TraPT: A Traceability Pattern Tool

CS04-13-00

ALISTAIR POTT
Department of Computer Science
University of Cape Town
Cape Town
South Africa

DANIEL BERMAN
Department of Computer Science
University of Cape Town
Cape Town
South Africa

SUPERVISOR:
JUSTIN KELLEHER
Department of Computer Science
University of Cape Town
Cape Town
South Africa

ABSTRACT

TraPT is a tool for the structured and collaborative creation and cataloguing of software patterns. The goal of the tool is to facilitate an increase in the creation and use of patterns in organisations.

The tool is comprised of two modules, a pattern creation tool and a pattern encyclopaedia tool. The pattern encyclopaedia aids in accessing and learning about patterns. The encyclopaedia includes detailed information about patterns and traceability. The pattern creation tool allows for the collaborative creation and review of patterns according to a defined creation workflow.

1. INTRODUCTION

Over the past decade software patterns have become well established both within computer science and in industry. The benefits of using patterns are well described and evidenced [1], [2].

Despite this increase in attention on patterns most organisations do not utilise patterns. This project hypothesises that this lack of utilisation is because there is no available tool for the structured creation and cataloguing of patterns.

The TraPT tool was conceived to fill this gap. The project is part of a larger research project which involved the creation of patterns. During that research it was noted that there are no tools for the structured creation and cataloguing of patterns.

As such TraPT is a tool for the creation and cataloguing of patterns in a collaborative environment. The aim of the project is to provide a tool which would increase the use and creation of patterns in an organisation.

The system is logically partitioned into two main modules. These modules are the pattern cataloguing tool and the pattern creation tool. The tools share a common pattern storage system through which they interact.

2. BACKGROUND

The work on the TraPT tool deals with patterns, traceability, and traceability patterns. As such background information is presented on each of these topics.

2.1 Patterns

A software pattern is a description of how to solve a problem which is sufficiently abstract that it can be reapplied in many different contexts. The pattern is recorded as text and often uses diagrams to get the point across. Another way of thinking of patterns is as explicit representations of experience in solving problems which recur.

This explicit representation of experience is valuable in many circumstances. It allows for better communication. It also means that a best solution need not be re-discovered.

2.1.1 Definition of Patterns

There are many definitions of patterns in literature [2], [3], [4], [5]. However, certain themes recur and it is these concepts which define the essence of software patterns:

- Patterns describe solutions to problems.
- The problems that patterns solve are recurring. Thus patterns are described in order to enable reuse of the best solutions.
- The descriptions of the solutions are abstract in nature. This is essential if the solutions are to be reused in a variety of diverse situations.
- A pattern describes the context (environment) of the problem and the effects of that context on the solution. This enables the applicability of the abstract solution to concrete situations within diverse contexts.
- The solutions described by patterns embody the experience or knowledge of the developers who came up with those solutions.

In summary, a pattern is a solution to a problem in a context. The solution is described in an abstract way so that it can be reused in a variety of contexts.

2.1.2 The Benefits of using Patterns

The use of patterns within software development has several benefits which are widely discussed and illustrated in literature. It is widely accepted that the use of patterns increases productivity, aids communication, and increases the quality of solutions.

The use of patterns increases productivity.

The fact that patterns can increase productivity is widely accepted [6], [7], [1], [8], [2], [9]. There are a number of reasons why pattern usage increases productivity.

Firstly, patterns are by definition reusable solutions and as such their use allows developers to avoid spending time rediscovering best solutions. It is the fact that patterns capture the why as well as the what of solutions as well that allows them to be reused in a variety of situations. [2] discusses a major study conducted at AT&T which concluded that ‘as much as half of software development effort can be attributed to discovery.’ Thus, it is concluded that if developers are able to apply patterns instead of discovering solutions their productivity would be increased.

The fact that patterns increase productivity is evidenced in several papers. For instance, [1] describes how the use of the 'Reactor' pattern vastly improved productivity during system redesigns at Ericsson. In that situation the entire system platform was being changed, and as such no reuse of code was possible. It is stated that "patterns were often the only way of leveraging previous development expertise" [1]. Further the use of patterns in these projects is attributed with having "reduced risk significantly and simplified (the) redevelopment effort."

