AN ABSTRACT OF THE THESIS OF

Sanchit Karve for the degree of Master of Science in Computer Science presented on September 6, 2012.

Title: Assessing Quality Attributes Of Open Source Software By Mining Low Ceremony Evidence.

Abstract approved:

_____

Christopher P. Scaffidi

Programmers often have to choose components online for reuse based on software quality. To help with this choice, most component repositories (SourceForge, CodeProject, etc.) provide information such as user ratings and reviews of components. However, the reusability of components is not immediately obvious from this material. To make things worse, reviews and other material could potentially contradict one another about reusability or any other issue. Moreover, there could be multiple components that claim to perform identical tasks in which case it becomes difficult to identify the most reusable component. This thesis presents the result of two studies aimed at discovering how to use material from a component repository to automatically characterize the reusability of components. It was found that two factors (out of three) were significant predictors of reusability. Additionally, a review summarizer was developed to summarize user reviews and return the overall opinion reflected by the comments.

Assessing Quality Attributes of Open Source Software by Mining Low Ceremony Evidence

by
Sanchit Karve

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented September 6, 2012
Commencement June 2013

Master of Science thesis of Sanchit Karve presented on September 6, 2012

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Sanchit Karve, Author

ACKNOWLEDGEMENTS

The writing of this thesis has been one of the most significant academic challenges I've ever had to face. There is no doubt that without the support of my advisor, Dr. Christopher Scaffidi, this thesis would not have been possible. I am truly indebted and thankful to him for all the guidance, patience and direction given to me throughout this difficult process. I am grateful to him for giving me the opportunity to work with him and learn from him. I wish to thank Dr. Carlos Jensen, Dr. Timothy Budd and my advisor for giving me one of the best classroom experiences at OSU. Every class of theirs made me appreciate concepts from a different perspective. Not only have they expanded my circle of knowledge but they have, in a way, taught me how to think and for that I'm extremely grateful. I thank Dr. Bella Bose for his immense help during my most stressful term in graduate school.

I also thank James Admire, Abdul Almorebah and Abbas Al Zawwad from my research group for their enthusiasm and help during my research and specially my friends Pingan Zhu, Ankur Shah, Kunal Kate, Karthick Subramanian, Vivek Jadye, Faezeh Bahmani, Theresa Migler-Von Dollen, Greg Gutshall, Jesse Hostetler, Sean McGregor, Jennifer Davidson, Aswin Raghavan, Vasanth Krishnamoorthy, Jing Zhao, Bharatwaj Appasamy and Vivek Selvaraj for a truly memorable experience at OSU. I've only known them for less than two years, but they will always be a cherished part of my life.

Finally, I thank my parents Capt. Sanjay Karve, Chitra Karve and my younger brother Sanket for their sacrifices that gave me the opportunity to study outside India. It is amazing how much your outlook and views on life, people, politics, religion and culture can change just by living in a foreign land. It has been a phenomenal experience for me and I owe it all to them. No language can express my gratitude for their encouragement and support.

This work was funded in part by NSF under Grant CCF- 1101107. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the sponsor.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

Assessing Quality Attributes of Open Source Software By Mining Low Ceremony Evidence

1 Introduction

Programmers must often choose components for reuse based on software quality. Incorrect decisions can cause users inconvenience, data loss, security breaches and monetary loss. Thus, programmers need effective approaches for assessing the qualities of components and other code.

This problem is particularly important as the only existing solutions available for evaluating software quality (formal verification and testing methods) are time consuming [37] and sometimes of little use [36]. The reason is that these approaches require access to the source code and/or specification of correct behavior. This information is not immediately available to programmers who often need to quickly choose the best component from a number of similar components on online repositories.

To help with this choice, most component repositories on the web provide material such as reviews and user ratings of components, but the reusability of components is not immediately obvious from this material. Information about reusability can be buried in material is typically long and textual, with information about reusability potentially buried amid substantial text that has nothing to do with reusability.

It is also difficult to determine if a component is reusable if it has mixed reviews (i.e. almost equal number of positive and negative ratings) or extremely positive reviews. In the first case, there is genuinely no way of knowing which side of the fence you should be on by looking purely at the distribution of ratings and comments. The latter is difficult to judge as it is not clear if the component has a high rating since everybody could use the component as-is (i.e., without modifications to the code) or since everybody could modify the component to suit their needs. There is also a high chance of missing out on a

crucial flaw in the component if it is buried among hundreds of exceedingly positive reviews.

We present an example from the CodeProject.com repository [43] which highlights these issues. The repository contains an article (which is a page that describes a downloadable component, see Figure 1) that demonstrates how to draw text over an arbitrary path by translating and rotating some characters to fit the geometry of the path when it changes its direction. The C# article has overwhelmingly positive reviews (all four and five star ratings) and out of about 30 comments which are mostly praiseworthy (such as "*great article! Thank you*" or just "*Thanks!*"), there exist only 2 comments which describe problems associated with the component that were apparent only when they tried to modify the code. Comments such as "I *want to write the value of the path-length on the path with following code: [user entered code] but I don't see any return value. Please help me!*" and "*I tried it like this: [user entered code] but the canvas stays empty. What's the reason for this?*" are clearly indicative of the component failing when modified.

