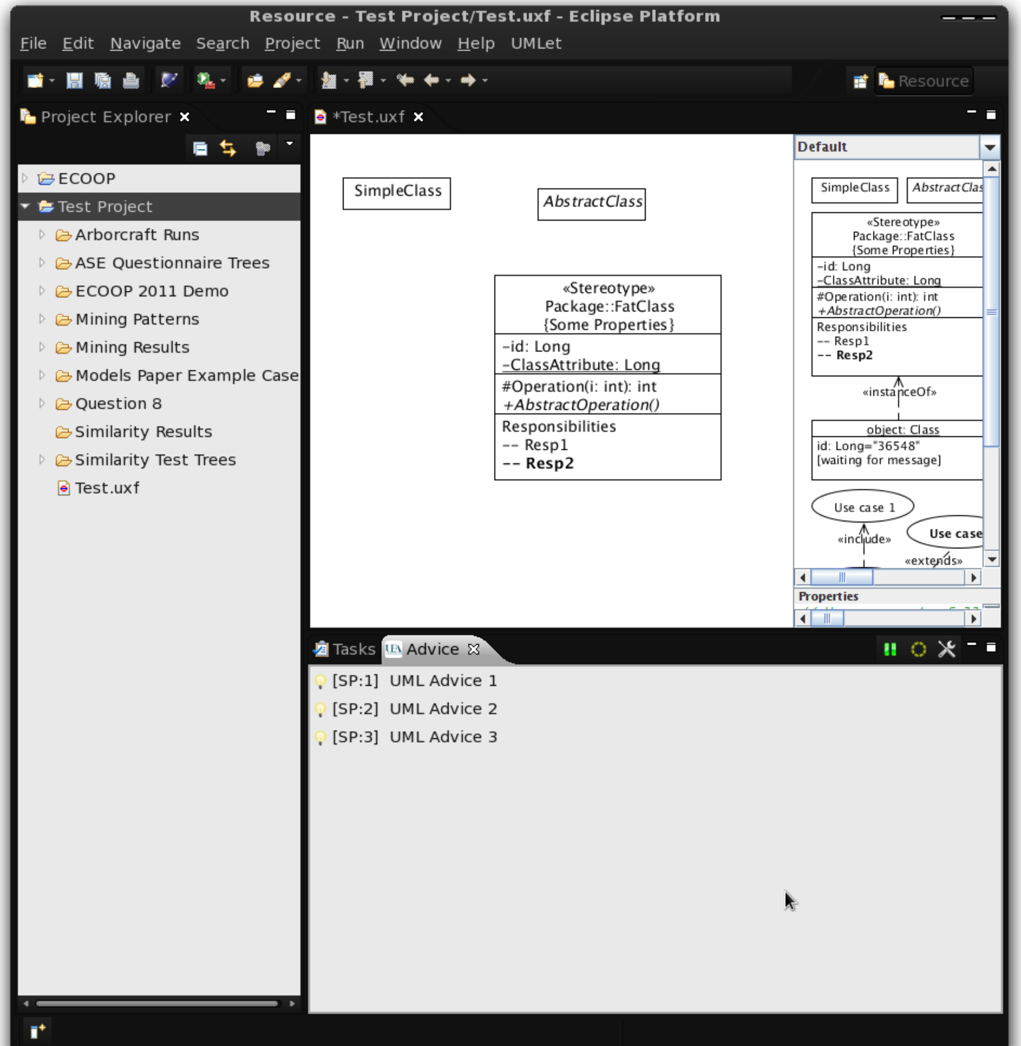
Recall: $$\frac{\#(\text{Approach} \cap \text{Base})}{\#\text{Base}}$$

Early Evaluation Results

For every case based on industrial software repositories so far:

- Our approach finds at least all classes identified by JRipples in *Base*
- This suggests our approach has an equal or better recall than JRipples
- However, our approach tends to have a somewhat lower precision than JRipples
  - Adjusting semantic relation threshold
  - Considering this in flooding algorithm
- Without the need for analysing source code