In [9] an experiment into the effect of pattern usage on code reuse and productivity is discussed. In this experiment patterns were applied in the development of two separate systems. After the development various means were used to estimate the productivity gains attributed to the use of patterns. It is the conclusion of this experiment that patterns increase code reuse (and thus productivity) significantly.

Patterns aid communication.

There is a wide range of information regarding the benefits that pattern usage offer to communication. Patterns are a compact way to reference a set of decisions and designs [7] while suppressing the "details not relevant at a given level of abstraction" [1].

In other words patterns are creating a "shared language for communicating experience and insight" [3]. Each pattern explicitly represents developer's experience and knowledge. Because the patterns are named, individuals can use those names to easily refer to that experience.

The contributions of pattern usage to communication are well evidenced in [1], [10], and [8]. In general the contribution is in the form of enabling users to easily communicate best practices at a higher level of abstraction than was possible before.

The use of patterns also benefits training and maintenance efforts. [1] states that because patterns explicitly record what developers implicitly know, their use enables organisations to "impart this knowledge to less experienced developers."

[8] discusses experiments conducted which show that the use of patterns does aid in maintenance efforts. The fact that patterns enable easier communication of knowledge and experience improves both training and maintenance.

[10] presents a broad survey of the effects of pattern usage in six large corporations including Motorola, Siemens, Ericsson, and IBM. From their experience in these situations the authors conclude among other things that patterns are a good communications medium.

Finally, because of their high level of abstraction patterns enable discussion above programming language barriers [1]. This is often useful when developers from very different backgrounds are working together.

Patterns increase quality.

This is a somewhat less often discussed benefit of using patterns. It is as a result of the first two benefits of pattern usage, namely: higher productivity and better communication. However much of the literature on patterns does agree that their use increases the quality of solutions [11], [3], [1], [2], [8].

[8] relates experiments conducted into the effect of pattern usage on maintenance projects. They conclude not only that these tasks were completed faster with the use of patterns but that fewer errors were made. The use of patterns increased the quality of the work done.

Because patterns allow developers to reuse best known solutions easily, quality is invariably improved. Patterns are developed collaboratively and over time. The review process which most patterns undergo (see Section 2.3.6) ensures that the quality of their solutions is maintained.

As such it is possible to avoid common mistakes and to develop better solutions by applying patterns rather than by developing solutions from scratch.

2.1.3 Pattern Catalogues

Various pattern catalogues exist on the Internet, in academic papers and in published pattern books. They each describe patterns for a related set of problems. Each catalogue is usually independent of all other catalogues and has an independent template that is specially designed to suit the patterns in the catalogue.

The Hillside Repository [12] has a range of pattern catalogues. These catalogues are part of the Hillside's website dedicated to patterns. Each catalogue has been submitted to the site and is independent of all the others in the repository. There is a limited search facility, but most patterns are found by searching through a long list of links and then following the chosen link to the home page of the contributor of the pattern catalogue. There is a wide variety of patterns that can be found, ranging from testing patterns, to integration and analysis patterns.

The most famous catalogue of patterns is the Gang of Four's design patterns. Their book [13] contains 23 design patterns along with programming code to give examples of how to implement the code.

Martin Fowler's book on analysis patterns is similar to the Gang of Four's book but it details analysis patterns. Fowler does not use a template to present his patterns but rather prefers a free-flow layout.

There are many other pattern catalogues available. None of the catalogues follow any standard for representing patterns. This makes patterns hard to identify.

One of the topics presented at the April, ChiliPLoP 2004 conference was the possible establishment of a pattern repository [14]. The repository would be peer reviewed. Many questions on the functionality the repository should provide as well as how it should look were posed.

2.1.4 How Patterns are defined

Patterns are defined using *pattern forms*. These consist of a set of fields such as 'motivation' or 'structure'. A pattern is defined by specifying the values of the form fields for that particular pattern.

In his books on patterns Alexander offered a pattern form in which he specified patterns [15]. Some patterns however, are not well suited to Alexander's form [7]. This fact is true of any particular pattern form and leads to the conclusion that there is no 'one best' pattern template (form) which can be applied across all pattern categories [16].