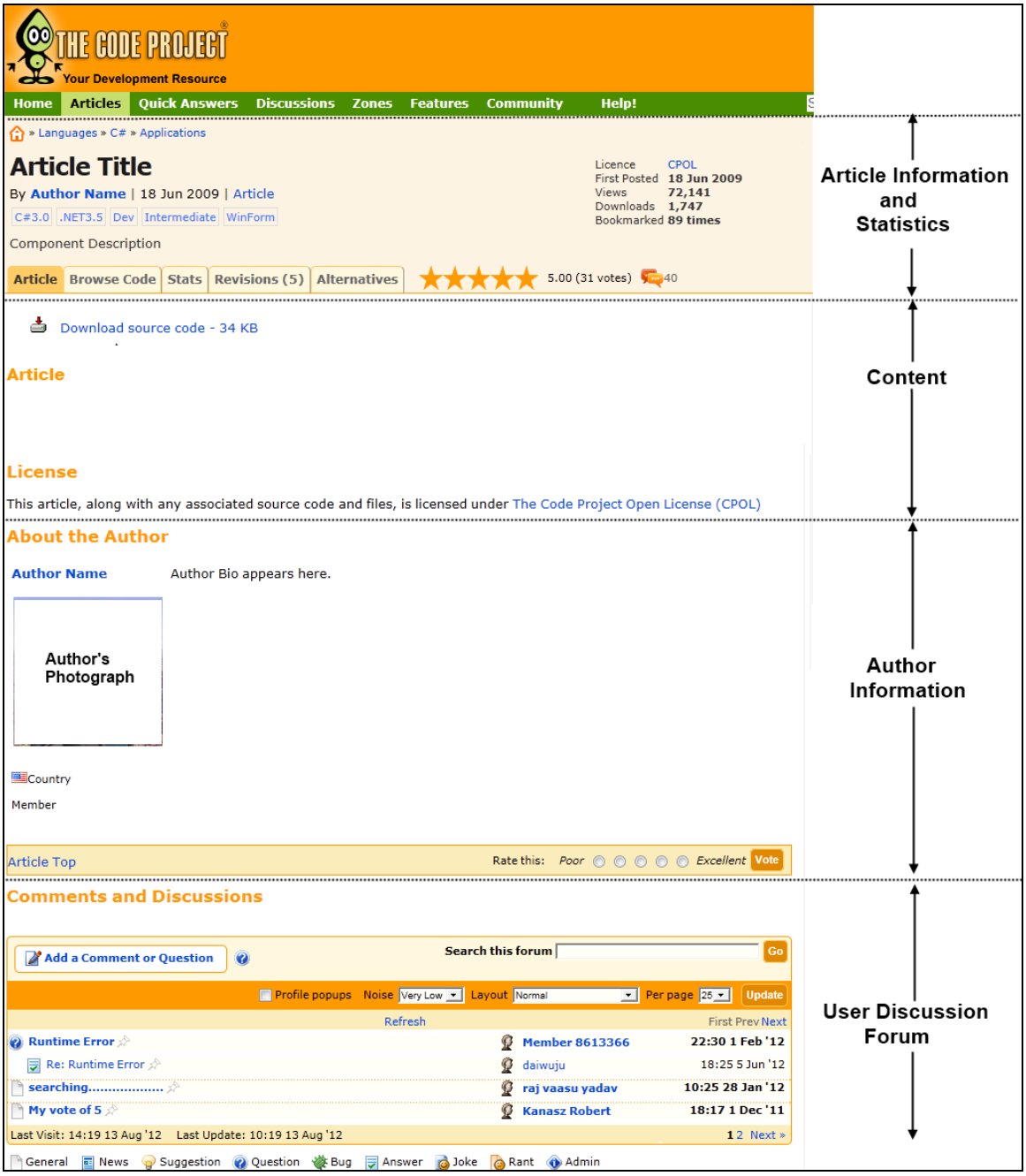

Figure 1: Typical Layout of an article on CodeProject.com

This thesis presents the result of two studies aimed at discovering how to use material from a component repository to automatically characterize the reusability of components. The first study was aimed at uncovering what kinds of information potentially related to reusability are present in material on a component repository. In this study, we automatically read text such as reviews off of a repository and computed statistics for each component that could potentially be indicative of reusability (such as Component Author's reputation, Component Author's expertise and Component Documentation Extensiveness). These computed statistics are typically informal, context-dependent and potentially contradictory and individually do not provide any information about reusability. Psychology studies in the past have shown that linear combinations of such informal statistics (or "cues") can be used to accurately model human decisions [8][10][13]. Since these cues are the only information available for a component on its web page, we aimed to derive a model that could predict the reusability of a component using combinations of cues (which in this case include user reviews and download counts of components). We believe it is a reasonable choice to include documentation related information in our set of cues as previous studies on software reuse [40] have shown that good documentation improves software reuse.

The second study investigated to what extent three factors extracted from these statistics were actually useful for estimating the reusability of components. We interviewed professional programmers who had previously downloaded and used components from this repository to provide us estimates of these components' reusability, and then performed multiple linear regression of these ratings against our three factors. We found that two out of the three factors were significant predictors of reusability (Component Author's Experience and Reputation and Component Documentation Extensiveness).

Overall, our approach investigates how well software quality can be assessed by using simple combinations of cues available for each component on its article page. We refer to

these cues as “Low Ceremony Evidence” (LCE): information that is typically incremental, informal, context-dependent, and frequently contradictory. This contrasts with quality evaluation techniques that rely on specifications or formal analysis of source code, which could be called “high ceremony” because of its formality. The information we call LCE has found widespread use in computer science outside of software engineering. We are the first to investigate its utility for assessing the reusability of software components.

The results of this thesis have potential applications in optimized search results, software maintenance and other areas. Search engines for code repositories could automatically guide programmers towards picking a more reusable component by sorting the most reusable components in search results. Software developers could also use the result of this research to detect areas of code that need maintenance. Ultimately, more research in this line of work would yield a scientifically sound approach for reasoning about software quality attributes when specifications and code are unavailable. By making better reasoned decisions, programmers will be able to avoid using components inappropriately, leading to fewer problems such as security breaches and software failures, whose costs are ultimately borne by users and society in general.

2 Related Work and Background

Five decades of psychology studies have shown that many human decisions can be accurately modeled with linear combinations of informal "cues" [8][10][13]. For example, people might guess that a man is old if he is mostly bald and his remaining hair is mostly gray. Such cues are typically incremental, informal, context-dependent, and frequently contradictory—what we call low ceremony evidence (LCE)—so they are not a guarantee of the conclusion's correctness. For example, neither baldness nor grayness of hair is proof of age. Rather, they each give evidence of age. Decision-making based on LCE is so widely accepted that numerous companies use it to provide recommendations. Examples include Consumer Reports rankings of products (based on product attributes), Money Magazine's rankings of Best Places to Live (based on city statistics), and US News and World Report's rankings of universities (based on school statistics and professors' opinions, under the assumption that the "wisdom of the crowd" is right [16]). Some recommendations come from linear models, others from checking if certain values exceed specified thresholds. For example, a LEED rating is determined by assigning points for each threshold met by a building's attributes, then comparing the total number of points to other thresholds [29]. The psychologists' experiments and LCE's widespread use indicate that LCE is a widely accepted basis for decision-making in everyday life.

The information we call LCE has found widespread use in computer science outside of software engineering. For example, machine learning research offers algorithms that judge if email is "spam" based on the presence of certain words (which thus serve as cues) [1]. Computer scientists have used "opinion words" such as "good" in an Amazon.com review to predict the numerical rating that accompanies the digital review [7], [11], and our own work has used such cues to provide enhanced product search tools for e-commerce shoppers [21]. The popularity of a video on YouTube.com or a news article on Digg.com at time t_1 can be used to predict its popularity at a later time t_2 [28].

These results show LCE is a viable basis for generating automated judgments about digital artifacts (in addition to physical objects in everyday life).

Field studies suggest that LCE plays a crucial role in decisions of software engineering practitioners. For example, one study of over 70 companies yielded seven principles that guide successful strategies for choosing which software to acquire [6]. These include selecting software that is popular, offered by well-qualified vendors, broadly compatible, and well-documented—all criteria that depend on the incremental, informal, context-dependent, contradictory information that we call LCE. Individual professional programmers are advised to select components using strikingly similar criteria, with the additional considerations of licensing restrictions and performance [9], [14], [16], [27]. These results indicate that LCE is already used by some practitioners, albeit informally, arguing for the viability of our plan to automatically mine and combine this information to make quality assessments.

