An Investigation into Patterns and Anti-Patterns in Search based Refactoring

faculty of environmental technology department of computer science and creative technologies University of The West Of England, bristol

|  |
| --- |
| Kieran Sears |
| bsc Computer Science |
| 7/1/2016 |

# Abstract

*Search Based Software Engineering (SBSE) explores the testing, design, requirements,*

*project management and refactoring of software using AI “search based optimization” techniques. This field of research focuses mostly on source code and software testing, with little on forward processing of conceptual diagrams and business models, which would assist developers in the preliminary stages of the product lifecycle, where early planning can prevent future inhibitions in terms of maintainability, extensibility and flexibility of code.*

*This paper utilizes a Genetic Algorithm (GA) to convert UML models stored in an XMI format to well-formed models that reduce coupling and give high cohesion, by employing Object Orientated decomposition to detect anti-patterns and refactor the model to a better formed architecture based on well-known design patterns. The primary aim of this is to outline and justify whether patterns and anti-patterns make for useful heuristics in reducing the search space and ultimately the computational effort required to run the GA.*

**Key Terms:**

- Genetic algorithm  
- software architecture design  
- software refactoring

Contents

[Abstract 1](#_Toc444370551)

[Introduction 4](#_Toc444370552)

[ethical Considerations 5](#_Toc444370553)

[Research Question 5](#_Toc444370554)

[Literature Search 6](#_Toc444370555)

[Literature Review 7](#_Toc444370556)

[Search Based Software Engineering 7](#_Toc444370557)

[UML 9](#_Toc444370558)

[XMI 10](#_Toc444370559)

[Genetic Algorithm 10](#_Toc444370560)

[Fitness Metrics 11](#_Toc444370561)

[Design Pattern 13](#_Toc444370562)

[Anti-Pattern 13](#_Toc444370563)

[What is the current state-of-the-art (technology and algorithms)? 16](#_Toc444370564)

[Related Work 16](#_Toc444370565)

[How have other people tried to solve it (products)? 16](#_Toc444370566)

[Project Scope 18](#_Toc444370567)

[What am I claiming about this project? (Claim/Hypothesis/Assertion) 18](#_Toc444370568)

[What are my project goals and objectives? 19](#_Toc444370569)

[What will be my success criteria? 19](#_Toc444370570)

[Requirements Engineering 19](#_Toc444370571)

[Must have 19](#_Toc444370572)

[Should have 20](#_Toc444370573)

[Could have 20](#_Toc444370574)

[Won't have 20](#_Toc444370575)

[Requirements selection criteria 20](#_Toc444370576)

[Types of problems and their suggested pattern fixes: 21](#_Toc444370577)

[Testing 22](#_Toc444370578)

[methodology 23](#_Toc444370579)

[Design 23](#_Toc444370580)

[What alternative designs have you considered? 23](#_Toc444370581)

[Why have you chosen the design that you have? 24](#_Toc444370582)

[How have you identified the different components of your design? 24](#_Toc444370583)

[fitness 24](#_Toc444370584)

[How are you going to describe your design? 24](#_Toc444370585)

[What issues arose during the design process? 24](#_Toc444370586)

[What test data did you specify? 24](#_Toc444370587)

[Implementation 24](#_Toc444370588)

[How will you implement your project and why? 24](#_Toc444370589)

[How will you record how you implemented it? 24](#_Toc444370590)

[What issues arose during implementation? 24](#_Toc444370591)

[How will you demonstrate that it works? 25](#_Toc444370592)

[How will you show what has been achieved? 25](#_Toc444370593)

[Has anything not been finished? 25](#_Toc444370594)

[Testing 25](#_Toc444370595)

[How will you test your code? 25](#_Toc444370596)

[How do your tests prove that your code works correctly? 25](#_Toc444370597)

[How will you document your test results? 25](#_Toc444370598)

[Conclusion 25](#_Toc444370599)

[Do your findings support or refute your claim? 25](#_Toc444370600)

[Have you met your original project goals? 25](#_Toc444370601)

[What limitations does your product have? 25](#_Toc444370602)

[What thoughts do you have now about your product? 25](#_Toc444370603)

[Future Work 25](#_Toc444370604)

[How could you improve your product? 25](#_Toc444370605)

[How could you improve the development process? 25](#_Toc444370606)

[What would you do differently? 25](#_Toc444370607)

# Introduction

Programming within the Object Orientated paradigm has the aim to modularize code into reusable and easily extensible classes which aid the readability, maintainability and repurposing of code. As the complexity of software systems grows throughout the product lifecycle the maintenance of software becomes exponentially more challenging as the amount of work dedicated to refactoring, documenting, extending and testing rapidly increases. Increasingly the use of AI techniques will be needed to accommodate developments in the field of computing, both in terms of additional complex system integration and being able to keep pace with time schedules of product release to remain on the cutting edge of technology.

This work outlines the approaches to refactoring the logical view of software architecture via automated transformations on UML Class Diagrams, and then goes onto investigate how specifically Design patterns and anti-patterns can be used as heuristics to accelerate this automated process to produce near optimal results.

Evolutionary computing methods, such as Genetic Algorithms, Simulated Annealing, Ant Colony Optimization, etc. have already demonstrated that manipulation of software architectures by treating the problem as a search task can already be automated to free up the resources spent maintaining code.

