

Mission Invitée supported by IUT  
Nantes



---

# Traceability Forensics: Identifying Semantic Relations in Legacy Software Systems

Joost Noppen  
University of East Anglia

---

Market-driven approach based on Monkey decision  
... for optimal use of resources in software  
development



# A Short Introduction



**Joost Noppen**

**Lecturer in  
Software Engineering**

University of  
East Anglia  
Norwich, UK

[j.noppen@uea.ac.uk](mailto:j.noppen@uea.ac.uk)





## A Short Introduction

### Research at the School of Computing Sciences at UEA

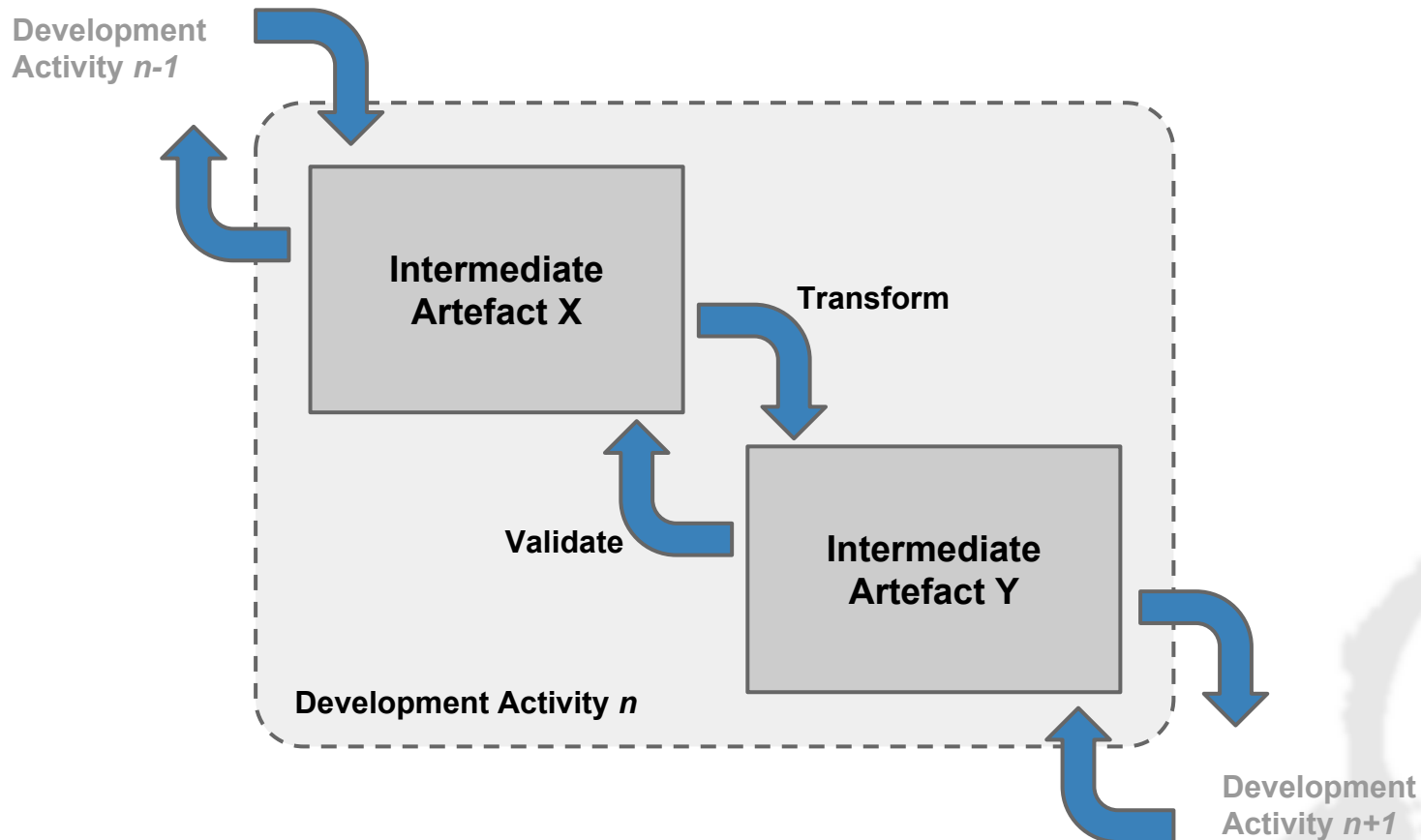
Research is divided into three laboratories:

- 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

## Software Engineering Research at UEA



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.