As such, a large variety of pattern forms has developed over time [16], [2]. However, all forms are merely a list of fields (elements) the specification of which comprises the pattern. Therefore, the definition of patterns is largely in prose. Although the inclusion of illustrating diagrams is essential [17], [2], [19], [20], most of the information defining a pattern is provided as the text under the headings of a pattern form.

2.1.5 Pattern Visualisation

Christopher Alexander maintained that the sketch is the essence of the pattern [2]. This is not surprising considering the power of illustrations to encompass a lot of information in an easily understandable form. For instance [20] states:

"Cognitive science emphasizes the strength of visual formalisms for human learning and problem solving. In software engineering, a clear, visual presentation of a system's architecture can significantly reduce the effort of comprehension."

Almost all patterns available include sketches [2]. However, the nature of the sketches included and the emphasis placed on them varies greatly. There is ongoing debate as to the best methods for specifying patterns visually.

On the one side of the debate is [2] who states:

"This is why the sketch is called a 'sketch' and not a 'graphical specification.' Most readers interpret refined diagrams too literally. There is much to be said for hand-drawn diagrams that abhor right angles and straight lines. Such a rough solution encourages the designer to craft or engineer the solution to the situation at hand.'

This side of the debate emphasises that if patterns are to be as abstract as they should be they need to have informal sketching. Coplien [2] suggests that more specific graphical representations are by definition more concrete and that thus some of the reusability of the solutions is lost with the loss of abstraction. [17] states that the use of conventional UML diagrams leads to "over specification" and a consequent loss of the abstract nature of patterns.

Proponents of less formal illustrations agree with Alexander that the developer should "carry out the detailed steps" of implementation according to his understanding of the solution and the context. They believe that the more precise

illustration techniques cause the user to interpret them too literally.

On the opposite end of the debate is [18] who states:

"Prevalent modeling notations such as Booch , OML, OMT, and UML are not sufficiently expressive in the constraints they can represent graphically. Consequently, the designer is forced to supplement modeling diagrams with constraints specified textually."

This side of the debate argues that the informal visualisation methods lead to ambiguity in the definition of patterns. [18] presents an extension to the UML formalisms which they believe enables the accurate representation of patterns as diagrams alone.

The examples provided which make use of this visualisation system are cumbersome and complex and do not succinctly convey the essence of the patterns. As a result this system of representation has not gained acceptance.

Throughout pattern literature a variety different approaches to illustrating the patterns have been used. These range from the rough hand-drawn sketches of [2] to the precise models of [18]. However, by far the most common approach is to use UML or some adaptation of UML [17], [21].

Many researchers adapt UML for the specification of patterns [17], [1], [10], [22], [18]. Another very common approach is to use UML in conjunction with some other illustrating format, often of the researcher's own invention [23], [21], [9]. Still other researchers abandon any well known notations and use their own to specify patterns [19], [16], [2].

2.2 Traceability

According to the IEEE, traceability is defined as the identification and documentation of derivation paths (upward) and allocation or flow down paths (downward) of work products in the work product hierarchy. This means that all artefacts in a project (requirements, documents, models, model elements, code) must be defined and they should be traceable from conception, through its entire development lifecycle, to its deployment, evolution and iterations in any of the lifecycle stages. In addition, all artefacts should be traceable in both the forward and backward direction enabling a person to trace from the implemented artefact back to its origin and vice versa.

Implementing requirements traceability provides two essential functions:

1. **It verifies that new systems comply with the specified requirements.**
   Neglecting to implement suitable requirements traceability procedures can lead to serious quality and control problems within a software development project.

2. **It accommodates impact analysis on proposed changes. This ensures the overall quality of a project.**

Change analysis and implementation is an expensive and error prone activity. There are many software development tools available that allow project artefacts to be built, such as Microsoft Visio and Rational Rose. However, these tools do not support the change analysis and implementation process. [24].

2.2.1 Benefits of Traceability

Traceability is considered a best practice [25] as it brings many benefits to a project.

