Reusing Through Requirements Traceability

Rob Pooley and Craig Warren
Heriot Watt University
Edinburgh
r.j.pooley@hw.ac.uk

Abstract

The Reuse of code artefacts can make development quicker, cheaper and more robust, but the process is complex and has many pitfalls: Code artefacts must exist, be available, be found, be understood, be valid and finally be integrated. Many software developers try to ensure that artefacts meet these requirements through a process of making code “reusable”. If the generation of reusable artefacts from developed code could be automated development time could be reduced through easier access. Ideally, code artefacts will be gathered and indexed automatically with no extra work added to the software development process. Our review of requirement management and version control tools identified a way to generate reuse artefacts through requirement traceability, using information from the requirement management system and code stored in the version control system. A prototype to search these indexed requirement artefacts, providing links to the implemented code, showed that this tool could substantially reduce development time for simple tasks. In a preliminary test, the prototype allowed discovery of multiple code artefacts to meet a single requirement in 63% of the time taken to implement a solution from scratch.

1. Introduction

While few software development processes take into account fully the relationship between the requirements of a project and the code created during implementation, there are tools which monitor changes to the code and match those changes to a requirement for the project. In the work described we show that, by monitoring these relationships and matching the requirements of a new project to those recorded for past projects, it may be possible to increase the efficiency of development by reusing the outcomes of any previous projects that are successfully matched. This idea will be referred to as “Reuse through Requirement Traceability” (RTRT).

2. Concepts used

The prototype uses existing tools for most of its functionality. These are basically a version control system as a code repository, a relational database which implements requirements management and querying, a command line interface for initialising the prototype and a Web interface for normal user interaction with the tool.

2.1 Version control

Version control (VC) allows software developers to store and manipulate multiple versions of a file. It allows recording of code changes so that old versions can be reinstated. VC systems allow for simultaneous development of artefacts within the same project or file by many developers.

Many VC tools exist; some are commercial products and others are open source solutions. Most Unix systems come with an open source VC system; initially Revision Control System (RCS), later replaced by Concurrent Versions System (CVS) and recently a replacement for CVS called Subversion (SVN) [1].

Commercial applications include Rational ClearCase [9], Telelogic Synergy [13] and MS Visual SourceSafe [16].

2.2 Software metrics

Software metrics [3] are currently a highly active area of research in software engineering. A wide range of software metrics have been proposed, to measure the complexity of both software artefacts (including designs) and the software development process. Software metrics can be split into 3 groups, source code metrics, design metrics and specification metrics [10]. Within each group several alternatives exist.

On reviewing this range of options, four types of code metrics have been identified which may be of value to a full RTRT system:
Source code metrics are the oldest form of metric. They can be as simple as number of lines of code used to meet a requirement or include measures of how complex the logic of the code is [4] or how strongly dependent modules are within the system. It is desirable to know how far a solution is coherent and how much coupling exists between modules to estimate how reusable these elements may be.

Developer Performance Metrics are the number of requirements a developer completes over a period of time. To be accurate, however, developer performance metrics should be combined with requirement complexity metrics, as described in the next paragraph.

Requirement Complexity Metric is a source code metric associated with a requirement. More complex code artefacts are harder to reuse [2]; this metric would allow a normalised estimate of the complexity of the code generated to meet a particular set of requirements, which could in turn lead a new user to select a solution with lower complexity. Intelligent selection of reusable solutions would lead to improved developer performance.

Error Metrics can report on the number of errors reported over a time period. This can indicate how close to completion a project is. Providing source code metrics [4] for the code to fix an error can indicate both the complexity of the change made and how well the development has progressed.

2.3 Requirement management

Requirement management, or software configuration management, monitors requirements and required changes throughout the software lifecycle so that a record (or trace) of what has been included and fixed in software is kept.

A change management system allows engineers or users to report bugs or required changes to a software project; these change requests are similar to requirements. Changes are audited either by project supervisors or testers performing code reviews and component testing. It is important that documentation of changes is well written and structured to help the auditing process make reuse easier in a future project.

Software configuration management is often considered to be about controlling, auditing and reporting on changes to a piece of software; to be able to control how people make changes to the software and make sure than the changes are correct and documented. During development, changes will be based on the requirements, but, afterwards, changes are made to fix defects and add new features.

In this paper the term “requirements management” refers to all types of requirement management including software configuration management and change management. Specifications generate requirements to be developed or further broken down into design points.

2.4 Existing change management tools

There are a few commercially available tools that meet some of our needs.