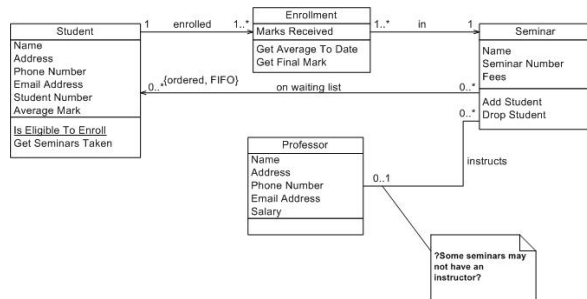
# Forensic Analysis of Legacy Software Systems

```
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

## 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

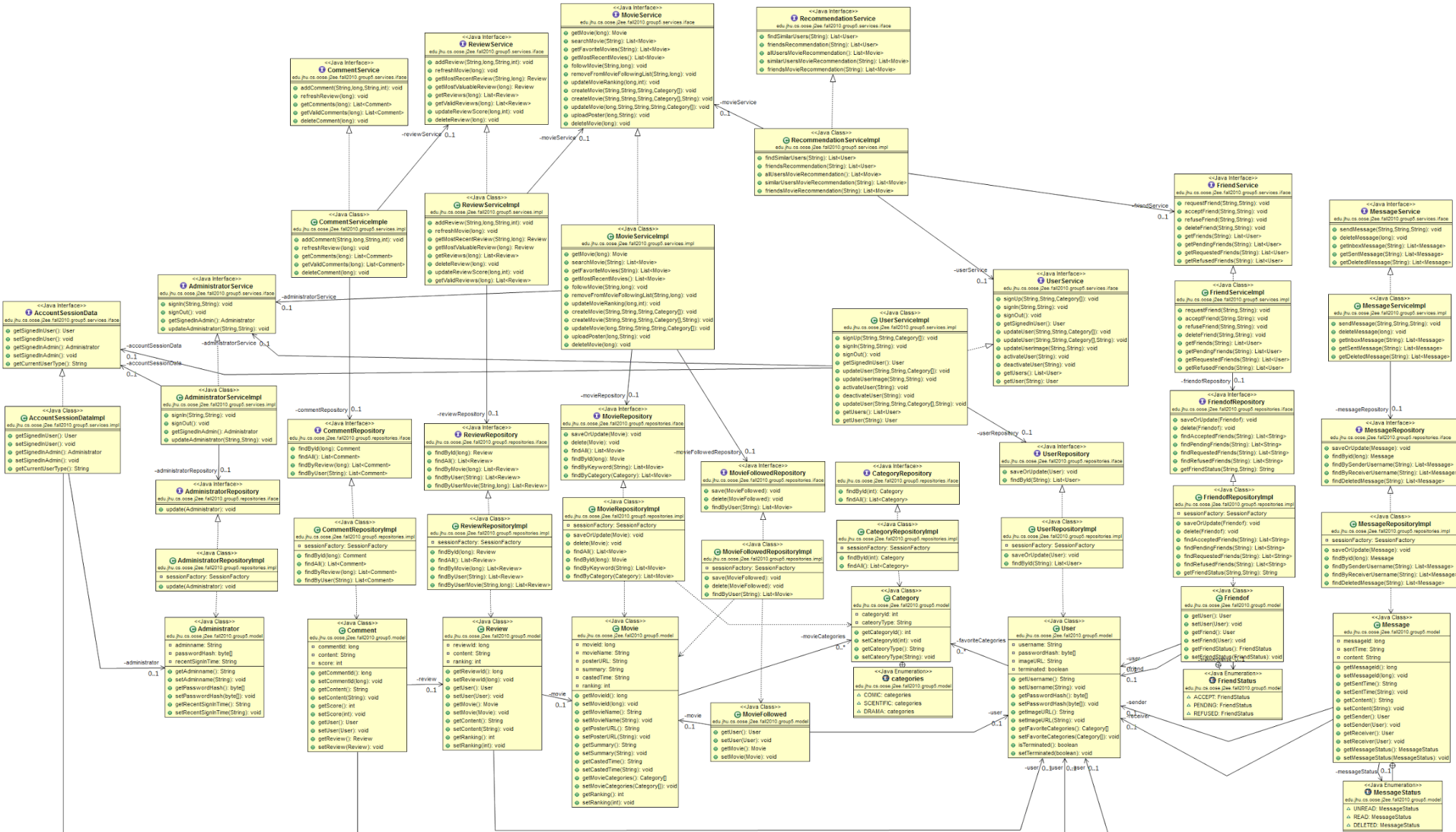
Traditional reverse engineering tries to reconstruct a class diagram from source code. At best this leaves the developer with a massive class diagram of potentially thousands of classes.



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?

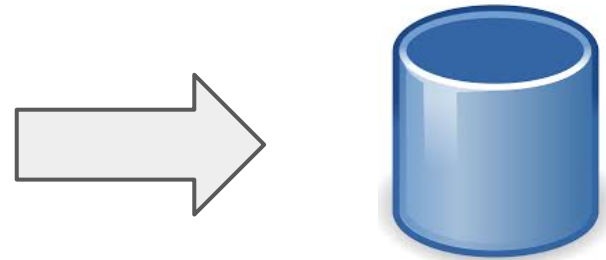
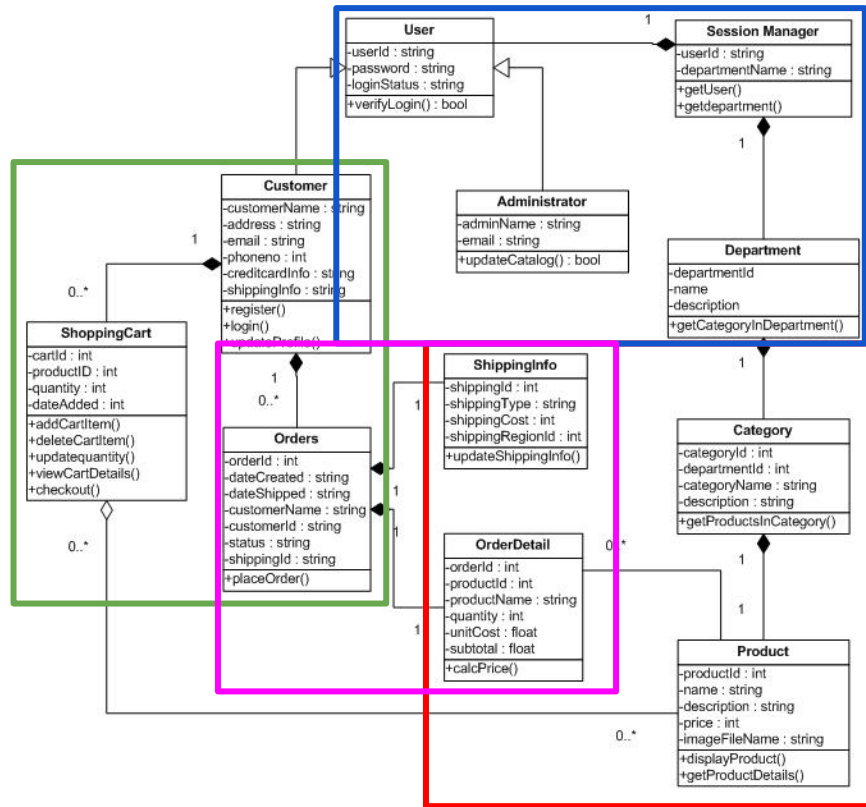