Yet despite the strong evidence of LCE's potential usefulness for software engineering (which our workshop papers argue in more detail [23], [25]), LCE has not been extensively used to assess software quality. Instead, the dominant approaches for assessing software qualities are formal verification (e.g., [2], [5], [12], [19], and [30]) and systematic testing (e.g., [17], [18], [20], [26], [31]). These approaches require access to a checkable formal or informal specification (respectively) of correct behavior. What's more important is that formal verification requires access to source code. Systematic testing involves testing if a component conforms exhaustively to a specification and it's very difficult to do so without access to the source code. Techniques such as boundary value analysis, basic path testing and state transition testing are nearly impossible to perform without having access to the internals of the software. Source code and specifications are typically not available when programmers need to choose a binary component or web service from an online repository or a software vendor. And when

assessing the quality of a human element in a ULS context, the concepts of specification and code are essentially meaningless. To date, LCE-like information has only been used in two areas of software engineering. First, components' defect density has been predicted based on complexity metrics, code churn, and other cues that indirectly shed light on how hard code is to create and maintain [15]. In addition, research has yielded a machine learning model that uses code-based cues to fairly accurately predict if scripts of a certain type (web macros) would be reused [22], [24]. While these areas of work rely on LCE (incremental, informal, context-dependent, contradictory information), computing these cues requires access to source code. This thesis aims to eliminate this dependence on source code and specifications by testing how accurately non-code-based LCE can be used to assess quality attributes.

3 Methodology

One key insight to emerge from decades of artificial engineering research is that cues can sometimes be contradictory. Research has shown that adding more cues can reduce a system's confidence [44] [8] in an automatically generated judgment. However, most assessments are accurate as long as the cues are consistent most of the time. With this in mind, the thesis is structured into two studies. The first study's aim was to determine which LCE cues, if any, are mostly mutually consistent. This information would help group potentially redundant information into a single entity. The second study's aim is to test how accurate simple combinations of LCE cues are estimates of component reusability, revealing the effectiveness of LCE cues for predicting component reusability. The methodological approaches for both studies are described below.

3.1 Determining Mutually Consistent LCE Cues

Our first objective was to collect data from CodeProject.com servers and compile all the relevant information into a data set (stored as a comma separated values (CSV) file). To do that, we developed a web crawler to visit all C# component pages (about 1200) and extract information about each component and dumped it in a CSV file on a local hard drive. Once the data set was compiled, we searched for components with incomplete data and deleted them. All columns (i.e. cues) which contained only zeroes were also removed. We developed another tool to read the user comments posted on the components' article page and summarize them. We then used exploratory factor analysis (explained in [41] and [42]) on the collected data set to look for mutually consistent cues.

3.1.1 Caching Component Information from CodeProject.com

To find mutually consistent sets of LCE cues, we harvested 24 LCE cues (Table 3.1.1) for about 1200 C# components available on the CodeProject.com source code repository.

Table 3.1.1: LCE Cues extracted from the CodeProject.com repository

Popularity and certification	***Opinions reflected in programmers' comments***
(a) Average Rating	(g) Total # comments
(b) (Total # views) / (component age)	(h) # positive opinion words in comments
(c) (Total # bookmarks) / (component age)	(i) # negative opinion words in comments
(d) # external links that link to component's page	(j) # mentions of bugs in comments
(e) Certified by CodeProject as meeting standards	(k) # comments explicitly flagged as "rants"
(f) Total # Downloads	(l) # comments explicitly flagged as "questions"
Documentation, licensing and compatibility	***Component Author Information***
(m) # code examples in documentation	(r) # previous components released by author
(n) # paragraphs in documentation	(s) Authors' avg. rating on previous components
(o) # screenshots in documentation	(t) # total components released by author
(p) unrestricted software license offered	(u) Authors' avg. rating on all components
(q) # of compatible execution platforms	(v) Real author identity known
	(w) Author describes self as senior developer
	(x) Author identifies self as American

These 30 cues were motivated by relevant literature (above) which suggests that popularity, vendor qualifications, documentation, licensing and compatibility were considerations in programmers' choices of software. Previous work [22] has shown that Americans' web macros were more likely to be reused than non-Americans', and this might generalize to components created by professional programmers.

Most of the data for the LCE cues are available on the components' web page (Figure 1) with the exception of (r), (s), (t) and (u) (from Table 1), which are available on the component author's page on CodeProject.com. Since CodeProject.com lacks a publicly accessible web service to extract component-related information, we developed a web crawler (with the permission of a CodeProject.com administrator) to visit every article

page in the CodeProject C# Knowledgebase and compile a CSV (Comma Separated Values) file containing all the LCE cues for each article.

We removed all components with incomplete information. For example, the web crawler did not retrieve the total downloads or views for certain components (due to inconsistent HTML markup on some article pages). 23 components matched these criteria and were removed. Additionally, some columns contained zeroes for all components and were removed. For instance, CodeProject.com allows user comments to be explicitly marked as a “question”, “bug” or “rant” but not a single component had comments which were explicitly marked. Hence, we removed these 3 columns from our data set as they would be of no use.

The number of external incoming links (or backlinks) to the component page are not available on CodeProject.com and were retrieved by the web crawler using OpenSiteExplorer’s Web API [32]. Since each CodeProject article can only be covered under one software license, we used separate columns for each license (BSD, GPL, LGPL, etc.) in the CSV file and assigned a value of 1 if the license was covered by the component and zero if it was not.

3.1.2 Processing User Reviews

The web crawler also cached all the user comments from the discussion forum for each article and parsed the comments for positive and negative opinion words using the procedure as represented by a flowchart in Figure 2.