Being able to use patterns and anti-patterns in this context of automated design will be a key component to justify maintenance changes to the user which the AI has made, as well as to reduce the search space in which candidate solutions are sought, as each additional software component increases this space exponentially also increasing the risk of producing false positive candidate solutions which should not be used due to inherent faults in their design.

## ethical Considerations

The nature of copyright and who maintains responsibility for the code generated remains ambiguous. Although code may qualify as literary work, due to it not being created directly by an author it cannot legally be patented. Despite this, responsibility for the improper functioning of programs still requires accountability, for this reason the use of search based software engineering for safety critical systems should be handled with extreme caution and still be rigorously tested before deployment.

Evolved solutions to software engineering make tractability complex. Being able to trace the origins of code, for solving bugs, creating fixes, justifying investment and allowing the training of employees, etc. requires documentation of some kind which outlines the changes made during automated maintenance. If deployment of these technologies in industry are made, this will be a complicated problem to automate in a high level human readable language, nevertheless an essential one.

## Research Question

In the following chapters we will be addressing how to identify anti-patterns in Class Diagrams to guide the mutation operator of a Genetic Algorithm (GA) to apply a corresponding Pattern “fix” to its architecture. This experimental prototype is to test the hypothesis that anti-patterns accelerate beneficial transformations on Class Diagrams operated on by a Genetic Algorithm to reduce coupling and increase cohesion among classes.

A critical statistical comparison to evolution which does not use anti-pattern to pattern heuristics will be used as a control measure as part of this investigation.

## Literature Search

Papers stemmed from critically peer reviewed sources, such as academic conferences or reputable publishers within the field of computer science such as IEEE Explore, Science Direct and ACM Digital Library.

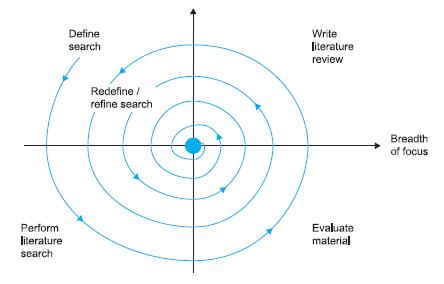
Key search terms were periodically refined when searching these databases beginning with the broad scope of “Search Based Software Engineering”, and narrowing as more related terms were found in shortlists of more related papers.

Shortlists were composed through manual assessment of abstracts and subsequently, if in parallel with the investigation, their conclusions. If these were both highly related their content was then studied to have information extracted including key terms and definitions, useful formulae and quotable passages. References were then reviewed for number of citations each author’s work had based on Google Scholar, and those who were highly credited were added to a personal list of professionals in the field.

Assessment criteria questions to address how related papers were included:

1. Does the article center on Software Modelling, especially in regard to UML?
2. Is it concerned with AI Search techniques?
3. Does it address refactoring or the restructuring of software architecture?
4. Does it contain references to Design Patterns and Anti-Patterns?

The graph in the “Literature searching and literature reviews” Chapter of [Projects in Computing and Information Systems; A Student’s Guide Christian W. Dawson] provides a succinct outline of the process involved as displayed in figure (TODO add figure reference).



# Literature Review

The dependent and independent factors of the investigation were determined through the collection and study of key background concepts found during the literature search. These are in turn disseminated and summarized whist being reviewed and critically evaluated based on: how they relate to the investigation, flaws in design, evidence to support claims and

### Search Based Software Engineering

The aim of Search Based Software Engineering (SBSE) is to move software engineering problems from human-based search to machine-based search, using a variety of techniques from the metaheuristic search, operations research and evolutionary computation paradigms (Harman, Mansouri and Zhang, 2012; 2013). Harman conducted a recent survey on work in this area which predominantly focused on

Evolutionary Computing is a field comprised of multiple search techniques each with their own advantages and disadvantages when applied to a given problem. Although these techniques can be broken into three distinct categories: Optimization, Modelling and Simulation. Respectively these sub fields can be seen as being focused on either the input, the model, or the outputs of a black box system. Because of the nature of our task being the optimization of design models as inputs, we will be looking at optimization techniques in particular. (TODO add citation: Introduction to evolutionary computation J.E. Smith)

Several authors have addressed the design problem of component selection and integration. This component selection problem is closely related to the requirement assignment problem.

Baker et al. [2006] presented results on greedy optimization and SA for component selection.

Yang et al. [2006] proposed an approach for the software integration problem by using GAs to reduce risk.

Classical OR techniques have also been applied to component selection problems: Desnos et al. [2008] combined backtracking and branch-and-bound techniques for automatic component substitution problem to optimize software reuse and evolution.

Alba et al. [Alba and Chicano 2007a,b,c; Alba et al. 2008; Chicano and Alba 2008b,b,c] also showed how Ant Colony Optimization (ACO) can be used to explore the state space used in model checking to seek counter examples.

Mahanti and Banerjee [2006] also proposed an approach for model checking, using ACO and Particle Swarm Optimization (PSO) techniques.

Other authors have also explored the relationship between SBSE and model checking. For instance, Johnson [2007] used model checking to measure fitness in the evolution of finite state machines.