# Forensic Analysis of Legacy Software Systems







# Forensic Analysis of Legacy Software Systems



## 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.

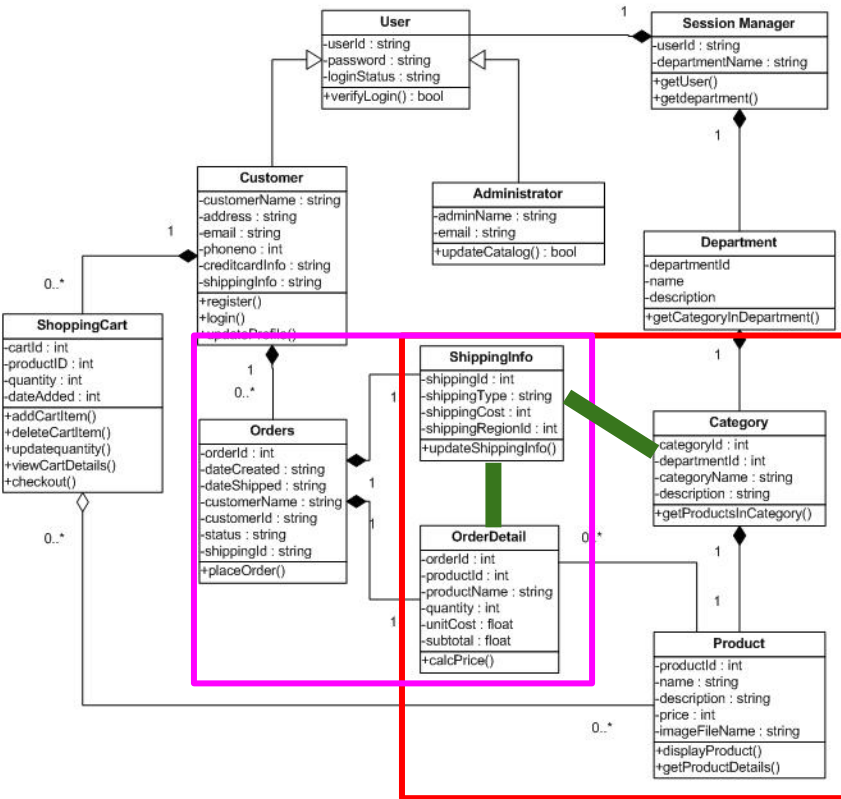
We model this by attaching every relation between classes with a number on the domain [0..1].

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:

$$\#Co-commits\ of\ C1\ and\ C2$$

$$Max(\#Commits\ of\ C1,\ \#Commits\ of\ C2)$$



# 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.

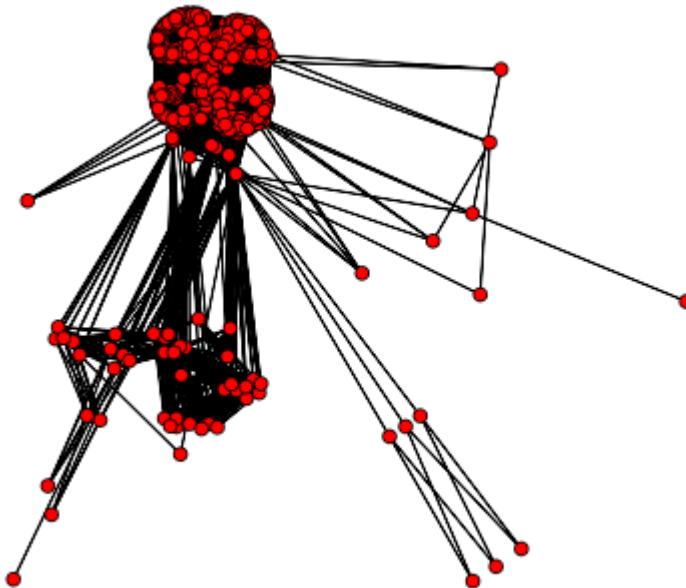
