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 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.

Consider making this bit more prominent, that it’s not

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

Consider introduction – why these technologies, how did I find them, are they famous, popular, cited, etc. Consider including data surrounding total users. ‘The author of this dissertation has experience with x, and they are popular y’ ‘a search on google using those key words revealed x, y, z’

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.

The technologies are presented in order from the simplest to the most complex

Consider rewriting, or find out how to quote the website (Emma Edwards?)

Consider changing exceptionally primitive (Minimal? Basic?)

*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, such as Resharper for Visual Studio. Its aim is to improve upon Visual Studio’s snippets, though works in quite the same way as the IDE snippet tools mentioned above. Resharper however, can also predict which snippets you may want from the context of your code, among other things. Despite this functionality, it again is an offline, personal repository and maintains the same problems mentioned above that befall the IDE solutions.

*git* is a version control software, storing revisions of software in a distributed revision control system, and is widely used in the software development industry [16]. Users can create a repository which maintains a complete history and full version-tracking of itself. This repository can then be cloned to another location to be worked on, and this second repository also maintains a complete history and full version-tracking. Commits can then be pushed between repositories, and changes are merged to allow for seamless collaboration even on the same files. However, git repositories mirror that of a file system, and as such have no real search or sort methods to allow for quick navigation through code snippets. Instead they are used to track source code for individual projects. This would make it cumbersome to use, and although it improves on Google Docs, as it is code oriented, it does not satisfy the ease of access nature that a code reuse repository would require.

*GitHub’s Gists* initially look very similar to pastebin and codepad, in that the user may type or paste a snippet of code into a textarea, giving it a title, description and privacy options, and then share it via a URL with other users. However, once created, they are treated as a git repository, and can be cloned, revised and added to. This means they act as a combined pastebin & git, bringing with it the benefits of both, however this solution doesn’t improve on the negative points mentioned above for either, and so again would not be suitable as a solution to the problem at hand.

*Codebase* is an online repository hosting service that works with git and other similar repository services, but with project tracking features on top. Similarly, Codebase is used to track source code for individual projects, and also handles a number of other project management features, such as tickets, bug tracking, time tracking, customisable permissions, AGILE development and more. A number of these features would be useful in our ideal solution, however again Codebase falls short as it is cumbersome to extract snippets from a repository built in this manner.

*Snipplr* is an online snippet sharing website, and comes close to solving the problem. Users can create public or private (personal) snippets, containing a title, description, source code and other meta data, and save them to their own snippet library. If public, other users can comment on these snippets, favourite them or share them on social media. Snipplr has a number of search filters, including searching all snippets or just personal snippets, searching titles and descriptions, searching source or for specific tags. However, the search functionality is particularly verbose, and only allows for searching one of these filters at any time. Combination search is impossible, so complex filters cannot be created.

Users have profiles, and can gain points and achievements for their snippets through community recognition, in a similar fashion to StackOverflow, and high scoring snippets are more visible to the community. Users also can edit their own profiles to provide personal information about themselves.

At first look Snipplr seems to fit the bill for our problem domain, but a number of features are lacking. There is no version control or snapshot history for code snippets, and by extension there is no tracking information other than time and user of last edit. Additionally, Snipplr has no functionality geared towards sharing within a company or institution. If a company, such as a web development agency, wished to track reusable snippets and provide access to a list of employees, there is no process to do this with Snipplr. Similarly if an educational institution wished to share code snippets with a specific subset of students, Snipplr would be unable to handle this request.

New section. “From above, establish criteria for systems”. Go through criteria one by one. When is it done well and when it is not. Could break the table in two. Consider having yes/no/sort of, instead of just yes. Explain criteria. Name the tables ‘table x’. Give rank to systems, priority to criteria? All about the story. Explain why we need the criteria – eliminate the intuition. Link history in to section above instead. Could use screenshots/or even the same task shown by different systems. Summarise what the table demonstrates.

Read D. Garlan et al., IEEE Software <http://dx.doi.org//10.1109/MS.2009.86> July-August 2009 (why code reuse is so hard)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Paste Dump | Exposed  API | Publicity  Settings | Collab-  oration | Online | Peer  Review | Advanced  Search/Sort | Version  Control | | Grouping  Privacy | Personal  Repository |
| Pastebin | ✓ | ✓ | ✓ |  | ✓ |  |  |  |  | |  |
| Codepad | ✓ |  | ✓ |  | ✓ |  |  |  |  | |  |
| Google Docs |  | ✓ | ✓ | ✓ | ✓ |  |  | ✓ | ✓ | | ✓ |
| Stack Overflow |  | ✓ |  | ✓ | ✓ | ✓ | ✓ | ✓ |  | |  |
| IDE & Plugins |  | ✓ |  |  |  |  |  | ✓ |  | | ✓ |
| git |  | ✓ | ✓ | ✓ | ✓ |  |  | ✓ | ✓ | | ✓ |
| GitHub Gists | ✓ | ✓ | ✓ | ✓ | ✓ |  |  | ✓ | ✓ | | ✓ |
| Codebase |  | ✓ | ✓ | ✓ | ✓ | ✓ |  | ✓ | ✓ | | ✓ |
| Snipplr |  | ✓ | ✓ | ✓ | ✓ | ✓ |  |  |  | | ✓ |

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

*Pastebin* was created in 2002. But did not gain popularity until 2010, when it reached 1 million ‘active’ (not spam or expired) pastes [17]. In June 2015 they had reached 65 million active pastes, showing its continued growth and the popularity of its paste dump service. It has always provided a paste dump service, and continues to do so.

*Codepad* also continues doing today what it originally set out to do, compiling/interpreting code in an online, paste dump space.

*Google Docs* originated when Google acquired Upstartle in 2006 and through 2007 merged their web-based word processor with Google Spreadsheets [18]. The original intentions, stemming from Upstartle, were to allow users to share documents instantly, as well as collaborate in real-time. This continued with the addition of presentations to Google Docs, and was refined over time to include extra usability features.

*Stack Overflow* began in 2008 as a website dedicated to helping users seek assistance on programming related issues. Soon after in 2009, additional websites were created along the same premise, under the Stack Exchange umbrella. Stack Overflow itself continues to play a Q&A facilitator for programmers looking for answers to specific, closed loop questions, and is moderated by the community.

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?), PlainText, Embed*
  + *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)*
  + *Markdown for description or comments*
  + *Report snippets*

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.*