Katz and Peled [2008a,b] provided a model checking based GP approach for verification and synthesis from specification. They present an approach that combines Hoare–logic–style assertion based specifications and model checking within a GP framework [He et al. 2008]. (Harman et al., 2014)

#### Model Driven Architecture/Development

Brosch et al [P. Brosch, P. Langer, M. Seidl, K. Wieland, M. Wimmer, G. Kappel, W. Retschitzegger and W. Schwinger, "An example is worth a thousand words: Composite operation modeling by-example," Model Driven Engineering Languages and Systems, pp. 271-285, 2009.] records refactoring operations acted on the model by users to automate model transformations. Pro's are developers dont require knowledge of the underlying modeling language or model transformation languages

“models may be viewed and treated as graphs, the algebraic graph transformation theory [Roz97, EEPT06] may be employed to describe model transformations in a formal, declarative, and rule-based fashion”

Brosch et al use Genetic Programming (GP) as a method for disseminating and performing transformations on UML architecture in

### UML

The Unified Modelling Language (UML) is the successor to the wave of object-oriented analysis and design (OOA&D) methods that appeared in the late '80s and early '90s. [Fowler, M., 2004. UML distilled: a brief guide to the standard object modeling language. Addison-Wesley Professional.] It has seen much success and is now considered an industry standard at capturing the structure and processes at both the micro and macro levels of enterprise systems.

A survey in the support of evolution of UML models conducted by (Amal Khalil and Juergen Dingel, 2013) [ Supporting the Evolution of UML Models in Model Driven Software Development: A Survey Amal Khalil and Juergen Dingel ]

### XMI

Extensible Markup Language (XML) Metadata Interchange (XMI) has numerous advantages when capturing models developed in UML. Extracts from [Grose, T.J., Doney, G.C. and Brodsky, S.A., 2002. Mastering Xmi: Java Programming with Xmi, XML and UML (Vol. 21). John Wiley & Sons.] have been shortlisted below:

* XMI leverages XML technologies.
* XMI enables you to use modeling with XML.
* Software that supports XMI creates schemas from models.
* Software that supports XMI provides a higher level of abstraction than XML elements and attributes.
* XMI helps you produce XML documents that can be easily exchanged.
* XMI enables you to create simple documents and make more advanced ones as your application evolves.
* XMI enables you to tailor the XML representation of your objects and document your choices in your models.
* XMI enables you to work with data and meta-data.

#### Class responsibility assignment

Bowman et al. [2008] applied the Strength Pareto Evolutionary Algorithm 2 (SPEA2) multiobjective optimization algorithm to provide decision support system for the Class Responsibility Assignment (CRA) problem.

Bowman et al. [Bowman, M., Briand, L. C., and Labiche, Y. Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms. IEEE Transaction on Software Engineering, 36(6):817–837, 2010] study the use of a multi-objective genetic algorithm (MOGA) in solving the class responsibility assignment problem

Although this project is not concerned with class responsibility in terms of GRASP patterns it is worth noting the distinction between class responsibility assignment (CRA) and model refactoring based on design patterns.

Decomposition of objects is a difficult task as factors which often conflict with one another such as encapsulation, granularity, Dependency, flexibility, performance, evolution, reusability can often blurred the lines between what is inappropriate architecture for a particular application. Finding classes which have no real world counterparts but improve composition has another dimension of complexity to the refactoring process. This is assisted through the use of design patterns which suggest less obvious abstractions to candidate solutions. Granularity, that is the size of classes, can also be tailored around design patterns by delegating responsibility on appropriate scales to classes.

Object composition is favored over inheritance – small inheritance trees, good encapsulation (attributes of parent class are hidden), can be less human readable and suffer performance issues, but ultimately can change behaviour of objects at runtime rather than compile time giving greater flexibility (Vlissides et al., 1995).

#### Fowler's refactorings

Fowler M., Refactoring - Improving the Design of Existing Code, Addison Wesley, 1999.

### Genetic Algorithm

Amoui et al. [Amoui, M., Mirarab, S., Ansari, S., and Lucas, C. A genetic algorithm approach to design evolution using design pattern transformation. International Journal of Information Technology and Intelligent Computing, 1:235– 245, 2007] use the GA approach to improve the reusability of software by applying architecture design patterns to a UML model. Their goal is to find the best sequence of transformations, i.e., pattern implementations. Best chromosomes are evolved so that abstract packages become more abstract and concrete packages, in turn, become more concrete.

[Simons, C. L., Parmee, I. C., and Gwynllyw, R. Interactive, evolutionary search in upstream object-oriented class design. IEEE Transactions on Software Engineering, 36(6):798816, 2010] Simons et al make use of a Genetic Algorithm for interactive AI which serves to assist developers in the upstream development of system architecture through transformations on Class Diagrams. Candidate solutions are the encoded attributes and methods derived from use cases allocated consecutively through surjections and partitions respectively to classes denoted by end of class “markers” to discretely represented genomes. Genetic Operators isolating discrete zones from the search space,

R¨aih¨a et al. [R¨aih¨a, O., Koskimies, K., and M¨akinen, E. Genetic synthesis of software architecture. In Proceedings of the 7th International Conference on Simulated Evolution and Learning (SEAL08), pages 565–574. Springer, 2008.] have taken the design of software architecture a step further than Simons and Parmee [Simons, C. L. and Parmee, I. C. A cross-disciplinary technology transfer for search-based evolutionary computing: from engineering design to software engineering design. Engineering Optimization, 39(5):631–648, 2007 ///////////////// Simons, C. L. and Parmee, I. C. Single and multi-objective genetic operators in object-oriented conceptual software design. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007), pages 1957–1958. ACM, 2007] by starting the design from a responsibility dependency graph. The dependency graph can also be achieved from use cases, but the architecture is developed further than the class distribution of actions and data.