This matrix combines the ground truth of the structural diagram with the detected semantic relations.

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.



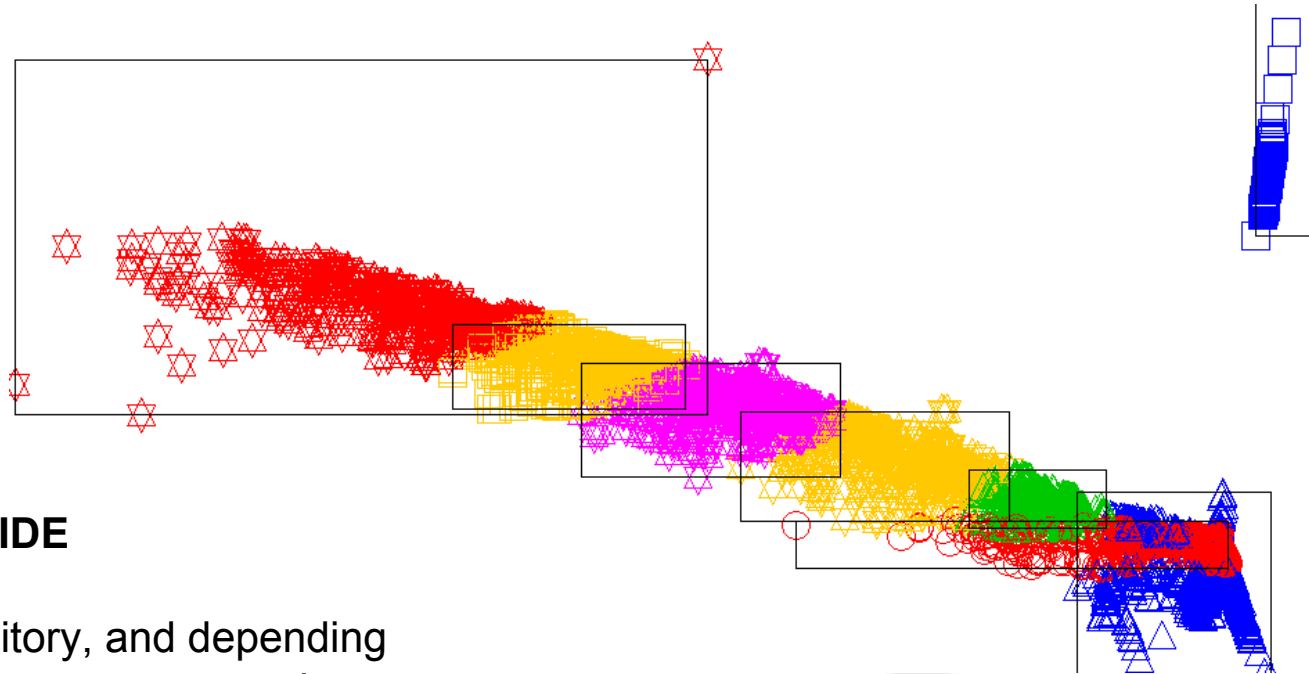
Interesting aspects and semantic concepts can already be observed:

- 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.



# Forensic Analysis of Legacy Software Systems

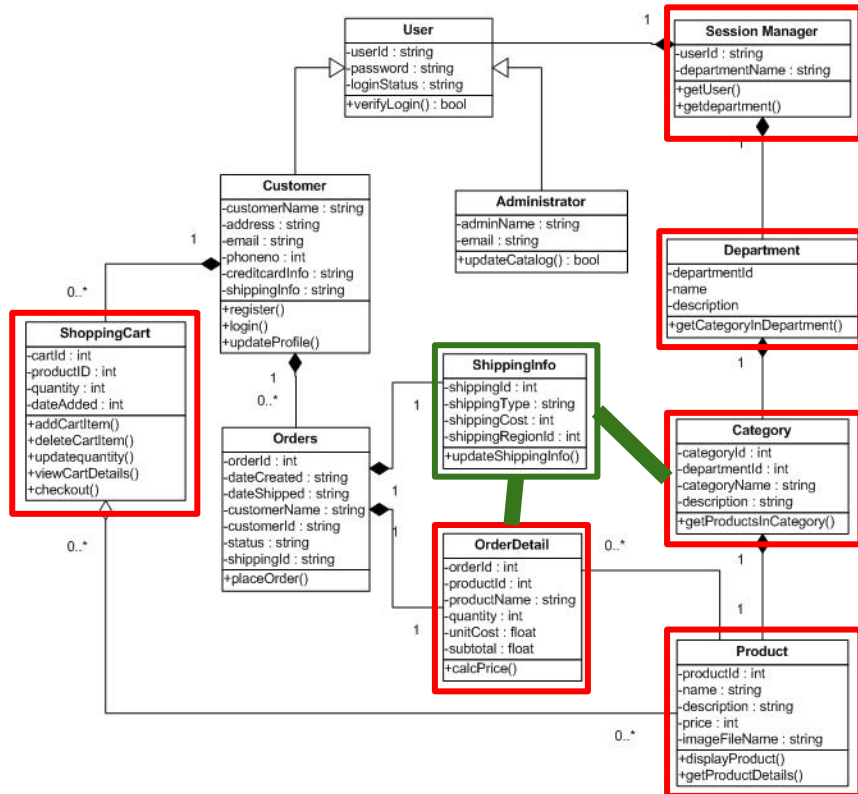


## Clustering the Eclipse IDE

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

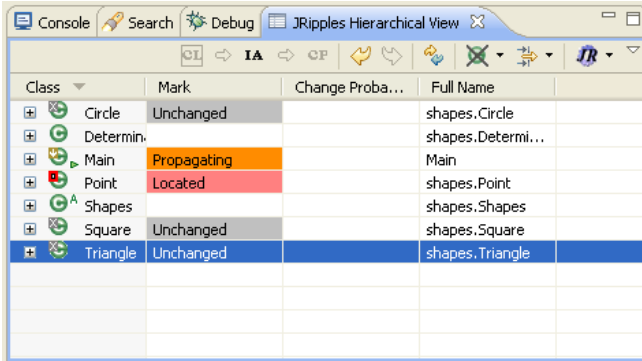