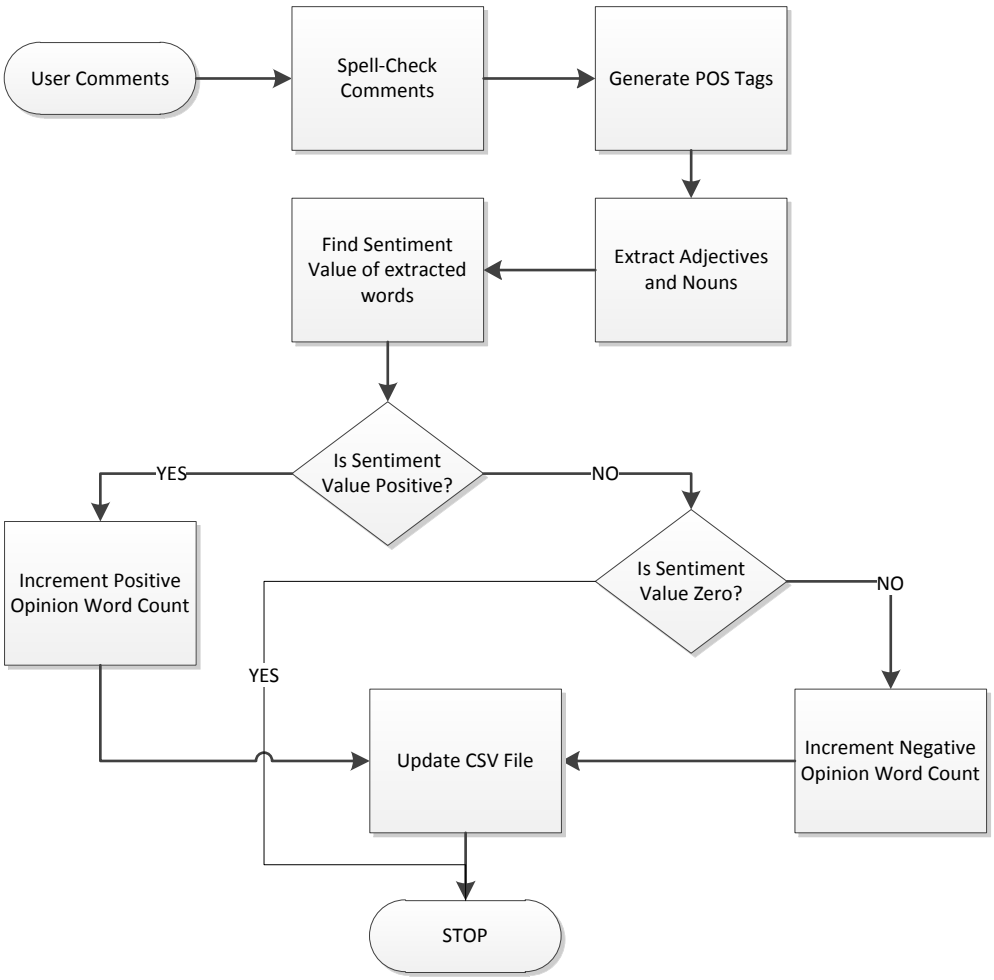

Figure 2: Flowchart of LCE extraction algorithm from user comments

The OpenNLP Natural Language Processing Library [33] was used to generate Parts-Of-Speech (POS) Tags [34] for each sentence in the user comments. However, in most cases the library failed to generate POS tags due to the fact that many comments were littered with incorrect use of punctuation and spelling errors. We solved this problem by passing the comments through a spell-checker (built along the lines of [3] and [4]) before letting OpenNLP parse them. All adjectives and nouns were extracted once the POS Tags were generated and sent to the SentiWordNet library [35] to retrieve a sentiment value (S_v) of the word. S_v is a number ranging from -1 to 1 where -1 represents “extremely negative” sentiment, 1 represents “extremely positive” sentiment and zero for no opinion. Hence,

we consider a word as being "positive" if $S_v > 0$ and negative if $S_v < 0$ and increment the LCE cues (h) and (i) (from Table 3.1.1) respectively when a sentiment value is retrieved.

We then analyzed the cues using exploratory factor analysis [41] [42], which identifies sets of variables that have high mutual correlation. Factor Analysis takes an input of columns and the estimated number of factors and returns the same columns with factor loadings assigned for each factor in the input. Our data set contained 28 columns (see Table 3.1.2) that represent the LCE cues listed in Table 3.1.1 which are passed to the factor analysis function as inputs. We assessed the agreement among the variables in each set using Cronbach's α, which is an accepted measure of consistency.

Table 3.1.2: Columns used as Input for Factor Analysis

Unedited – 1 if the article has not been edited by CodeProject editors, 0 otherwise.
Rating – Average rating of the component.
Downloads – Total number of downloads.
Age – Number of days since the component was first submitted.
TotalViews – Total Views of the component page.
TotalBookmarks – Total number of users who have bookmarked the component.
IncomingLinks – Number of external pages that link to the component URL.
TotalComments – Number of comments on the component discussion forum.
PositiveOpinionWords – Number of positive opinion words listed in the discussion forum.
NegativeOpinionWords – Number of negative opinion words listed in the discussion forum.
CodeExamplesInDocs – Number of source code examples in documentation.
ParasInDocs – Number of paragraphs (textual content) in documentation.
Screenshots – Number of screenshots in documentation.
IsCPOL – 1 if component covered under CodeProject License, 0 otherwise.
IsGPL – 1 if component covered under GPL license, 0 otherwise.
IsMSPL – 1 if component covered under Microsoft Public License, 0 otherwise.
IsLGPL – 1 if component covered under LGPL license, 0 otherwise.
RantComments – Number of comments explicitly marked as "rants".
QuestionComments – Number of comments explicitly marked as "questions".
BugMentions – Number of mentions of "bug" in comments.
CompatibleExecPlatforms – Number of platforms the component claims to work on.
PreviousComponentsByAuthor – Number of components posted by the author before the current component.
TotalComponentsByAuthor – Total components submitted by the author.
AvgRatingOnPreviousComponents – Average rating on previous components submitted by the author.
AvgRatingOnAllComponents – Average rating on all components submitted by the author.
AnonymousAuthor – 1 if author is anonymous, 0 otherwise.
IsAuthorSeniorDev – 1 if author describes him/herself as a senior developer, 0 otherwise.
IsAuthorAmerican – 1 if author describes him/herself as an American, 0 otherwise.

3.1.3 Cronbach's Alpha

The Coefficient Alpha was developed by Lee Cronbach [39] in 1951to provide a measure of the internal consistency of a test or scale; it is expressed as a number between 0 and 1. Internal consistency describes the extent to which all the items in a test measure the same concept or construct and hence it is connected to the inter-relatedness of the items within the test. Internal consistency should be determined before a test can be employed for research or examination purposes to ensure validity. In addition, reliability estimates show the amount of measurement error in a test. Put simply, this interpretation of reliability is the correlation of test with itself. As the estimate of reliability increases, the fraction of a test score that is attributable to error will decrease. It is of note that the reliability of a test reveals the effect of measurement error on the observed score of a group than on an individual.

There are different reports about the acceptable values of Cronbach's alpha [38], ranging from 0.70 to 1.0. A low value of alpha (< 0.50) is usually due to poor correlation between items. A high value of alpha (> 0.90) is excellent but may suggest redundancies [38] and show that the test length be shortened.

3.2 Estimating Reusability with Simple Combinations of LCE Cues

In order to test how well combinations of LCE can predict reusability, an assessment of components' actual reusability is necessary. To obtain this information, we conducted a user study where we asked programmers who use CodeProject.com to identify a C# component that they have tried to reuse, rate its reusability (on a Likert scale), to explain what led them to believe it was reusable and to explain any problems that they encountered. We recruited participants with the help of a CodeProject.com administrator and interviewed them using a web-based chat application that we created. Each participant was asked to review up to five components and was asked a set of questions

about their reuse experiences including a reusability rating on a scale of 1-5 for each reviewed component. At the end of the user study, we used multiple linear regression to estimate the participant's reusability rating based on the factor scores that were obtained as a result of factor analysis.

3.2.1 Recruitment of Participants

Programmers from all over the world were recruited through an invitation sent via the CodeProject.com monthly newsletter as well as a "tweet" on CodeProject.com's Twitter page (see Appendix). They were given an option to participate either via telephone or online chat and were allowed to review up to 5 components hosted on CodeProject.com. Among those who had applied, only those who wrote software at their place of work were selected. The screening process was conducted by asking the participants (via email/telephone) if they wrote code professionally and anybody who answered in the affirmative was considered to have passed the criteria. The participants were paid 10$ (via PayPal) for each component they reviewed up to a maximum of 50$ per person. Each participant was assigned a 1 hour time slot and asked to visit a web-based chat room application (see Figure 3) which we developed for the purpose of this study. The chartroom was accessible only to a participant (by use of a unique identifier appended to the chat room URL) during the assigned interview slot to protect the participant's identity. All conversations were logged by the chat applications along with timestamps.

Figure 3: Screenshot of the web-based chat application developed for the user study

Participants were asked to answer three questions for each component that they reviewed; which are listed as follows:

1) "What's the URL of component that you tried to reuse?"
2) "What problems, if any, did you encounter while trying to reuse this component?"
3) "When you downloaded the component, what led you to believe that it was reusable?"
4) "On a scale of 1 to 5, where 1 is very hard to reuse and 5 is very easy to reuse, how would you rate the reusability of this component?"