R¨aih¨a et al. [R¨aih¨a, O., Koskimies, K., M¨akinen, E., and Syst¨a, T. Pattern-based genetic model refinements in MDA. Nordic Journal of Computing, 14(4):322–339, 2008.] have also applied GAs in model transformations that can be understood as pattern-based refinements. In MDA, such transformations can be exploited for deriving a Platform Independent Model from a Computationally Independent Model. The approach uses design patterns as the basis of mutations and exploits various quality metrics for deriving a fitness function. They give a genetic representation of models and propose transformations for them. The results suggest that GAs provide a feasible vehicle for model transformations, leading to convergent and reasonably fast transformation process. R¨aih¨a et al. [R¨aih¨a, O., Koskimies, K., and M¨akinen, E. Scenario-based genetic synthesis of software architecture. In Proceedings of the 4th International Conference on Software Engineering Advances (ICSEA09), pages 437–445. IEEE, 2009.] have also later on added scenarios, which are common in real world architecture evaluations, to evaluate the fitness of their synthesized architectures

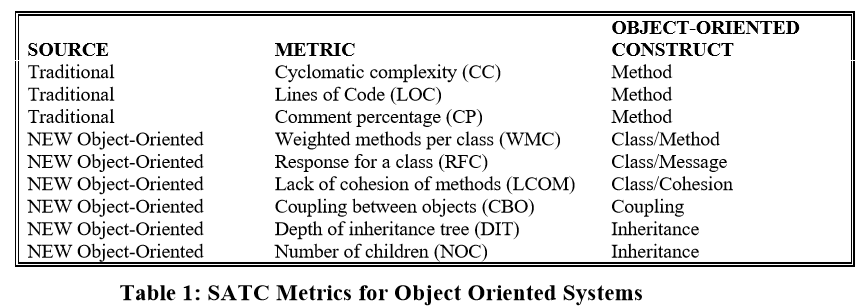
### Fitness Metrics

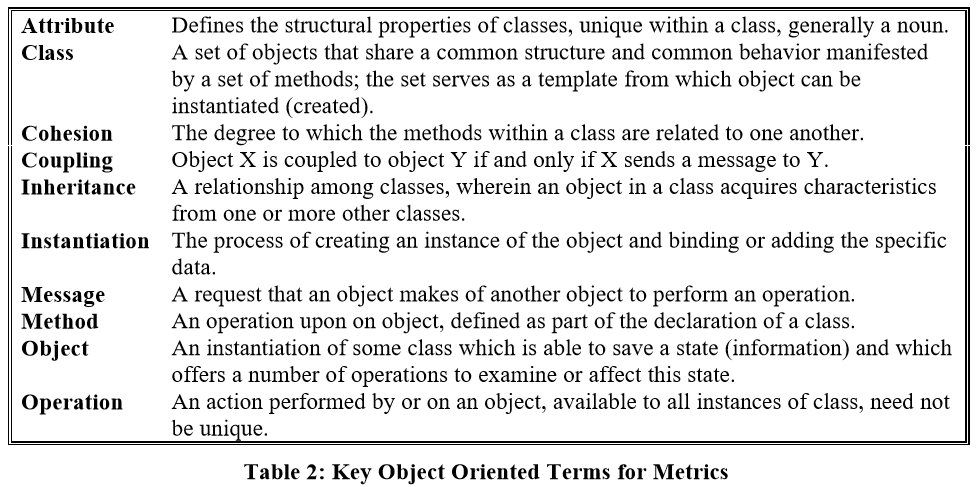
[O'Keeffe M. and 0 Cinneide M., "Towards design improvement through combinatorial optimization," Proc. Workshop on Directions in Software Engineering Environments, 2004.] and [Seng I., Stammel J. and Burkhard D., "Search-based determination of refactorings for improving the class structure of object-oriented systems," Proc. Conf. on Genetic and Evolutionary Computation, 2006.] both use Sum of Weighted Objectives, not multi-Objective measures of fitness. Although work has been done on multi-objective SAs there is less evidence of its application to GAs which [Multi-Objective Genetic Algorithm to Support Class Responsibility Assignment; Bowman, Briand, Labiche] demonstrates. Bowman’s work highly relates to our topic as he also aims on improving early OOAD models. But does not focus on problem driven processes in the way of identifying and converting anti-patterns to patterns, but instead centers on class responsibility assignment making it a more general approach to the problem, with this said the fitness metrics applied are still relevant to the problem domain and so will be discussed.

Bowman uses dependencies among methods and attributes to calculate the coupling and cohesion between classes. This information is drawn from Object Constraint Language as well as other UML Diagrams, i.e. sequence diagrams, in order to identify what attributes and association ends can be accessed by which methods. From this three measures of coupling are used from [Briand L. C., Daly J. and Wuest J., "A Unified Framework for Coupling Measurement in Object-Oriented Systems," IEEE TSE, 25 (1), pp. 91-121, 1999.], these being: Method-Attribute Coupling, Method-Method Coupling and Method-Generalization Coupling. And Two measures of cohesion taken from [Briand L. C., Daly J. and Wuest J., "A Unified Framework for Cohesion Measurement in Object-Oriented Systems," Empirical Software Engineering, 3 (1), pp. 65-117, 1998.], which are: Tight Class Cohesion and Ratio if Cohesive Interactions. TODO evaluate these metrics

The metrics evaluate the object oriented concepts: methods, classes, coupling, and inheritance.