Used to determine how changing one class affects the rest of the system.

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



Class	Mark	Change Proba...	Full Name
Circle	Unchanged		shapes.Circle
Determin...			shapes.Determi...
Main	Propagating		Main
Point	Located		shapes.Point
Shapes			shapes.Shapes
Square	Unchanged		shapes.Square
Triangle	Unchanged		shapes.Triangle

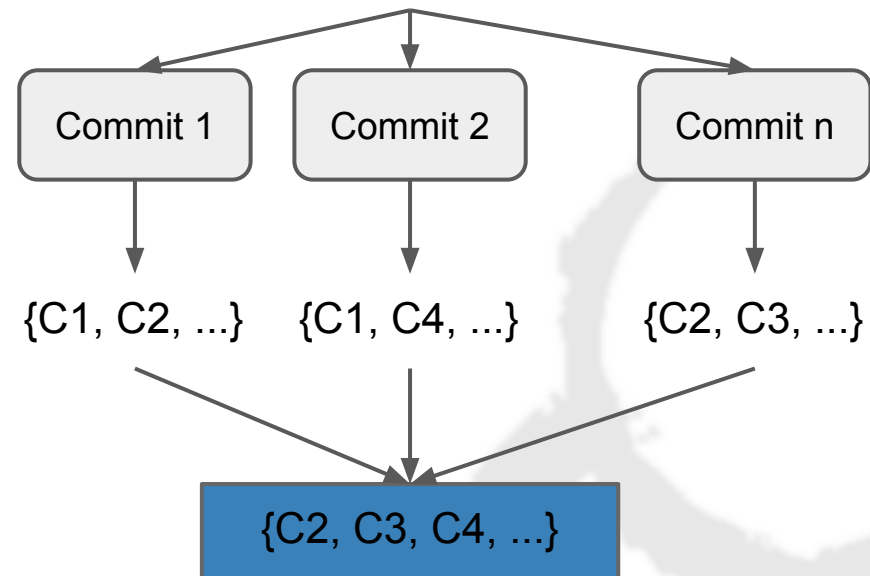
## Change Impact Analysis Evaluation

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.



<i>Article</i>	
-	name : String
-	contents : String