All the collected component reviews were compiled into a CSV file following which all chat conversations were "anonymized" by removing all email addresses from the chat log. Additionally, all occurrences of the participant's name were replaced with a unique

string. We did not maintain any links from the unique string to the participant's identity as an additional measure to protect the identity of the study participant.

After the user study data was "anonymized", we looked at all the user responses for question 2 and kept count of the total number of users that cited each kind of problem. The types of reusability problems were not decided before-hand and were created purely based on user responses. A similar approach was used for question 3, where all the list of reasons provided by the users was tabulated with the number of the users that cited the same reason.

3.2.2 Regression Analysis on Participant Data

We used multiple linear regression to determine if there was a relationship between any of the factors (obtained through factor analysis) and the participants' reusability ratings. If a relationship did exist, we would take note of the r^2 value to judge its predictive power. We considered results to be significant if the p-values were less than or equal to 0.05.

4 Results

As described in the previous section, we performed two studies to determine which LCE cues (if any) were mutually consistent and test simple combinations of LCE cues to gauge accuracy of reusability predictions.

4.1 Generating reusability scores for CodeProject components

We performed factor analysis on the data set (N=1176) we compiled by using our approach detailed in section 3.1. A Cattell scree test (Figure 4) was performed to get a sense of the potential number of factors present in the data set.

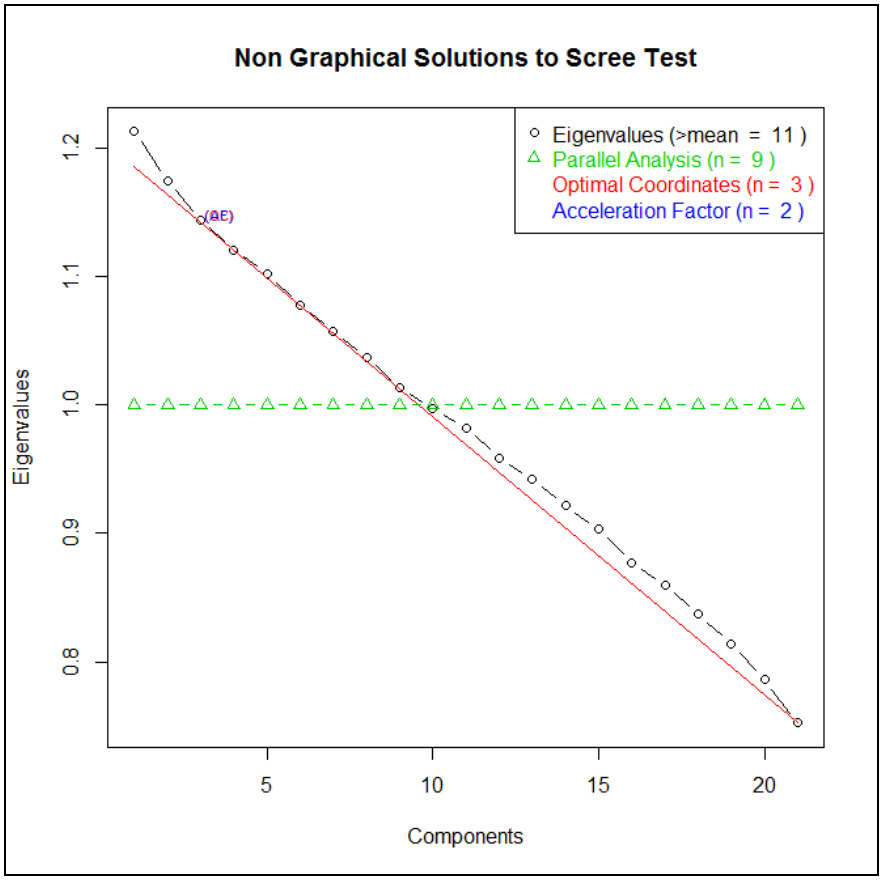

Figure 4: Scree plot for CodeProject component data set

Since there is no definitive method to determine the number of factors, we arrived at three potential factors as it satisfies Kaiser's criterion (described in Section 3.1.3).

Since we hypothesize correlated LCE cues, we used various oblique rotations to find a model for picking our factors. The promax rotation yielded the lowest factor loadings greater than zero (factor loadings can exceed 1.0). All LCE cues with high uniqueness values (uniqueness > 0.84) were discarded while the remaining cues were categorized into a factor which had the highest factor loadings among the three factors. The factor analysis categorized the cues under three factors and reduced the original set of LCE cues from 28 (Table 3.1.2) into 10 as listed in Table 4.1. The Cronbach's α obtained for factors F1, F2 and F3 were 0.53, 0.76 and 0.59 respectively (N = 1176). The internal consistencies for factors F1 and F3 are short of acceptable while that for factor F2 is good. The validity of these three factors as measures of reusability is assessed in the second study.

Table 4.1: Factors identified by factor analysis

F1: User Opinions	*F2: Author Information*	*F3: Documentation*
(A) (# bookmarks) / (component age)	(E) # previous components by author	(H) # code examples
(B) Total # comments	(F) Total # components released by author	(I) # paragraphs
(C) # positive opinion words	(G) Authors' avg. rating on previous components	(J) # screenshots
(D) # of backlinks to component page		

Factor Scores were computed using the factor loadings for each contributing factor and compiled into a CSV file. The scoring formula for each factor is listed as follows:

- F1 Score = (0.764 * A) + (0.949 * B) + (0.818 * C) + (0.387 * D)
- F2 Score = (0.809 * E) + (1.027 * F) + (0.296 * G)
- F3 Score = (0.534 * H) + (0.636 * I) + (0.38 * J)
  (where letters A to J represent LCE cues from Table 4.1)

4.2 User study

We received 45 responses as a result of our study being advertised on CodeProject's newsletter and Twitter account. When we contacted these respondents and explained the study in detail, 11 agreed to participate. Each programmer preferred participating via online chat instead of a telephone call. We present some interesting observations from the data that the programmers' shared with us as well as the result of our regression to test the predictive power of our factor scoring model.

4.2.1 User Responses

As listed in section 3.2.1, we asked our participants about their experiences while trying to reuse their chosen components – more specifically, their problems related to reuse and why they felt their selected components appeared to be reusable. Tables 4.2.1 through 4.2.4 show a comparison of different reasons provided by the participants for these questions.