Rational ClearQuest performs the actions of a software configuration management system [9]. ClearQuest uses unique change identifiers to record of change requests. Descriptions of the required changes or bugs are provided by the person making the request. The record can be updated by a developer to document work done during the change. Any testing information that will help audit the change and a list of tests and results related to the change should be included as well. Change status is imported to ClearQuest, showing whether a change is still untouched, being worked on or completed and ready to be audited. ClearQuest handles requirements and changes in the same way; changes are requirements to edit existing code.

Telelogic Products provide a suite of tools designed to support the software development lifecycle, when used in combination [14].

2.5 Traceability

Traceability has been described as “The ability to trace the life of an artefact developed during the software lifecycle from its inception to its use” [12]. In this definition the inception of an artefact would be the creation of a requirement for it and the artefact would be realised in source code. Providing a link between the requirements model and the code artefact would be an implementation of traceability. This is merely a basic example of traceability. Traceability can be used to link test reports to code artefacts and bug reports, sections of UML models to traceability links are the focus of RTRT. They are relatively simple links and are easy to store. A study by Selby [10] found that module reuse could reduce the average “total development time per source line” by a significant factor when reuse was at the level of complete pre-existing modules. This is an impressive but plausible saving because total module reuse involves no editing of the code. Instead the time is spent finding the module to reuse and interfacing it with the current project. By providing direct links from requirements to code and supporting querying of the requirements, we gain a share of these benefits, although we do not believe we can necessarily match Selby’s rather idealised performance.
3. Design and implementation

The RTRT prototype consists of a requirements repository - stored in a MySQL relational database - a code repository - stored in the subversion version control system - and the RTRT system, which controls querying and updating of the repositories. The architecture is shown in Figure 1.

![RTRT system architecture](image)

Figure 1: RTRT system architecture

This architecture is based on the classic model-view-controller (MVC) pattern, where the two repositories form the model, storing the data for the system in one standard format. The RTRT system implements both the view, by allowing the user to examine the data in various ways without the data itself being changed, and the controller, which allows the user to direct the system to present different views.

There are two user interfaces to the system.

The Web Interface is a browser based application which allows users to interact with the requirements repository and to manage users, projects, requirements and code reuse. This is the normal means of interacting with the system when it is in use.

The command line interface generates command line scripts to manage the code repository while populating it and the requirements repository with the initial information about previous projects, in the form of matchable requirements with traceability links to coded solutions. It is not itself used for querying.

3.1 Database design

The requirements repository uses a relational database, in fact MySQL [7], to store information about the system users, projects being managed and the requirements of those projects. Requirements may contain notes written by developers and, in addition, contain changes which hold code samples and complexity numbers. Changes are also associated with a note marking any comments the developer made when checking-in the change.

Each requirement record includes:
- A unique identifier
- A meaningful name
- A type (Requirement or Change)
- A link to a project
- Its assigned user
- An assignment date
- A state (Not started, In Progress, On Hold, Ready for Audit and Complete)

Whenever a requirement is registered as completed, a record must be kept of the changes made in the code repository, using that requirement’s unique identifier. Recorded changes must be stored in the requirements repository in case the version control repository is deleted.

3.2 Version control technology: subversion

The subversion tool [1] provides version control facilities for the prototype. These include a remote storage location for the code repository, versioning of meta-files and directories and a change logging system. The repository also includes comparison scripts for viewing changes between versions; these can be used to derive changes in code to use in the requirements repository.

Although the prototype uses subversion it is easy to replace it with another version control system since the RTRT system interacts with it by calling its commands as part of a script. This allows the system to be converted to work with any version control tool that offers a command line interface, in almost any development environment.

3.3 Web application technology: Ruby on Rails

Ruby on Rails (Rails) [14] was used to implement the web interface. Rails is a framework based on the object oriented scripting language Ruby [15].

Ruby was developed by the Japanese developer Yukihiro Matsumoto in 1995 and is designed to be as flexible and powerful as a scripting language while still fully supporting object orientation; it is entirely open source and is aimed at making programming easier through a more natural syntax.

Rails is a web application frame work designed around the Ruby programming language. Rails is
designed to make it easier to develop, deploy and maintain a web application. Rails follows the MVC architecture for software development which was used in the RTRT system architecture described above. It provides scripts to generate models based on the tables of the database it is connected to. Thus, if a table exists called ‘users’ it will generate an object called ‘User’ which has all of the same attributes as the database table.

Rails uses the idea of “convention over configuration”, which means that the framework has a set of conventions which it follows by default for the running of the applications; there is no need to specifically configure it. However, you can still choose to override options yourself if you don’t want to follow these conventions. Rails also comes with scripts to generate the basic views you will need to perform standard procedures on your database. So the ‘users’ table from the example database above will also be used to generate a corresponding Web form, which can be used as the interface to create and edit user objects and save them to the database as well as a view which allows you to list all of the users currently in the database. All of Rails’ views are editable in HTML and embedded Ruby code.