Traceability brings **accountability** and **management** to a project. Artefacts can be tested and reviewed, and comparisons of the versions of artefacts can be made. Traceability can also be used to plan the order of development of artefacts by taking into account which artefacts rely on other artefacts. Better management decisions can be made because there is more information about all aspects of the project.

Traceability allows for the comparison of the **requirement specification and the final product**. This allows for the correlation between what the project stakeholders wanted and what was actually produced to be found.

Software is a continuously evolving product. **System evolution** relies on being able to reflect requirement changes in the relevant artefacts. Traceability shows the relationship between different artefacts and therefore simplifies **change management** and **impact analysis**.

All the benefits discussed above, if traceability is properly implemented, assure the **quality** of the product produced as there is better management and the final product meets the specified requirements. These factors all **increase the chance of the success** of a project [26].

2.2.2 Traceability Problem

Traceability has many benefits. Despite this, traceability usage remains rare. According to Scott Ambler, “It’s rare to find a software project team that can that can honestly claim full requirements traceability throughout a project, especially if the team uses object-orientated technology.” [27]

Poorly understood user requirements and unnecessary features incorporated in to projects are the cause of many failures. One third of all projects are successful while over half are faced with exceeding their budget and time or not meeting the requirements. Only 54 percent of the features that are in the initial design are implemented. The situation seems to be getting worse as this is a decrease from the 67 percent reported in 2000. Of the features that are successfully implemented about 45 percent are never used [26].

2.3 Traceability Patterns

Traceability patterns are a new category of pattern. They provide proven solutions to traceability problems. They provide the benefits of patterns to the complex task of traceability. The application of traceability patterns facilitates a well structured approach to traceability.

Traceability patterns are classified according to their functions. Justin Kelleher [28] has identified five classifications.

1. **Business Tracing Pattern** – These patterns provide a connection between the client and the organisation. The patterns link requirements to legal binding contracts.
2. **System Tracing Pattern** – This classification describes a traceability pattern between various stakeholders in a project.
3. **Design Tracing Pattern** – These patterns define the tracing between the requirements, architectural components and design components in any project.
4. **Test Tracing Pattern –** This classification describes tracing between the design and the testing in a project.
5. **Development Tracing Pattern –** Once the project has been successfully completed, the final system needs to be traced back to the contract.

The pattern classifications follow the development of a project from the definition of the project to its deployment. When developing a traceability solution, the user can implement patterns from each classification in order.

3. APPROACH

3.1 The Storage System

It was decided to use a MySQL database as the storage system for TraPT. This was largely because it makes simultaneous and distributed access easily possible.

Both the Pattern Encyclopaedia and the Pattern Creator access the same database which allows for close interoperability. Note that the system was specifically modularly designed such that the storage system can be altered with minimal effect.

3.2 The Pattern Encyclopaedia

Research was conducted in the fields of patterns and traceability. The observations made were used to create the Pattern Encyclopaedia.

The objectives of the Pattern Encyclopaedia are to enable users to learn about patterns and traceability, identify patterns that solve a problem, view the patterns and understand how patterns fit into the software development process. This allows the users to apply the patterns effectively.

Learning about patterns and traceability is facilitated by comprehensive information. The information includes definitions, applicability and examples. Academic papers and articles are used as a supplementary source of information.

The Encyclopaedia provides search functionality in three forms. The users can browse through a list of patters that are categorised and classified within the categories. Users that know certain attributes of a pattern can search for it on those attributes. Users that do not know anything about a pattern, or even if it exists, but have a problem to solve, can search for patterns by problem description.

An extensive catalogue of patterns is provided. This includes numerous traceability patterns as well as patterns for other stages of the software development process.

3.3 The Pattern Creator

The Pattern Creator was designed at a high level through the application of the MVC (Model-View-Control) pattern [13]. As such separate modules are designed with standard interfaces between them:

- The *model* is used to store all the data associated with a pattern.
- The *view* is used to allow the user to view the model in a variety of ways.
- The *control* allows the user to manipulate the model. This is usually done using a view to give the user access.

The functionality of the Pattern Creator can be logically split into pattern definition and pattern management.