Table 4.2.1: Participant responses for intended type of reuse (N=36 component reviews)

Type of Reuse Intended	Number of Responses
"Out of the box" Reuse	4
Reuse by Modification	20
Used as Reference	6
Did not mention or motive unclear	8

Table 4.2.2: Reasons voluntarily offered by participants for why their selected component had no reusability issues (N=15 component reviews)

Reason why Participant did not have any issue with Reuse	Number of Responses
Clear Explanation of features in Documentation	10
Source Code was well written	6
Did not provide a reason	3

Table 4.2.3: Participant responses for problems encountered while trying to reuse their selected component (N=36 component reviews)

Type of problem encountered while trying to reuse component	Number of Responses
Found the component too complex to understand	7
Not sure how the library code works	15
Not sure how to reuse the code	9
Component did not work as advertised	2
Other	3
None	15

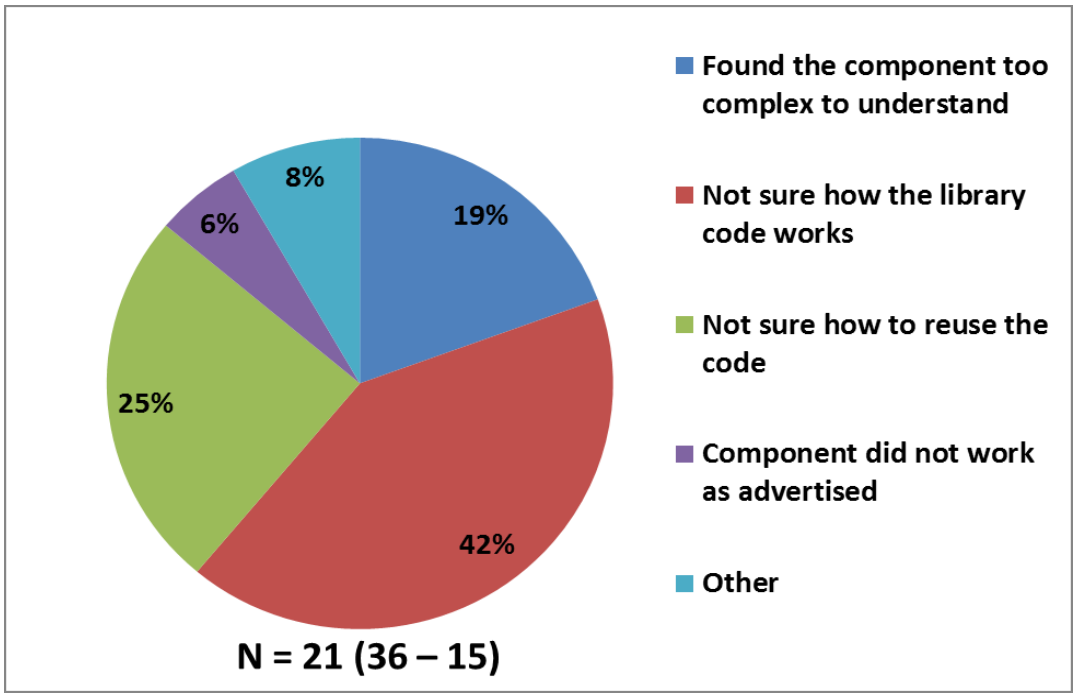

Figure 5: Types of Problems Encountered By Programmers

Table 4.2.4: Participant responses for why their selected components appeared to be reusable (N=36 component reviews)

Why Components appeared to be reusable	Number of Responses
Lists features I'm interested in	5
Code Examples	11
Demo Application	2
Docs claimed to be extendable	1
No other alternative component found	4
Clear and Detailed Documentation Text	13
Success of Reuse Reflected in Comments	2
Screenshots	9
Article Title/Description similar to what participant wanted	14
Good Articles Previously Written by Author	1
Used an API previously familiar to participant	1

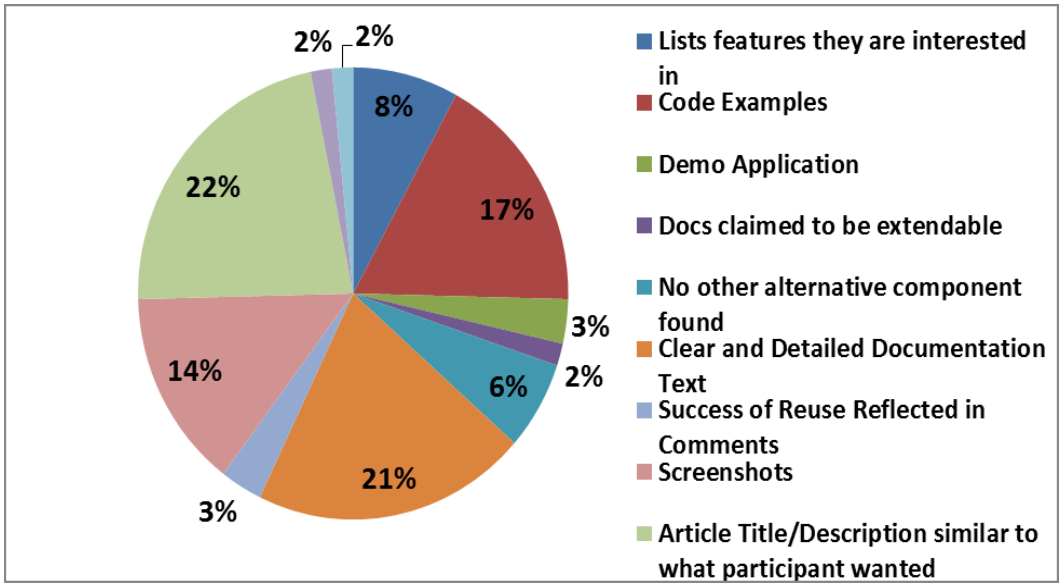

Figure 6: Reasons Why Programmers Felt Component Was Reusable (N=36)

Our participant-base comprised of programmers who chose components for black box and white box reuse. A component is said to be reused as a "black box", if it is used without any modifications. In this type of reuse, the programmer has no knowledge of the

component's inner workings, and only supplies it with the required inputs to obtain an output which is either used as input to other functions or just displayed back to the user. White box reuse is referred to the scenario where a programmer modifies a component's source code to suit his/her own requirements. This requires the programmer to understand how a component works "under the hood" (i.e., at the lowest form of abstraction) so that modifications are possible.

All the participants who reused components "as-is" (black box reuse) had no problems with reusability and rated the components as highly reusable (i.e., with a score of 5) while a majority of those aiming to adapt the code to suit their needs had issues with the component. In 42% of the cases, the programmers had no idea how the library worked and faced problems in figuring out which parts of code to modify due to lack of comments and unintuitive naming conventions. A quarter of reusability problems were attributed to the fact that the participants had no idea how to reuse or extend the code for use in their own applications. Unknown third-party components and dependency issues were cited as reasons for not knowing how to reuse the component. One participant when asked on his experience trying to reuse a component said, "*Since I wanted to use this control as a starting point for a much more specialized color picker component, I had to modify the code. It turned out that the author did do a lot about making his control reusable, but not in a way that the code is reusable (easily to alter) but only the component*". Another participant tried to explain why he picked one component over the other and said, "*I rate it higher than ZedGraph, since here the problem was in code modification - the component itself was quite easy to use; which was not so the case from the ZedGraph component*".

We believe that lack of detailed documentation caused most people to spend a lot of extra time trying to figure out how the component worked so that they could modify it. This is reflected in Table 4.2.3 where a majority of participants had no idea how the component

source code worked and is also consistent with prior research [40] which states that documentation matters in software reuse by explaining that "*Reuse is something that is far easier to say than do. Doing it requires both good design and very good documentation. Even when we see good design, which is still infrequently, we won't see the components reused without good documentation*". One participant (among many others) expressed his displeasure with the lack of documentation by saying, "*there is no documentation as such and I have to guess the functionality I needed by [looking at] the function names. Some are intuitive but [it took] a lot of work to figure out which function to use.*"

Ironically, it is the documentation (see Table 4.2.4) that seems to convince developers that a component is reusable in the first place. In 54% of the responses regarding why the participants thought their selected component appeared to be useful, they cited detailed explanations with accompanying screenshots and code examples. It may well be the case that most components on CodeProject have documentation suited for "as-is" reuse rather than reuse by modification.