The metrics focus on internal object structure that reflects the complexity of each individual entity and on external complexity that measures the interactions among entities. The metrics measure computational complexity that affects the efficiency of an algorithm and the use of machine resources, as well as psychological complexity factors that affect the ability of a programmer to create, comprehend, modify, and maintain software.





[Title: Applying and Interpreting Object Oriented Metrics Presenter: Dr. Linda H. Rosenberg Track: Track 7 - Measures/Metrics Day: Wednesday Keywords: Metrics, Object-Oriented]

Simons et al use Cohesion of Methods (Harrison et al) and Coupling between Objects (Briand et al) as fitness metrics for their Genetic Algorithm. They note the limitation of Cohesion of Methods in its inability to distinguish the number of methods in a given class and compensate by using a multiplier, so that classes with limited methods do not converge on high or low scores due to their limited scope. For instance results normalize between ranges 0.0 and 1.0, if a class has only a single method which accesses an attribute within the same class, its score would equal 1.0 for cohesiveness.

Coupling between Objects is taken as the measure of dependencies between classes if a given method of a class has a dependency on an attribute from another class. Again it’s limitation of not providing the strength of the coupling between two given classes (in terms of the number of methods in class x using attributes in class x) is addressed by first determining the set of all methods the first source class has which are dependent on the second target class and then dividing this value by the set of all uses of methods dependent on attributes.

Both these metrics provide a fair basis for determining coupling and cohesion, but undermine other ways of calculating fitness such as those previously mentioned in this chapter. Although the paper is focused on interactive design with user feedback being the predominant driving force for evolution and not multi-objective global search, which is understandable as often metrics conflict and the discovery of novel useful designs which work are more beneficial to the designer than those taken from a search spaced narrowed by presumptions of optimal architectures made by aggregated metrics.

### Design Pattern

It would be natural to suppose that work on design patterns could and should form a foundation for a strand of work on SBSE for design (Harman et al., 2014). Vlissides, J., Helm, R., Johnson, R. and Gamma, E known commonly in the field as the “Gang Of Four” outline descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context of OO-design called Design Patterns ( [Gamma, E., Helm, R., Johnson, R., and Vlissides, J. Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995]). Each design pattern has a name, a problem it is often associated with improving, the solution on how to refactor to a better practice design and finally the consequences of employing such designs.

The consequences of the design patterns are important as metrics of measuring performance in OO-design often conflict, for instance,

This possibility has recently been explored in detail by Raih¨ a¨ et al. [Raih¨ a 2008a,b; R ¨ aih¨ a¨ et al. 2008], who proposed a GA-based approach to automatically synthesize software architectures consisting of several design patterns.

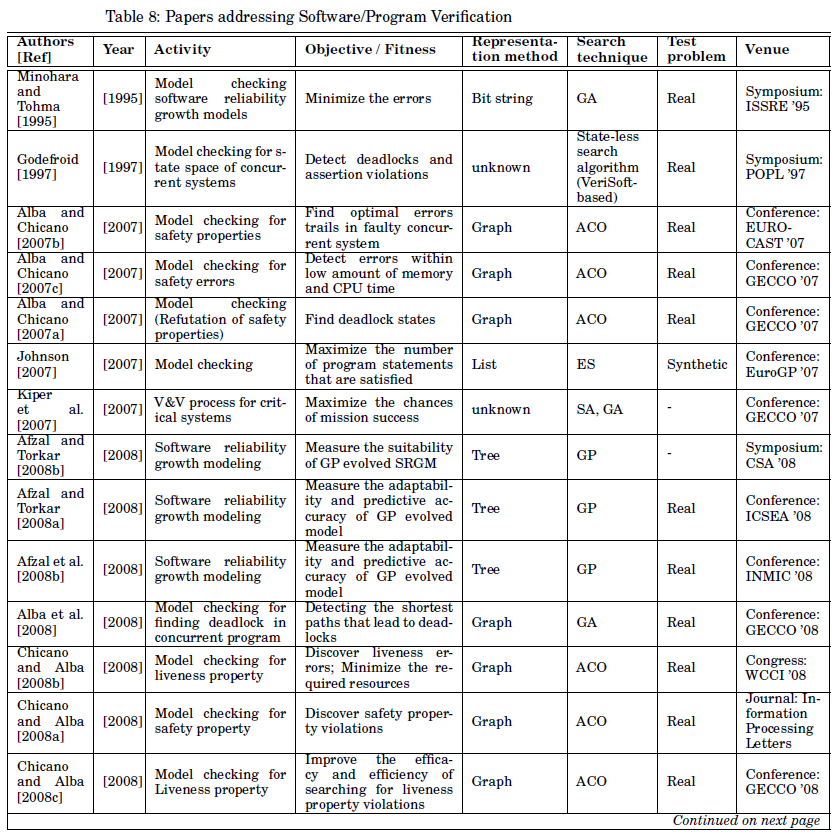
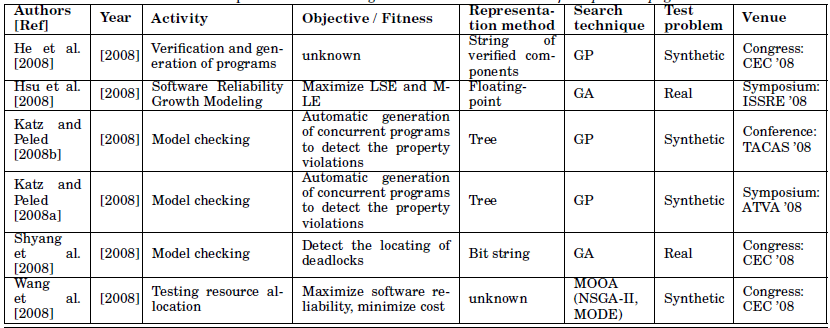
### Anti-Pattern