3.3.1 Pattern Definition

Pattern definition regards the selection of a pattern form and then the definition of the pattern in terms of the fields of that form.

Within this functionality the pattern creator is able to enter text and to insert diagrams. These diagrams can either be loaded from external files, or created using the integrated diagramming tool.

3.3.2 Pattern Management

The knowledge management patterns of [19] were applied to the creation of patterns. These patterns suggest several features for the creation of any knowledge (in this case patterns). For instance, within pattern management a creation process (workflow) through which patterns must pass is defined. Within this workflow users play roles in moving the pattern toward publishing. Documents can be assigned to patterns.

3.4 Integration

The Pattern Encyclopaedia and Creator interface through the pattern storage mechanism. Each module can be used as a stand-alone program or as a complete pattern tool.

4. RESULTS

Testing was conducted on different levels. This is shown in Figure 1.

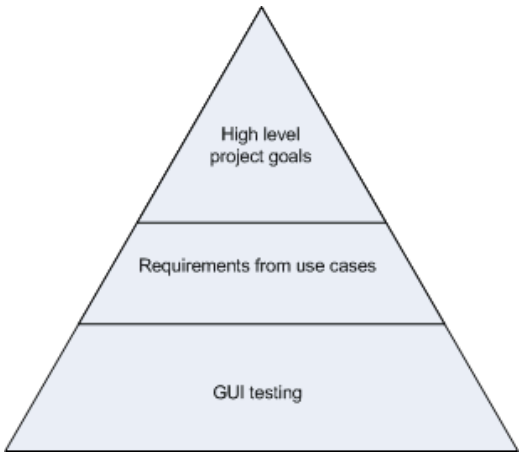

Figure 1: Hierarchy of testing conducted.

At the highest level is the validation of the project as a whole in terms of its goals. Does the system increase pattern usage and creation? On the second level is user testing to establish if the requirements of the system have been met.

Finally, at the lowest level there is testing of the actual user interface.

4.1 Project Validation

Ideally, an experiment into the effect of the system on an organisation should be conducted. The aim of such an experiment would be to show that using this system in an organisation would increase pattern usage and creation.

The methodology for such an experiment has been laid out and a prospective organisation has been identified. However, due to the time constraints placed on the project such an experiment could not be undertaken.

Two system demonstrations with industry experts were conducted. The aim of these interviews was to get some measure of project validation from the comments of these experts.

The experts indicated strong enthusiasm for the TraPT tool. It was noted by the experts that although they would like to use patterns in their organisations, this was not formally done as present. It was suggested that a structured tool for creating and accessing patterns would alleviate this problem and thus increase pattern usage.

4.2 Requirements Testing

Usability testing of the finer grained system requirements was conducted. This was done in order to ensure that the system met the requirements extracted at the start of the project. The results of this testing were largely positive and all system requirements were met.

4.3 GUI Testing

Low level testing of the system GUI's was conducted. This was largely done using heuristic testing [29].

Several minor GUI problems emerged during this testing. However, these had no lasting effect on the system.

5. CONCLUSIONS

5.1 Using TraPT should increase pattern usage

This conclusion follows from the expert interviews conducted. It is the opinion of these experts that the introduction of TraPT to their organisations would increase pattern usage and creation.

5.2 More testing is necessary

Due to time constraints the full project validation experiments could not be carried out. Thus, in order to validate this project conclusively more testing is required.

5.3 Pattern application is human intensive

It is concluded that this tool alone is not sufficient to increase pattern usage. Using pattern effectively is a human intensive activity. A considerable investment in time is required to gain the full benefits of patterns.

6. FUTURE WORK

The TraPT tool and research that was done in the process of creating the tool is part of a larger project on traceability patterns.

Thus future work can be partitioned into future work exclusively on this project and future work regarding the research which fostered the creation of TraPT.

Future work on TraPT itself is largely regarding the further validation of the project. It was concluded that further testing into the effect of using TraPT on pattern usage was necessary.

In terms of the greater research of which TraPT forms a part there is a lot of future work to be done. TraPT can be used as a tool or as the basis for other tools which will be used in that research.