It is also important to note that although 54% of perceived reusability was due to documentation, we believe this figure is actually a lot larger as 23% of those responses explicitly stated that they thought a component was reusable simply because the description was similar to what they really wanted. Other reasons given for the perceived reusability were:

- Positive user comments.
- Use of APIs previously used by developer.
- Quality of previous articles written by the component author.

Ten programmer responses also mentioned reasons why they did not encounter any problems while attempting to reuse their chosen components. Two reasons of praise were cited for this justification – namely clear explanation of features in documentation and

good quality of the component source code. These response types have been tabulated in Table 4.2.2.

4.2.2 Evaluation of Factor Scoring Model

Interviewees provided 38 ratings of the open source components available on the website, enabling us to test if there was any relationship between any of the three factors and the participants' reusability ratings. We discarded 2 component ratings (from a total of 38) as they did not have a downloadable component on CodeProject.com.

After performing regression of the participant rating (response vector) against F1, F2 and F3 (linear predictors) together (N=36), we obtained p-values of 0.72, 0.03 and 0.01 with coefficient values of -3.45×10^{-5}, 9.83×10^{-3} and 6.82×10^{-3} respectively for each factor. The analysis showed that F2 and F3 are significant predictors of participant reusability ratings with a moderate level of predictiveness (r^2 is 0.347, p-value = 0.0038). The coefficient values indicate that the reusability rating provided by programmers slightly increases with an increase in scores for factors F2 and F3.

Let us reexamine the example stated at the beginning of this thesis that presented a component whose reusability is difficult to assess due to conflicting ratings and reviews. The contradicting opinions which prevented us earlier from making a well-reasoned decision can now be made using the factors found by our factor analysis output. This time, we can just look at the component author's posting experience (F2) as well as the extensiveness of the documentation (F3) and notice that the author has posted a total of only three articles of which two were posted before the component in question. The articles posted before also have a lower average rating on the website (4.4) than the component under consideration (4.9). The documentation present at the component page is also just a few paragraphs long with just 3 screenshots. The coefficient values are an

indication that an increase in factors F2 and F3 leads to an increase in the rating. Since F2 and F3 are low, the rating would not be inflated and would yield a reusability rating of about 3.0 out of 5, which considering the reusability problems expressed in the comments, is a much more realistic score in the sense that users can now expect to have some issues with reusing the component even though it is reusable. The original score of 4.9 gives the impression that the component can be reused with almost no problems even though that is not true.

5 Conclusions and Future Research Opportunities

This thesis described our investigation into predicting reusability for open source components hosted on CodeProject.com. We used previous empirical work from relevant studies to derive our central hypothesis, which is that software quality can be assessed by using simple combinations of LCE cues available for each component on its article page [22], [38]. We conducted two studies – one to identify candidate factors for estimating reusability, and another to test the performance of these factors by comparison to reusability ratings provided by professional programmers.

Given the statistical results, we can conclude that the CodeProject repository contains three kinds of information (User Activity, Documentation Extensiveness and Author Reputation) of which two (Documentation Extensiveness and Author Reputation) are actually predictive of reusability. The factor-based scoring scheme proposed has a moderate predictive power of reusability ($r^2 = 0.347$) and a p-value of 0.003 (significant). However, our model may not work well with components that do not appear to be reusable since we only tested it with components that look reusable.

Future work could aim to improve predictiveness of the proposed model by aiming for a higher r^2 value. One possible method of increasing the predictive power of this model would be to investigate the distribution of component ratings based on type of reuse (as-is or reuse by modification). Our investigation showed that a component's overall rating on the website did not agree with the LCE cues investigated in this thesis. It is possible that the ratings are indicative of reusability but it's also likely that it could be representative of something else. It is important to learn why some users choose to review components while others don't. Users may be motivated to post a comment if a component did not meet their expectations rather than if it did. If that is true, it is likely that these components are highly ranked based only their value as an "as-is" reusable component

instead of their overall reusability. An in-depth study into reviewer motivations would help improve the reusability estimates of this model. We could also investigate whether different factors are useful for estimating black box reusability versus white box reusability, resulting in two different models specialized for each kind of estimate.

Our overall method of harvesting data, extracting factors and testing factors with interview data could be retested with more participants and with other online source code repositories (such as SourceForge, Google Code, GitHub, etc.) to verify if similar results are obtained across all repositories. Although we did write a tool to summarize user reviews on CodeProject.com, we did not verify the accuracy of the tool with a user study. An automated method to calculate the overall attitude towards a component could influence a component's reusability rating.

The research could be used to design improved systems to help people to promote reuse of components. For example, our scoring system could be implemented in a source code search engine where search results are sorted in decreasing order of their "reusability scores". The same scoring model could include summarized user reviews in order to help users learn more about components. Potential benefits to the component author could also be explored. For example, the review summaries and reusability scores may be able to help component developers discover areas of the components that need improvement. And finally, since our reusability scores are derived using LCE cues, some of which (documentation) are under the control of the author, it throws our model at risk of being deliberately manipulated to artificially inflate a component's rating. Various measures could be investigated and adopted in the future to prevent authors from fooling the system to provide misleading results.

Bibliography

[1] ANDROUTSOPOULOS, I., KOUTSIAS, J., CHANDRINOS, K., AND SPYROPOULOS, C. 2000. An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages. *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 160-167.

[2] BALL, T., AND RAJAMANI, S. 2002. The SLAM project: Debugging system software via static analysis. *POPL '02: Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* ACM, 1-3.

[3] WHITELAW, C., & HUTCHINSON, B. (2009). Using the web for language independent spellchecking and autocorrection. *Conference on Empirical Methods in Natural Language Processing*, (August), 890–899.

[4] CHOUDHURY, M., & THOMAS, M. (2007). How difficult is it to develop a perfect spell-checker? A cross-linguistic analysis through complex network approach. A*ssociation for Computational Linguistics*, 81–88.

[5] CLARKE, E., EMERSON, E., AND SISTLA, A. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*. *8*, 2, 263.

[6] DAMSGAARD, J., AND KARLSBJERG, J. 2010. Seven principles for selecting software packages. *Communications of ACM*. *53*, 8, 63-71.

[7] DAVE, K., LAWRENCE, S., AND PENNOCK, D. 2003. Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. *WWW '03: Proceedings of the 12th International Conference on World Wide Web* ACM, 519-528.

[8] EINHORN, H., AND HOGARTH, R. 1981. Behavioral decision theory: Processes of judgment and choice. *Annual Review of Psychology*. *32*, 1, 53-88.

[9] HAEFLIGER, S., VON KROGH, G., AND SPAETH, S. 2008. Code reuse in open source software. *Management Science*. *54*, 1, 180-193.