Jensen and Cheng [Jensen, A. C. and Cheng, B. H. C. On the use of genetic programming for automated refactoring and the introduction of design patterns. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010), pages 1341–1348. ACM, 2010.] present an approach based on genetic programming (GP) for generating refactoring strategies that introduce design patterns. They have implemented a tool, RE-MODEL, which takes as input a UML class diagram representing the system under design. The system is refactored by applying minitransformations. The encoding is made in tree form (suitable for GP), where each node is a transformation. A sequence of mini-transformations can produce a design pattern; a subset of the patterns specified by Gamma et al. [Gamma, E., Helm, R., Johnson, R., and Vlissides, J. Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.] is used to identify desirable mini-transformation sequences. Mutations are applied by simply changing one node (transformation), and crossover is applied as exchanging subtrees. The QMOOD [Bansiya, J. and Davis, C. G. A hierarchical model for object-oriented design quality assessment. IEEE Transactions on Software Engineering, 28(1):4–17, 2002.] metrics suite is used for fitness calculations. In addition to the QMOOD metrics, the authors also give a penalty based on the number of used mini-transformations and reward the existence of (any) design patterns. The output consists of a refactored software design as well as the set of steps to transform the original design into the refactored design. This way the refactoring can be done either automatically or manually; this decision is left for the software engineer.

This approach is close to those of R¨aih¨a et al. [R¨aih¨a, O., Koskimies, K., and M¨akinen, E. Genetic synthesis of software architecture. In Proceedings of the 7th International Conference on Simulated Evolution and Learning (SEAL08), pages 565–574. Springer, 2008.] and the approach used here, the difference being that Jensen and Cheng have clearly a refactoring point of view.

Note: — the simple existence of a pattern is not a reason for reward itself in the fitness function

Related work has been narrowed to explore work which was produced from 2008 onward for the sake of keeping research current and ensuring the prototype design of this project which draws inspiration from these works is contemporary. However, a table is included below of those works which precede 2008 taken from Search Based Software Engineering: Trends, Techniques and Applications; by Mark Harman, S. Afshin Mansouri, Yuanyuan Zhang. This is purely for historical purposes to demonstrate the origins of when work in this particular field began in earnest.



El-Sharqwi et al [M. El-Sharqwi, H. Mahdi and I. El-Madah, "Pattern-based model refactoring," in Computer Engineering and Systems (ICCES), 2010 International Conference on, 2010, pp. 301-306.] present an approach to apply model refactoring based on design patterns that are defined in XML notation. A design pattern consists of three parts: a Problem Specification describing the context where the design pattern can be applied to improve some quality aspect, a Target Specification describing the design pattern itself, and a Transformation Specification describing a sequence of primitive transformations required to apply the design pattern.

Cortellessa, Vittorio, Antinisca Di Marco, Romina Eramo, Alfonso Pierantonio, and Catia Trubiani. "Digging into UML models to remove performance antipatterns." In Proceedings of the 2010 ICSE Workshop on Quantitative Stochastic Models in the Verification and Design of Software Systems, pp. 9-16. ACM, 2010.

# Project Scope

## What am I claiming about this project? (Claim/Hypothesis/Assertion)

Non-functional requirements improvement through refactoring, being the redistribution of classes, variables and methods across the class hierarchy in order to facilitate future adaptations and extensions [Fowler M., 1999, Refactoring: Improving the Design of Existing Programs, Addison-Wesley.]

Endogenous transformations as the language and platforms used remain the same - source model to a target model and both models conform to the same metamodel, as opposed to exogenous - source model to a target model and both models conform to different metamodels. e.g., map Platform Independent Models (PIMs) to Platform Specific Models (PSMs) which is needed to handle tasks such as code generation, reverse engineering, and migration. (Khalil and Dingel, 2013)

The detection of anti-patterns prior to evolving refactoring solutions, and then during the evolution process will reduce the search space of the refactoring problem down to those solutions which actually improve upon the original design of the system in question, in terms of its performance when measured against metrics of good Object orientated design.

## What are my project goals and objectives?

Parse the meta-model input files into memory and represent them in such a way that the GA can measure and evolve them.

Implement the Anti-Pattern / Pattern analyzer for the meta-model representation which outputs messages to the mutation operator to guide its transformations.

Implement the Genetic Algorithm to process at least a population size of one, with a mutation operator which can optionally make use or not of messages from the pattern analyzer, and output its results to a human readable form.

## What will be my success criteria?

Success criteria will be

# Requirements Engineering

“Requirements engineering is the process of discovering, documenting and managing the requirements for a computer-based system. The goal of requirements engineering is to produce a set of system requirements which, as far as possible, is complete, consistent, relevant and reflects what the customer actually wants.” [Sommerville, I. and Sawyer, P., 1997. Requirements engineering: a good practice guide. John Wiley & Sons, Inc.]

### Must have

The minimum useable subset of requirements isolated from the literature review are that the prototype software must have: a parser to extract the information from the XMI test cases and represent it in memory, a detection system to identify anti-patterns and patterns already present in the models architecture in memory, an encoder to take the model in memory and represent it in a way which allows it to be manipulated by a GA, a GA to initialize a population of encoded representations and process them through a mutation operator, a measure of fitness of representations for the coupling and cohesion metrics as a basis for selection of candidate solutions, a system to decode the GA representations back into a human readable (preferably graphical) form.