+	PAGENAME_SUFFIX : String
+	getName() : String
+	setName(newName : String) : void
+	getContents() : String
+	setContents(newContents : String) : void



# Forensic Analysis of Legacy Software Systems

Article	
- name : String	
- contents : String	
+ PAGENAME_SUFFIX : String	
<hr/>	
+ getName() : String	
+ setName(newName : String) : void	
+ getContents() : String	
+ setContents(newContents : String) : void	

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

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

## 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

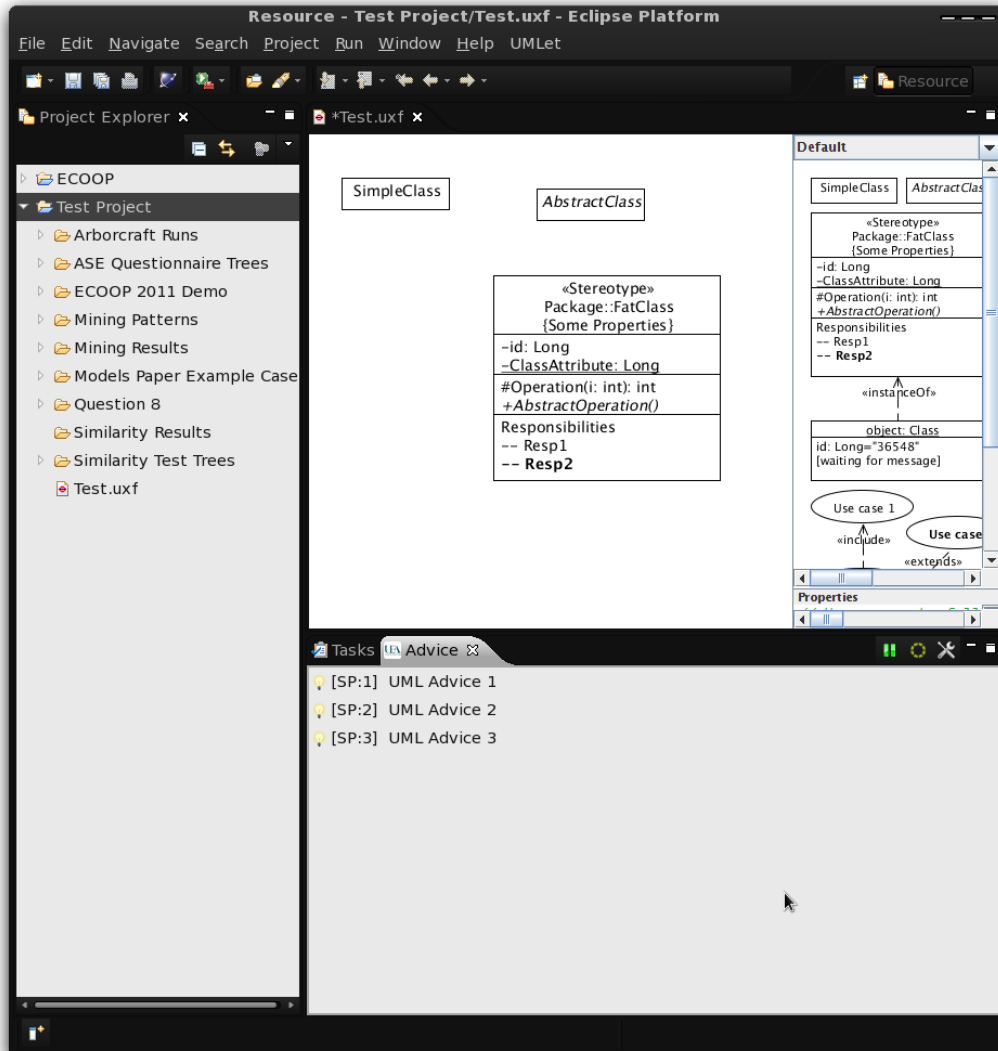
**Precision:**

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

**Recall:**

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

# 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

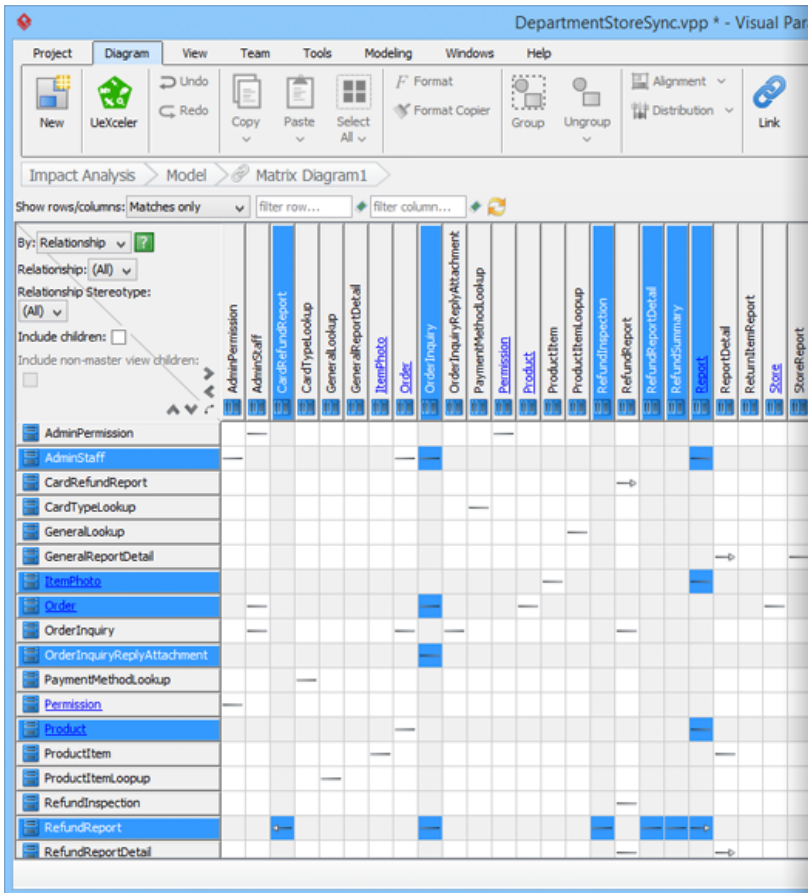
## Change Impact Analysis

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

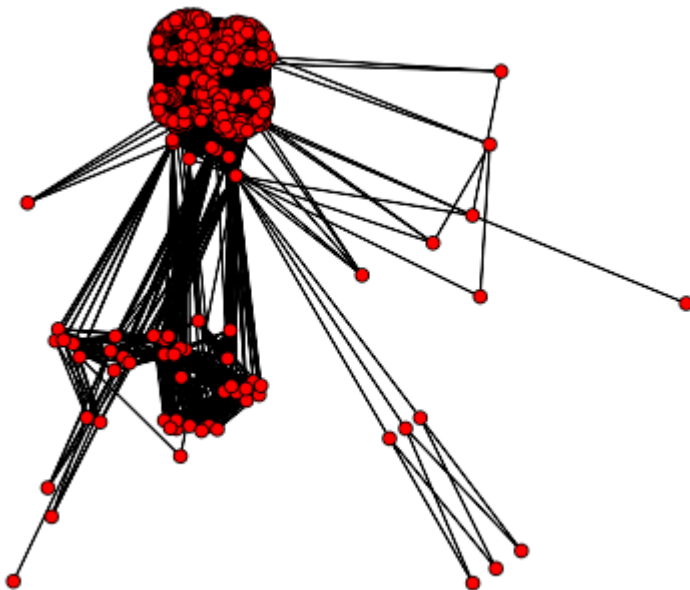
### Semantic Concept Identification

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

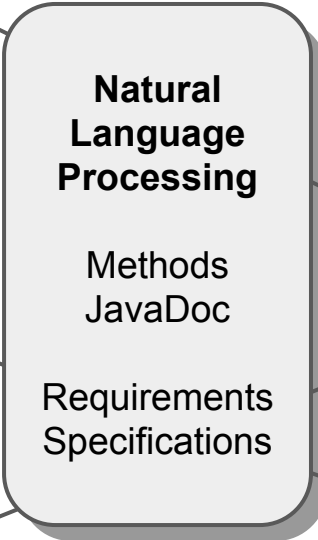


# Future Work