[10] HASTIE, R., AND DAWES, R. 2001. *Rational Choice in an Uncertain World: The Psychology of Judgment and Decision Making*. Sage Publications.

[11] HU, M., AND LIU, B. 2004. Mining and summarizing customer reviews. *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 168-177.

[12] JACKSON, D. 2006. Dependable software by design. *Scientific American*. *294*, 6, 68.

[13] KARELAIA, N., AND HOGARTH, R. 2008. Determinants of linear judgment: A meta-analysis of lens model studies. *Psychological Bulletin*. *134*, 3, 404.

[14] MICHAELS, L. 1996. In Search of a Portable Screen Library. *Dr. Dobbs Journal*. Sep 1996, http://www.drdobbs.com/184403226

[15] MOSER, R., PEDRYCZ, W., AND SUCCI, G. 2008. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction.

Proceedings of the 30th International Conference on Software Engineering, 181-190.

[16] NELSON, M. 2002. Inside Intel's JPEG Library. *Dr. Dobbs Journal*. July 2002, http://www.drdobbs.com/184405097

[17] NTAFOS, S. 1988. A comparison of some structural testing strategies. *IEEE Transactions on Software Engineering*. *14*, 6, 868-874.

[18] OFFUTT, A., PAN, J., TEWARY, K., AND ZHANG, T. 1996. An experimental evaluation of data flow and mutation testing. *Software: Practice and Experience*. *26*, 2, 165-176.

[19] OWRE, S., RAJAN, S., RUSHBY, J., SHANKAR, N., AND SRIVAS, M. 1996. PVS: Combining specification, proof checking, and model checking. *Computer Aided Verification*, 411-414.

[20] ROTHERMEL, G., UNTCH, R., CHU, C., AND HARROLD, M. 2001. Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering*, 929-948.

[21] SCAFFIDI, C., BIERHOFF, K., CHANG, E., FELKER, M., NG, H., AND JIN, C. 2007a. Red Opal: Product-feature scoring from reviews. *Proceedings of the 8th ACM Conference on Electronic Commerce*, 182-191.

[22] SCAFFIDI, C., BOGART, C., BURNETT, M., CYPHER, A., MYERS, B., AND SHAW, M. 2010. Using traits of web macro scripts to predict reuse. *Journal of Visual Languages and Computing*. *21*, 277-291.

[23] SCAFFIDI, C., AND SHAW, M. 2007b. Developing confidence in software through credentials and low-ceremony evidence. *International Workshop on Living with Uncertainties at the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)*.

[24] SCAFFIDI, C., AND SHAW, M. 2009. Inferring reusability of end-user programmers' code from low-ceremony evidence. *End User Programming for the Web Workshop, at the Conference on Human Factors in Computing Systems (CHI 2009)*.

[25] SCAFFIDI, C., AND SHAW, M. 2007c. Toward a calculus of confidence. *First International Workshop on the Economics of Software and Computation, at the 29th International Conference on Software Engineering (ICSE 2007)*.

[26] SEN, K., MARINOV, D., AND AGHA, G. 2005. CUTE: A concolic unit testing engine for C. *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 263-272.

[27] SOOD, M. 1998. Examining JDBC Drivers. *Dr. Dobbs Journal*. Jan 1998, http://www.drdobbs.com/java/184410469

[28] SZABO, G., AND HUBERMAN, B. 2010. Predicting the popularity of online content. *Communications of ACM*. *53*, 8, 80-88.

[29] U.S. GREEN BUILDING COUNCIL. 2010 (updated). *LEED 2009 for New Construction and Major Renovations*. http://www.usgbc.org/ShowFile.aspx?DocumentID=7283, (retrieved on July 20th, 2010).

[30] VISSER, W., HAVELUND, K., BRAT, G., PARK, S., AND LERDA, F. 2003. Model checking programs. *Automated Software Engineering*. *10*, 2, 203-232.

[31] WEYUKER, E., AND JENG, B. 1991. Analyzing partition testing strategies. *IEEE Transactions on Software Engineering*. *17*, 7, 703-711.
[32] THE MOZSCAPE API, http://www.seomoz.org/api , (retrieved on July 6th, 2012).
[33] THE OPENNLP PROJECT, http://opennlp.sourceforge.net, (retrieved on September 16th, 2011)
[34] MARCUS, E., MARCINKIEWICZ, M. AND SANTORINI, B. 1993. Building a large annotated corpus of English: The Penn Treebank. *Comput. Linguist.* 19, 2 (June 1993), 313-330.
[35] BACCIANELLA, S. (2010). Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. *Language Resources and …*, *0*, 2200–2204.
[36] HAILPERN, B., SANTHANAM, P. 2002. Software debugging, testing, and verification. *IBM Systems Journal*, *41*(1).
[37] TRETMANS, J., BELINFANTE, A. 1999. Automatic testing with formal methods. *European Int. Conference on Software Testing.*
[38] TAVAKOL, M., DENNICK, R. 2011. Making sense of Cronbach's alpha. I*nternational Journal of Medical Education*
[39] CRONBACH, L. 1951. Coefficient alpha and the internal structure of tests. P*sychometrika*, *16*(3).
[40] PARNAS, D. 1994. Software Aging. *Proceedings of the 16th International Conference on Software Engineering*
[41] DECOSTER, J. (1998). Overview of Factor Analysis, http://www.stat-help.com/factor.pdf (retrieved on August 15th, 2012).
[42] IBM STATISTICS BASE – FACTOR ANALYSIS, http://pic.dhe.ibm.com/infocenter/spssstat/v21r0m0/index.jsp?topic=%2Fcom.ibm.spss.statistics.help%2Fidh_fact.htm (retrieved on August 15th, 2012).
[43] TEXT ON A PATH FOR SILVERLIGHT, http://www.codeproject.com/Articles/30478/Text-on-A-Path-for-Silverlight, (retrieved on August 29th, 2012).
[44] BREWKA, G., NIEMALA, I., AND TRUSZCZYNSKI, M. 2008. Nonmonotonic reasoning. Foundations of Artificial Intelligence. 3, 239-284

Appendix

1. Recruitment message sent via the CodeProject newsletter

"CodeProject is currently working with a professor at Oregon State University to better understand what makes components reusable. In this paid study, we will ask you to identify up to 5 components on the CodeProject.com website that you have tried to use, and to describe whether it was easy or hard to use each. You would be able to do the study via an online interactive chat or a telephone, at your choice. As compensation, you would receive $10 for each component that you tell about (via PayPal). If you've got a few minutes to spare, then you can read more about the study at http://experiments.eecs.oregonstate.edu/general/components/"

2. Recruitment message sent via CodeProject's Twitter account

"Please take a moment to participate in this research study of component reusability. (And please RT) http://experiments.eecs.oregonstate.edu/general/components/"
URL: https://twitter.com/thecodeproject/statuses/203499826879528960

3. Questions asked to each participant in the user study

i) "What's the URL of component that you tried to reuse?"
ii) "What problems, if any, did you encounter while trying to reuse this component?"
iii) "When you downloaded the component, what led you to believe that it was reusable?"
iv) "On a scale of 1 to 5, where 1 is very hard to reuse and 5 is very easy to reuse, how would you rate the reusability of this component?"