This approach reduced greatly the time needed to implement the RTRT prototype. Certainly it was much quicker and easier to debug than a conventional scripting approach.

4. Evaluation

To see whether RTRT reduces development time, a series of tasks were performed measuring time to completion. Programmers were asked to write code to meet a requirement; half were given access to the RTRT prototype, half not.

We suggest that the RTRT idea should work better with complex reusability because when reusable code is found and used then the time saved is equivalent to the development time of that code minus the time to find and adapt the reusable solution; the more complex the code needed, the more time will be saved.

To get more useful data from the evaluation process, three tasks were defined, with different complexity. The time for each programmer to complete each task was recorded individually and the programmers’ perception of the difficulty was also recorded. This latter information is important because if the programmer is more experienced then the development time might be assumed to be lower overall. Test subjects were asked to evaluate the difficulty of each task on a scale of 1 to 10. Comparing two programmers’ perception of the difficulty of the same task then allowed us to compare their effectiveness in completing it.

For convenience, the subjects used were staff or students of the Department of Computer Science of Heriot Watt University and the programming language used was Java, which is the Department’s first programming language. This meant that there were few familiarization problems, since everyone was experienced in both Java and in the use of classes to implement modularity in an object oriented context.

4.1 The tests

We wanted to try out a range of tests, with different properties. It was particularly important to provide a range of complexity in the tests, since our assumption that more complex tasks would generate greater savings needed to be tested. Selby had noted a similar trend in his much more extensive study, but that was not in the context of a prototype tool.

Within the constraints of the project we performed tests using three tasks, of increasing complexity:

1. Simple Object
   Creation of an appropriate class for objects storing information about individuals: Included
   - a class defining an object.
   - Several instance variables within object.
   - Get and Set methods for all variables.
   - Simple constructor.

2. Complex Object
   Creation of classes to support a simple model of a bank providing services to its customers
   - a class defining an object.
   - Several instance variables within object.
   - Get and Set methods for all variables.
   - Complex constructor which takes arguments.
   - Instance methods to perform operations on the data in the class.

3. Main Method
   A complete program, implementing a class which
   - uses the main method.
   - Input and Output to the command line.
   - Program logic and loops.
4.2 Searchable solutions within the prototype

The RTRT prototype was pre-populated with a selection of sample solutions to the tasks. There were multiple samples for each task, some more useful than others. Some of the solutions which were provided were deliberately unhelpful for the tasks given.

**Task 1:** A full solution to the first task was available in the RTRT prototype, although there are four other classes which store information about people so the user is still required to select the correct class from the list. The full solution decreases the value of the first test for comparing the RTRT prototype to standard programming but allows users a chance to get used to using the prototype.

**Task 2:** A nearly complete solution for this task exists in the RTRT prototype. Only the method to add interest is not included. Two different types of bank account classes exist within the RTRT tool. 4 methods also exist to calculate interest, but only 1 is the correct method for the task set. This task provides RTRT testers to chance to edit an existing solution while standard developers practise their Java programming.

**Task 3:** The final task is the most complex and no sample solution exists in the RTRT prototype. There are samples programs with parts that can be combined to meet all of the requirements of the task but these must be identified by the programmer. Samples which need to be located are:
- Input and output streams
- Parsing integers
- Arrays
- Even or odd numbers
- For loops

This task allows for the best evaluation of the RTRT system, as both groups of programmers have the same complex problem to solve, but the RTRT programmers can find candidate solutions within the RTRT prototype, while the other programmers must search their memory or other sources for solutions. On the other hand, the RTRT users are required to frame their queries carefully, since there are several factors to be weighed up before deciding on the most appropriate artefacts to reuse. If this task shows gains in productivity, we can take the RTRT concept seriously.

4.3 Results

A total of twelve programmers helped test the RTRT prototype. Six performed the tasks using the prototype while the other six programmed as they would normally. The same persons were in each group for all the tasks and each group contained a random combination of experience and skill, with no attempt being made to pre-form groups with particular properties.

The full results are shown in table 3 at the end of this paper.

![Table 1: Average Task Time in seconds](image)

<table>
<thead>
<tr>
<th>Task</th>
<th>Average (RTRT)</th>
<th>Average (Without)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>152</td>
<td>347</td>
</tr>
<tr>
<td>Task 2</td>
<td>250</td>
<td>367</td>
</tr>
<tr>
<td>Task 3</td>
<td>698</td>
<td>1108</td>
</tr>
</tbody>
</table>

![Table 2: Average Programmer Skill](image)

<table>
<thead>
<tr>
<th>Programmer Skill</th>
<th>Average (RTRT)</th>
<th>Average (Without)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5.5</td>
<td>6.2</td>
</tr>
</tbody>
</table>

5. Conclusions and further work

We are aware that our results are based on a very small sample, and so our conclusions are quite tentative. We hope, however, to convince the reader that this approach is worthwhile.