Forensic Analysis of Legacy Software Systems



Implementation

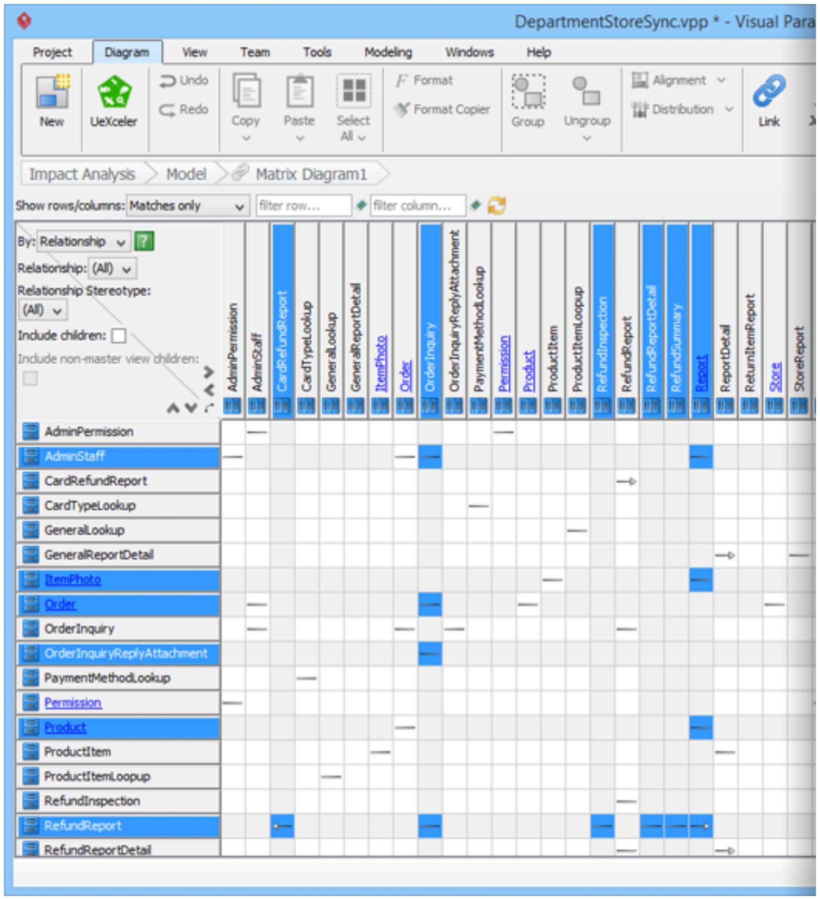
Our approach is implemented on the D-UEA-ST platform, a software toolkit developed at UEA.

Runs as an extension for Eclipse and can interact with git-based repositories and most modern reverse-engineering tools using XMI and XMI-transformers.

Git harvesting, flooding, and clustering can be performed locally or on linux-based supercomputer clusters

Completely open source and freely available via bitbucket (... very soon!)

Future Work



Change Impact Analysis

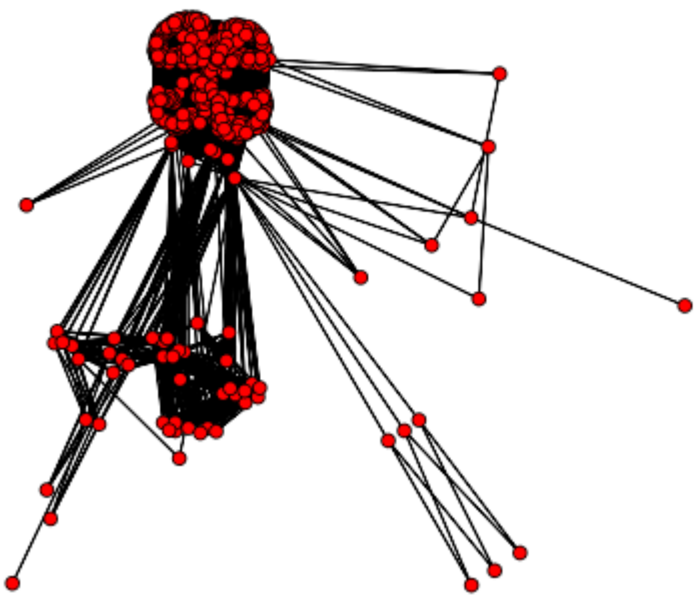
Determine the best set of operation parameters, such as thresholds and repository filters.

Complete full evaluation and publish

Include additional information sources, such as call graphs or expert opinion.

Support various levels of change impact, such as methods and attributes, or architectures and components.