Justin Kelleher is currently involved in his Doctoral studies on traceability and traceability patterns. He is defining a traceability patterns that are intended to aid the application of traceability in projects. The patterns are relevant to both software engineering and other disciplines.

Mikael Simmonson is currently working on the expression of traceability in UML diagrams. His research is part of his Masters Degree project. Mikael has found a way of structuring traceability of a project into directed graphs. This representation increases the amount of information that can be stored in traceability structures.

Both the TraPT tool and Mikael's work will form part of the Justin's PhD dissertation.

REFERENCES

[1] Schmidt, D. C. "Using Design Patterns to Develop Reusable Object-Oriented Communications Software," *Communications of the ACM*, 38(10) 1995.

[2] Coplien, J. O. *Software Patterns.* SIGS Books and Mulitmedia, 1996.

[3] Appleton, B., "Patterns and Software: Essential Concepts and Terminology," 2004, http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html.

[4] Wikipedia: The Free Encyclopedia, "Design pattern (computer science)," August 2004, http://en.wikipedia.org/wiki/Design_pattern_%28computer_science%29.

[5] Alexander, C., *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press, 1977.

[6] Correa, A. L., Werner, C. M. L., and Zaverucha, G. "Object Oriented Design Expertise Reuse: an Approach Based on Heuristics, Design Patterns and Anti-Patterns," Federal University of Rio de Janeiro.

[7] Gerth, T., Schachtschabel, R., and Schönefeld R.. "Using Patterns in Design and Documentation of Software," Technical University of Ilmenau.

[8] Prechalt, L., Unger, B., Philippsen, M., and Tichy W. "Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance," Universitäte Karlsruhe.

[9] Geyer-Schulz, A., and Hahsler, M. "Software Engineering with Analysis Patterns," Universit¨at Karlsruhe.

[10] Beck, K., Crocker, R., Coplien ,J. O., Dominick, L., Meszaros, G., Paulisch, F., and Vlissides, J. "Industrial Experience with Design Patterns."

[11] Harich, J., "Introduction to Software Patterns," September 2004, http://smile.jcon.org/soft/info/patterns/IntroductionToPatterns.html.

[12] The Hillside Group, "Patterns," September 2004, http://www.hillside.net/patterns

[13] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns*. Addison-Wesley, 1995.

[14] ChiliPLoP Conference 2004, "Pattern Central," April 2004. http://hillside.net/chiliplop/2004/patterns_central_2004.htm.

[15] The Hillside Group, "Pattern Conferences". http://hillside.net/conferences/.

[16] Riehle, D., and Züllighoven, H. "A Pattern Language for Tool Construction and Integration Based on the Tools and Materials Metaphor," University of Hamburg.

[17] Mak, J. K. H., Choy, C. S. T., and Lun, D. P. K. "Precise Modeling of Design Patterns in UML," The Hong Kong Polytechnic University.

[18] Lauder, A., and Kent, S. "Precise Visual Specification of Design Patterns," University of Brighton.

[19] Herman, T., Hoffman, M., Jahnke, I., Kienle, A., Kunau, G., Loser, K., and Menold, N. "Concepts for Usable Patterns of Groupware Applications," Informatics and Society, University of Dortmund.

[20] Schauer, R., and Keller, R. K. "Pattern Visualisation for Software Comprehension," Université de Montréal.

[21] Riehle, D. "Composite Design Patterns," Union Bank of Switzerland.

[22] MacDonald, S., Szafron, D., Schaeffer, J., Anvik, J., Bromling, S., and Tan, K. "Generative Design Patterns," University of Alberta.

[23] Lange, M. "Patterns for Testing Software," Gemplus GmbH.

[24] Sommerville, I., *Software Engineering 6th Edition*. Addison-Wesley Publishers Limited, 2001.

[25] Ramesh, B., and Jarke, M. Towards Reference Models for Requirements Traceability.

[26] What are your Requirements, 2003. The Standish Group International Inc.

[27] Ambler, S. Tracing Your Design. April 1999.

[28] Kelleher. J. Dissertation, Rough Draft, 2004.

[29] Nielson, J., "Heuristic Evaluation," http://www.useit.com/papers/heuristic/.