These essential requirements produce an intuitive way to break down the project into smaller subsystems which could be worked on individually which aided in discerning where and if a component of the prototype had failed.

### Should have

Requirements which should be part of the project are: a way of parsing the best solutions back into XMI as an output, a Pareto-front selection system to allow the user to select the fittest subset of solutions and get user-feedback to drive the GA toward solutions which are more applicable to the problem domain and business logic that the meta-model input is attempting to address.

### Could have

Requirements which could be part of the project are a wider selection of fitness metrics other than coupling and cohesion to ensure selection processes are more refined.

### Won't have

The program will not change algorithms within the system, and will only focus on the architecture of the design. It will not be able to generate new functionality to the models. It will not be able to merge the functionality of two systems together. It is not designed to rigorously test the implementation of the software architecture as it deals only with conceptual models.

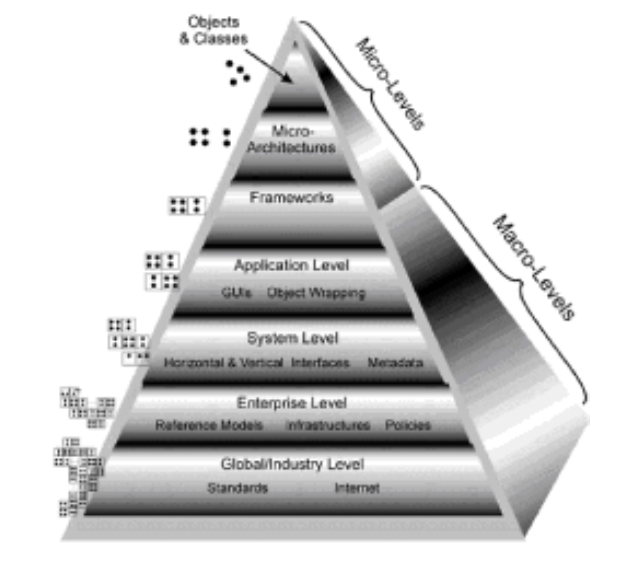
### Requirements selection criteria

The requirements were identified and prioritized by those aspects of the prototype which directly address the underlying hypothesis of the investigation, and those which would be more involved in the roadmap of improving automated software refactoring in general. They also were chosen based on the timeframe of the project, isolating those requirements which would take the longest time to implement and those which unfortunately fall outside of the deadline given.

Based on the literature review the following is supportive evidence of the decisions made for the requirements.

### Types of problems and their suggested pattern fixes:

Being able to implement the conditional statements needed to sufficiently capture the “rules of thumb” provided by (Vlissides et al., 1995), as well as create the database, for anti-patterns and corresponding patterns and be able to provide this information to the mutation operator so that it can apply the necessary transformations, is one of the most challenging aspects of the project. It requires a generous amount of testing for robustness and is not easily broken down into sub-tasks TODO reword : as each anti-pattern requires the feedback from subsequent generations of candidate solutions in order to perform its identification and application of anti-pattern to patterns to the mutation operator.



TODO Talk about scope in relation to “AntiPatterns

Refactoring Software, Architectures, and Projects in Crisis” William J. Brown, Raphael C. Malveau, Hays W. McCormick III, Thomas J. Mowbray

Capturing behaviour / semantics of original models “access”, “update”, and “call” preservation [T. Mens, N. Van Eetvelde, S. Demeyer and D. Janssens, "Formalizing refactorings with graph transformations," Journal of Software Maintenance and Evolution: Research and Practice, vol. 17, pp. 247-276, 2005.]. 1) accesses the same variables after the refactoring as it did before the refactoring, 2) updates the same variables after the refactoring as it did before the refactoring, 3) performs the same method calls after the refactoring as it did before the refactoring.

Constraint with Action Language Ajila and Alam [S. A. Ajila and S. Alam, "Using a formal language constructs for software model evolution," in Semantic Computing, 2009. ICSC'09. IEEE International Conference on, 2009, pp. 390-395.] for the automatic dependency analysis of the model. A formal specification language called TLA (Temporal Logic of Actions) is used to specify the actions and the operations in CAL. A model checker is used to verify and reason about these TLA specifications

Sunyé et al [G. Sunyé, D. Pollet, Y. Le Traon and J. M. Jézéquel, "Refactoring UML models," «UML» 2001—The Unified Modeling Language.Modeling Languages, Concepts, and Tools, pp. 134-148, 2001.]  set for class diagrams and statecharts, which can be defined as OCL constraints at the metamodel level, both the pre-condition and the post-condition expressed in the OCL constraints would ensure that the applied refactorings are behavior preserving. refactorings set included the addition, removal, move, generalization and specialization of modeling elements for class diagrams and operations such as folding incoming/outgoing actions, unfolding entry/exit action, grouping states, folding outgoing transitions, unfolding outgoing transition, moving state into composite state and moving state out of composite state for statecharts

Gorp et al [P. Van Gorp, H. Stenten, T. Mens and S. Demeyer, "Towards automating source-consistent UML refactorings," «UML» 2003-the Unified Modeling Language.Modeling Languages and Applications, pp. 144-158, 2003.]   using OCL, in terms of a precondition of the restrictions that need to be satisfied in the model before applying the refactoring step, a post-condition of the properties to be satisfied in the model by the refactoring, and the “code smells” or the problem that can be improved by the refactoring. (Fujaba tool to apply the two refactorings: Pull Up Method and Extract Method.)