```

1 /**
2  * My first console program for CS 170.
3  *
4  * @author (your name goes here) ← your name
5  * @version (place the date here) ← the date
6  */
7 public class MyFirstConsoleProgram
8 {
9     /**
10    * These are the commands your program will carry out.
11    */
12    public static void main(String[] args)
13    {
14        System.out.println("Hello, I love you. ");
15        System.out.println("Won't you tell me your name?");
16        System.out.println("My name is Stephen Gilbert, ");
17        System.out.println("by the way."); ← your name
18    }
19 }
20
21

```



[Company Name] [Version Number]

### 3 Requirements Statement

Discuss the functional and informational requirements based on the opportunities and deficiencies that were identified in the Concept/Proposal document.

#### 3.1 Existing Methods and Procedures

Describe the current methods and procedures used to satisfy existing requirements. Provide the reader with a summary of any analysis that has been performed on the existing system's ability to satisfy the objectives, goals, and critical success factors as identified in the previous chapter. Describe existing products and services delivered to customers.

Product/Service	Used by Customers to
Identify the product/service	Describe the product's purpose, key functions, and customer base
Identify the product/service	Describe the product's purpose, key functions, and customer base
Identify the product/service	Describe the product's purpose, key functions, and customer base

#### 3.2 Required Capabilities

##### 3.2.1 User Requirements

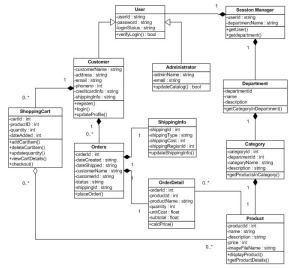
Describe user requirements in functional terms. These can be in narrative form (i.e. text) and/or written from the user's perspective (i.e. what the user requires from the system). Use flowcharts and/or use cases if it helps illustrate the requirements and their interrelationships.

For requirements that may improve existing methods and procedures, state the extent of anticipated improvements and its relationship to opportunities and deficiencies as captured in the Concept/Proposal. Ensure that all functions are identified and described in sufficient detail for the System Designers to make accurate estimates of the resources and efforts required. Failure to do so will result in inaccurate estimates and possibly require the project to be re-scoped, resulting in a change and cost overrun.

Req#	Ranking	Requirement	Improvement
1	(HML)	Describe the first user requirement in terms of the overall functionality.	Describe the extent of anticipated improvements and its relationship to opportunities and deficiencies.
2	(HML)	Describe the second user requirement in terms of the overall functionality.	Describe the extent of anticipated improvements and its relationship to opportunities and deficiencies.

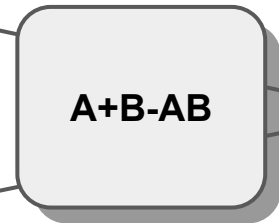
# Future Work

## 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	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