5.1 Analysis

Clearly, the average time taken to complete each task is lower for those testers using the RTRT prototype. This includes task 3, the most complex.

It is possible that the tests could have been affected by varying abilities of the programmers in each group, but Table 2 shows that the average skill rating of the programmers using RTRT was lower than those not using the prototype. This would suggest that if the results where skewed by the abilities of the individual programmers they would be in favour of the group not using the RTRT prototype.

We can claim that RTRT seems to improve programmer productivity, but we need considerable further testing and evaluation to establish how general these benefits are. In particular, we want to look at how well RTRT works in realistic software production environments, when used for serious applications.
5.2 Future Testing

If further testing was to be carried out on the prototype to strengthen the evidence then the current set of tasks could be repeated with more subjects. These subjects could be taken from a larger population as well, including academics, experienced developers and people newer to the Java programming language. With more tests carried out over a greater range of programming skill it would be possible to make more secure conclusions based on characteristics of the sub-groups, such as beginner programmer versus daily programmer.

It would be of greatest interest to develop more tasks of increased difficulty to see if the RTRT prototype continues to perform as well as it did on these simpler tasks. Tasks involving integration of multiple code modules, more complex interfaces and logics structures would be a sensible next round of testing.

We need to refine the attributes used to provide the most effective results for users and to consider the effectiveness of our representation of requirements in driving the matching process. This could begin with analysis of what factors were actually the keys in matches used in our next tests and we intend to instrument the prototype to support this. We also need to revisit our choice of fields and to consider whether there might be other data which would direct us to better solutions or be better understood by our programmers.

5.3 Long term testing

True value is expected to be achieved from the RTRT concept by automatically generating the searchable code artefacts from standard development projects, rather than entering them by hand. In order to test whether this is possible it would be necessary to use the prototype over a long period as a development tool in a software development house. During the testing developers would use the prototype as their project management tool and report on its usefulness. To minimise the disruption to the software developers, records should be kept of all reuse searches performed and developers should be prompted after a search to find out whether the search yielded usable results.

Long term testing would also require further development of the prototype and a testing partner. It is unlikely any testing partner will want to switch their current project management tool, which suggests that a long term test would require the RTRT concept to be integrated into a project management tool.

5.4 Overall conclusions

Notwithstanding the limited opportunities for evaluation so far and the difficulties in extending these into real world situations, we feel able to claim some success for the approach. Certainly we are confident that this approach is worth serious consideration as an improvement over previous tools and techniques.

Our eventual goal is to see the functionality of RTRT integrated within project management tools. That would both allow validation and encourage the full use of the technique to gain benefits in improved efficiency of code production.

6. References

7. MySQL home page http://www.mysql.com/company/ (last accessed
10. Selby, R.W. Enabling Reuse-Based Software Development of Large-Scale Systems - IEEE


Table 3: Complete results

<table>
<thead>
<tr>
<th>Tester</th>
<th>RTRT Prototype</th>
<th>Skill</th>
<th>Time</th>
<th>Time</th>
<th>Time</th>
<th>Difficulty</th>
<th>Difficulty</th>
<th>Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FALSE</td>
<td>6</td>
<td>296</td>
<td>245</td>
<td>1073</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>FALSE</td>
<td>8</td>
<td>183</td>
<td>275</td>
<td>646</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>FALSE</td>
<td>5</td>
<td>706</td>
<td>419</td>
<td>1596</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>TRUE</td>
<td>5</td>
<td>432</td>
<td>257</td>
<td>523</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>TRUE</td>
<td>3</td>
<td>161</td>
<td>549</td>
<td>1221</td>
<td>2</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>TRUE</td>
<td>6</td>
<td>73</td>
<td>138</td>
<td>559</td>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>FALSE</td>
<td>4</td>
<td>644</td>
<td>791</td>
<td>1774</td>
<td>4</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>TRUE</td>
<td>6</td>
<td>110</td>
<td>235</td>
<td>906</td>
<td>1</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>TRUE</td>
<td>6</td>
<td>73</td>
<td>129</td>
<td>317</td>
<td>1</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>FALSE</td>
<td>7</td>
<td>202</td>
<td>313</td>
<td>915</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>TRUE</td>
<td>7</td>
<td>67</td>
<td>195</td>
<td>665</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>FALSE</td>
<td>7</td>
<td>51</td>
<td>161</td>
<td>645</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>