 Van Der Straeten et al [R. Van Der Straeten, V. Jonckers and T. Mens, "A formal approach to model refactoring and model refinement," Software and Systems Modeling, vol. 6, pp. 139-162, 2007]. The authors in this work formalized the behavioral specification of a system represented by UML state machine and sequence diagrams in Description Logic and defined two properties, observation call preservation and invocation call preservation, to check the behavior preservation between a class and its refactored version along with their corresponding state machine and sequence diagrams. Tool support is implemented, as a plug-in for the Poseidon CASE tool, and is tested on small examples.

Astels [D. Astels and others, "Refactoring with UML," in Proc. 3rd Int’l Conf. eXtreme Programming and Flexible Processes in Software Engineering, 2002, pp. 67-70] provides examples on bad design smells in class and sequence diagrams and described a number of refactoring actions for such model smells. The author was motivated with the fact that bad smells detection can be easier in the model level more than in the code level.

Simulated Annealing, which has reasonable performance on many search problems. In this specific problem because of the independent instinct of neighborhoods and discrete search space, this method is not supposed to perform well. [Fowler M., 1999, Refactoring: Improving the Design of Existing Programs, Addison-Wesley.]

### Testing

#### Functional Requirements

Test cases for the functional requirements were written out using Microsoft Excel for succinctness and to easily visualize the success of the project throughout its lifecycle. They break down into the subsystem which is being addressed in the prototype, the task that subsystem is solving, any pre or post conditions needed for the task, the input into the subsystem task including test cases for outliers and faulty information, the expected output of the subsystem task, the actual output of the subsystem task, and the colour coordinated success of the subsystem task where green indicates a successful completion of test, yellow indicates that the test case passed but with unexpected output and therefore needs addressing in future, and red indicates a failed test case where output was unexpected and does not meet necessary requirements.

#### Non-Functional Requirements

Non-functional requirements such as the extendibility and maintainability of the prototype were conceived during the conceptual design phase of the project. Class types based on Usecases were formulated and had their responsibilities isolated to increase cohesion so that a good distribution of responsibility was delegated in such a way as to promote loose coupling. In particular the use of a Model-View-Controller architecture was decided upon to ensure a separation of concerns among those classes which were responsible for representing data into data structures, displaying data to the user, and manipulating data in a manageable way.

#### Test Input Data (XMI metamodels)

The test data used initially was that of a simple Book Shop test case, to ensure that the simplest of models could be refactored to meet minimum requirements, and in a way which was easy to analyse manually to ensure correctness.

## methodology

Agile methodology will be used when developing the system. Alternatives considered were iterative design in the form of rapid prototyping, waterfall and lean methodologies. Agile was chosen first and foremost as it is renowned for its success in industry. It also maintains the component based paradigm minimizing risk in terms of addressing bugs and changes as requirements analyses evolve, and can produce at least a version of the work within a given time constraint even if it is not the latest release.

Waterfall would have been a preferred alternative, as the time constraints for the project would have allowed a considerable amount of the planning work to be demonstrated to show understanding of the subject topic even if implementation were left unresolved and incomplete. This notion was repelled however as the nature of an investigation would require at least some experimentation in order to draw any kind of conclusions on the stated hypothesis. Therefore, the most appropriate method was to capture the minimum requirements through planning, building, testing and reviewing, before if having time, doing another iteration of a more refined and extended version.

# Design

## What alternative designs have you considered?

Fitness

 SDMetrics [J. Wüst. SDMetrics. http://sdmetrics.com/, last accessed: November 19, 2012, 2011.] metrics are defined in current OO design quality measurement tools that are used in analyzing the structure of UML models

Examples of these metrics are presented by Kim and Boldyreff [H. Kim and C. Boldyreff, "Developing software metrics applicable to UML models," in 6th ECOOP Workshop on Quantitative Approaches in ObjectOriented Software Engineering, 2002.] and by Enckevort [T. Enckevort, "Refactoring UML models: Using open architecture ware to measure UML model quality and perform pattern matching on UML models with OCL queries," in Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, 2009, pp. 635-646.].

## Why have you chosen the design that you have?

## How have you identified the different components of your design?

### fitness

## How are you going to describe your design?

## What issues arose during the design process?

## What test data did you specify?

The first test case was a class diagram of a bookshop which included

# Implementation

## How will you implement your project and why?

## How will you record how you implemented it?

Records of implementation are given by a GitHub repository which was connected to the Netbeans IDE which was used for programming. Commits were regularly made after each subsystem task was completed following the initial commit following completion of the XMI parser and the overall Class architecture.

## What issues arose during implementation?

How to use dependencies formed from method parameters and return values that map to other classes in the system. Especially when there are already aggregation or composition relationships between these classes. The decision was made to prioritize the aggregation or composition relationship over the dependencies as these relationships already demonstrate the reliance on the external class.

## How will you demonstrate that it works?

## How will you show what has been achieved?

## Has anything not been finished?

# Testing

## How will you test your code?

## How do your tests prove that your code works correctly?

## How will you document your test results?

# Conclusion

## Do your findings support or refute your claim?

## Have you met your original project goals?

## What limitations does your product have?

## What thoughts do you have now about your product?

# Future Work

## How could you improve your product?

## How could you improve the development process?

## What would you do differently?