Chapter 3

Literature & Technology Survey

1. Introduction

Throughout this chapter, we explore existing technologies, reviewing what areas of the problem they solve and which requirements they are lacking. We also seek to confirm the need for such a technology to fully solve the issues we have identified, through researching the problem domain via published literature.

1. Overview

One of the expected outcomes of this dissertation is an online, collaborative platform to facilitate cross and multi project code reuse, effective code searching, and code sharing and peer review.

To begin, we must identify key points surrounding the current state of cross project code reuse:

* Has code reuse evolved over time, and how?
* What tools are there currently that attempt to partially or completely fulfil the goal of cross project, multi user code reuse?
* How have these tools evolved, and what were their original intentions?
* What usability features do these tools implement and utilise?
* Is there literature to support the idea that code reuse remains sub-optimal?
* Is there ongoing research in to new methods of code reuse or code sharing, specifically cross project?
* How do current developers feel about their existing environment in relation to code reuse, and what do they feel works well or poorly?

The majority of this chapter will explore each of these points in detail, with a focus on personal, smaller scale code reuse, which will be what is primarily referred to by the term ‘code reuse’.

1. Has code reuse evolved over time, and how?

As previously established in Chapter 2, code reuse in some primitive form has existed since the advent of coding itself, with programmers simply sharing pieces of code between them. Also established is that the research into code reuse can be traced back to Douglas McIlroy in 1968, and and his proposal for the software industry to be based on reusable components [5, 6]. Douglas saw the software industry in the same light as he saw a manufacturing industry – parts should be purchasable from suppliers to be used in building a more complex system, such as purchasing the individual parts required to build a car. He envisioned catalogues of interchangeable routines built to particular specifications that could be purchased and used, such as tyres for a vehicle, and their many technical specifications.

We can see, from modern day application, that these catalogues do not exist quite as Douglas’ had initially described, though they do exist in some form or another. First, consider component-based software engineering: the idea that software components can be made interchangeable and reliable, similar to hardware components [9]. Then, this clearly flows from Douglas’ idea of catalogues, but maintains a much more abstract view on how it should be implemented. This principle is what countless software companies utilise, enabling a product to be developed to perform a specific task, with exposed hooks or APIs, allowing it to be dropped in to a more complex system with relative ease.

A practical example, to aid understanding, would be a software called Card.io [10]. The software itself enables the user to take a picture of their debit or credit card, and then reads the card number and expiry date from the picture, filling out card payment fields automatically with no typing. The software exposes a number of APIs to allow for embedding in applications, and as such is used in a plethora of applications that take payments from a customer, for example the PayPal payment library v.zero [11], which itself is a modular component and can be embedded in further systems.

This demonstrates the commercialisation of Douglas’ ideas, but this is not the only evolution of reuse, specifically for the personal code reuse of the small-scale developer demographic. However, comparatively less literature has been published and few research efforts made regarding such a demographic, as noted by Norton [12] and Min-Sheng [13]. The majority of literature refers to large software organisations, but there is considerable need for personal code reuse as well. Norton notes that the majority of small-scale reuse is ad hoc and unstructured, whereas structuring such reuse would enable the developer to become much more efficient with their reuse, and so improve the benefits of engaging with it. The history of such personal reuse, therefore, is difficult to research, but Norton and Min-Sheng both note that there is no obvious practice or standard in place, unlike component-based software engineering.

Therefore, it is clear that code reuse remains a popular concept, and one which can be, and is, employed in multiple ways, dependent on individual ideals; from Douglas’ ideas on industrialisation of software development to the current day company repository and work methods, code reuse is ever popular. Despite this, small-scale code reuse has been given far less attention, despite it being applicable for individuals throughout their careers, and specifically those that work in relative isolation, such as students, hobbyists, consultants or freelancers [12]. In the next section, we look at potential tools these developers could use, as well as those available for larger organisations and the industry as a whole.

1. What tools are there currently that attempt to partially or completely fulfil the goal of cross project, multi user code reuse?

* *Research existing tools that have some form of storage, reuse or share*
  + *~~Codebase~~*
  + *~~git/GitHub~~*
  + *~~GitHub’s Gists~~*
  + *~~StackOverflow (code sharing)~~*
  + *~~Google Docs~~*
  + *~~IDE Plugins (e.g. Yasnippet, Resharper)~~*
  + *~~Pastebin/Codepad~~*
  + *~~Snipplr~~*
  + *Subversion*

In this section we will take a look at existing technologies related to the problem domain and analyse their functionality. At the end of the section, we list a set of features these tools may possess, related to fulfilling the original problem description set out in chapter 2, and examine how completely the tools fulfil these requirements.