Future Work



Semantic Concept Identification

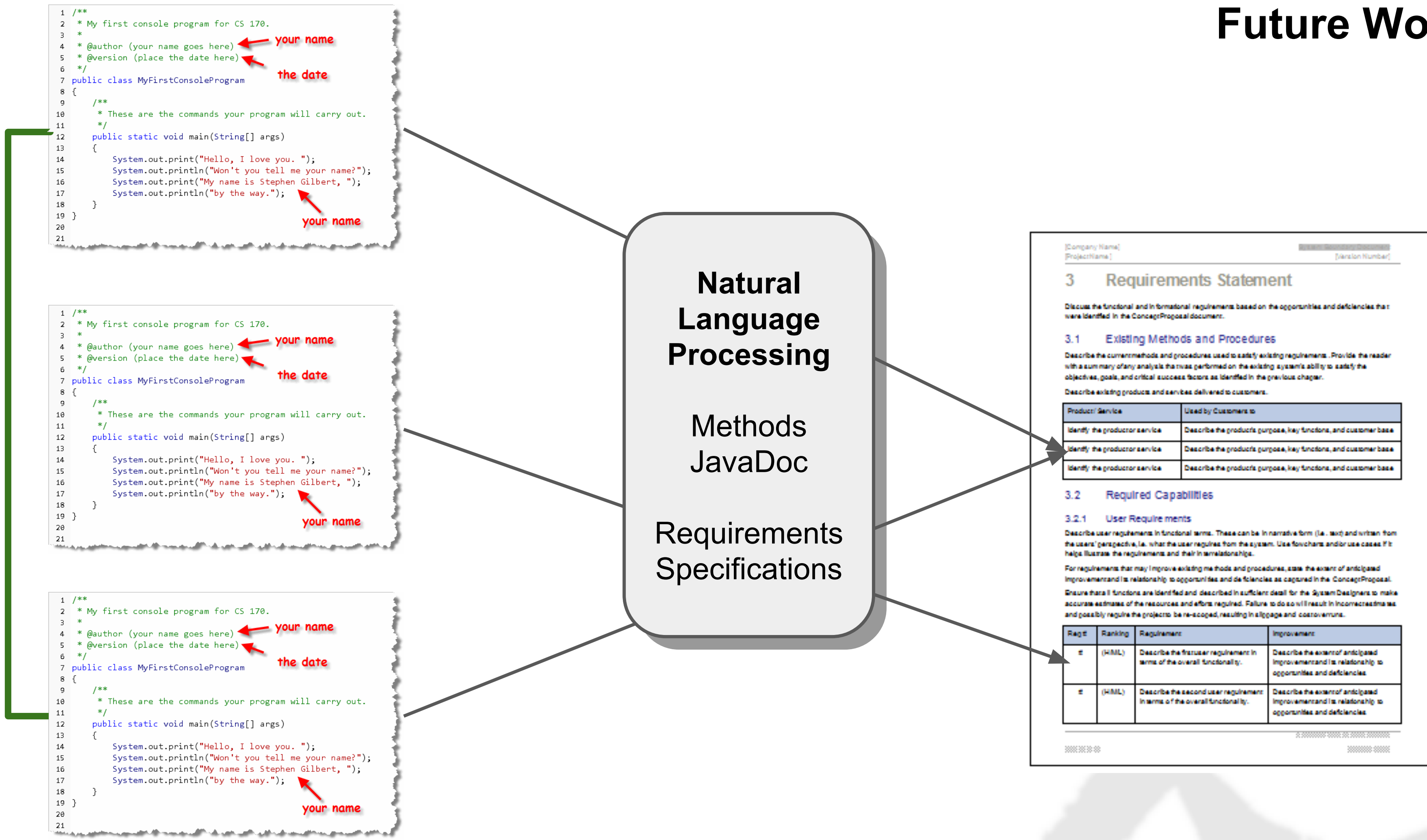
Determine the best set of operation parameters, such as thresholds and repository filters.

Include additional information sources, such as domain models or expert opinion.

Support various levels of concept identification, architecture, component, etc.

Evaluation

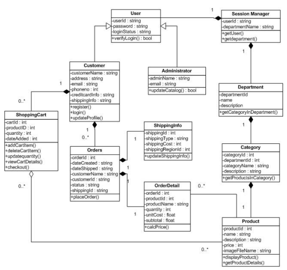
Future Work



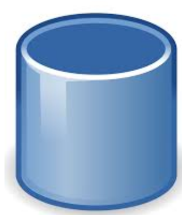
/**
* My first console program for CS 170.
*
* @author (your name goes here)
* @version (place the date here)
*/
public class MyFirstConsoleProgram
{
/**
* These are the commands your program will carry out.
*/
public static void main(String[] args)
{
System.out.print("Hello, I love you. ");
System.out.println("Won't you tell me your name?");
System.out.print("My name is Stephen Gilbert, ");
System.out.println("by the way.");
}
}
your name
the date
your name
Natural Language Processing
Methods
JavaDoc
Requirements Specifications
3 Requirements Statement
3.1 Existing Methods and Procedures
3.2 Required Capabilities
3.2.1 User Requirements

Future Work

A Flexible Base Relation Structure



	a	b	c	d	e
a	1	**1**		**1**	
b		1			**1**
c			1		
d				1	
e					1



	a	b	c	d	e
a	1	**0.3**		**0.9**	
b		1	**0.2**		**0.8**
c			1	**0.6**	
d				1	
e					1

A+B-AB

	a	b	c	d	e
a	1	**1**		**1**	
b		1	**0.2**		**1**
c			1	**0.6**	
d				1	
e					1

Conclusions

Software Engineering Research at UEA

- A relatively young group, but well underway in making UEA a centre for Software Engineering research in the UK
- Research focus is on computational approaches for supporting software development
- Working on prominent areas such as legacy system analysis, software product lines and software architectures.
- Combining techniques from fuzzy set and probability theory, pattern recognition, data mining and optimisation theory.

Traceability Forensics Research

- A flexible and generic framework for capturing structural and semantic relations between software artefacts of legacy systems
- Full support for relation harvesting from version management repositories
- Successful application to Change Impact Analysis with competitive results
- Promising first results with concept identification using clustering

Potential for Collaboration

Design Decision Optimisation

- Automated reasoning about design decisions
- Capture and analyse uncertainty
- Insight into quality and functionality trade-offs
- Variety of levels
  - Process management
  - Architectures
  - Implementation

Natural Language Processing

- Documentation analysis and design
- Model similarity
- Software trace reconstructions

Data Mining/Pattern Recognition

- Leverage historical data in design
- Infer and predict faults in software systems
- Reaction models for self-aware systems