*Codepad.org* is “an online compiler/interpreter, and a simple collaboration tool” [14] that allows users to type or paste code from one of 12 languages in to a multiline text box, and then it runs that code and outputs a short URL containing both the source code and any runtime output, which can then be shared by the end user. It is an exceptionally primitive code share tool, where source can be saved as ‘pastes’, and shared via URL. There is little other functionality, other than the ability to make a paste private or save a paste to your account.

*Pastebin.com* is similar to Codepad, in that it facilitates ‘copy paste’ code sharing via a short URL, but does not run the code. It contains more options such as language specific syntax highlighting, paste expiration settings, public or private pastes as well as giving it a title, but again does not expand further than simply ‘code paste’ sharing. Pastebin also exposes an API to allow other tools to utilise its method of sharing, though most seem to be browser plugins [15].

*Google Docs* allows users to create and edit documents online, including text documents, spreadsheets, slideshows and forms. While not specific to code, it boasts good online, collaborative features for editing documents, as well as cloud storage, and the ability to make documents private, public shareable or public editable. It also includes version history and roll backs for version control. Despite these features, it is not designed for code storage or retrieval. Retrieving code from the storage would quickly become cumbersome as no efficient categorisation or search tools are present, and performance is on par with NTFS. Therefore, Google Docs’ main appeal is real time collaboration and version control of documents.

*Stack Overflow* is a language-independent, collaboratively edited question and answer website, and contains a vast wealth of knowledge, and often examples too. Questions and answers are voted up and down, and edited in the same fashion as a wiki. Reputation is gained or lost from these votes, resulting in self-policing and consistent high quality content. These features relate to the problem domain as they help reduce stale code, and provide peer review for code snippets. They also allow editing of answers as programming languages evolve and standards change, which is something a static repository often fails to achieve. However, as it is a question and answer website, code sharing is reactive, and usually occurs as an example within an answer. It most certainly does not act as an adequate repository, despite its many usability and collaborative features. Despite this, it is worth noting for its usability features, as discussed.

*Integrated Development Environments (IDE)* may come with snippet tools. Sublime and Atom are both text editors for code and markup that include some form of snippet repository within them. These are offline, personal repositories, but the specific files can be zipped and sent to others, albeit with effort from both parties. The snippets are written and stored as files with some form of markup, for example XML. This then allows them to be named, and recalled simply by writing the name of the snippet. For code that is used often, this is a fast and efficient way to reuse code, but requires the developer remember both what snippets are available, and what name they all go by. Searching the repository is often cumbersome, and redundancy is quite possible, as snippets may go forgotten. As it’s an offline, personal repository, the problems relating to stale code and lack of peer review remain.

*IDE Plugins* are also an option, as some IDEs don’t provide tools for code reuse by default. Resharper for Visual Studio is one such plugin

*git/GitHub*

*GitHub’s Gists*

*CodeBase*

*Snipplr*

1. How have these tools evolved, and what were their original intentions?

* *Find git repos or other version controls for the above projects. Perhaps they were documented through development, or have blog posts relating to why and how they went about creating them. Possibly an about section on their website.*

1. What usability features do these tools implement and utilise?

* *Identified the following, talk about each briefly:*
  + *IDE Intergration via API*
  + *WYSIWYG Editor*
  + *Syntax Highlighting; Line Numbering (Possibly GeSHi?)*
  + *Rate, Pin/Favourite, Comment, Share*
  + *Searching/Sorting, Tagging, Indexing*
  + *Public/Private snippets/groups*
  + *User groups/companies/institutions*
  + *Forks/Change Requests/Branches*
  + *Quick code dump sharing (like pastebin)*
  + *Cloud storage/online editing (like google docs)*
  + *Related code*
  + *Collecting multiple snippets together*
  + *Facebook Login for easy account creation/future login*
  + *Profile – information about the user, their submitted code etc*
  + *Achievements (First submission, Top Rated, etc)*
  + *Social Media integration for sharing code*
  + *Wordpress Plugin*
  + *Ability to run code itself (codepad)*

1. Is there literature to support the idea that code reuse remains sub-optimal?

* *Research and review existing literature with key terms ‘code reuse’ amongst others, to identify content for this section*

1. Is there ongoing research in to new methods of code reuse or code sharing, specifically cross project?

* *Potential recent literature can cover this section. Maybe online research that has yet to be properly published can be found.*
* *[12, 13] – see this reference for information about personal code reuse, namely that not much research has been done on it, except this Norton bloke*

1. How do current developers feel about their existing environment in relation to code reuse, and what do they feel works well or poorly?

* *Developing survey to send to a number of companies and contacts that have agreed to be surveyed (Hex Digital, PayPal, Students of University of Bath, Freelancers).*
* *Survey development has started*

1. Conclusions

* *Here we will summarise what we have learnt from the above headings and apply it to the initial problem to decide how to continue. The idea is that this section proves the need for the deliverable.*