

Characterizing Reusability of End-User Web Macro Scripts

Chris Scaffidi¹, Chris Bogart², Margaret Burnett², Allen Cypher³, Brad Myers¹, Mary Shaw¹

¹Carnegie Mellon University
Pittsburgh, PA 15213

²Oregon State University
Corvallis, OR 97331

³IBM Research
Almaden, CA 95120

{cscaffid, bam, mary.shaw}
@cs.cmu.edu

{bogart, burnett}
@eecs.oregonstate.edu

acypher
@us.ibm.com

ABSTRACT

Recommendations by software repositories depend on explicit or implicit models for evaluating the quality and relevance of components for programming tasks. As a step toward creating such a model for evaluating end-user web macro scripts, we have identified script characteristics that correspond to the likelihood of script reuse. For example, the likelihood of reuse increases with the number of variables and comments in the script, the number of online forum postings by the script's author, and the presence of popular keywords in the script's source code. We discuss possible applications of our results for new recommendation features.

Categories and Subject Descriptors

D.2.13 [Software Engineering]: Reusable software—*reuse models*

General Terms

Design, Human Factors, Reliability

Keywords

End-user software engineering, end-user programming, reuse

1. INTRODUCTION

Software repositories provide recommendations in several ways to programmers. All recommend components in response to search queries [6]. Some repositories watch programmers work and then proactively suggest high-quality, relevant components for reuse [15]. Conversely, other repositories identify *low*-quality, defective components that need fixes [10].

While such systems successfully offer recommendations for professional programmers, they often depend on information that is unavailable in the context of end-user programmers—those millions of information workers, teachers, and others who do not consider programming to be their main job, but who do it nonetheless in order to get their job done [18]. For example, some recommendations are derived from call graphs, inheritance hierarchies, method signatures, and similar information based on types [8][14][16]. This is appropriate in object-oriented programming but less so for end users' spreadsheets and scripts, which do not call each other, inherit from each other, or contain statically typed variables [4][12][22]. Other recommendations come from version

information about classes that are often modified simultaneously [10][23], but with no calls between scripts, there is little reason why they should be modified together. Some recommendations depend on components' functional specifications [9], documentation [6], or other carefully provided meta-information [13], but end-user programming processes are very informal and do not include investing time in specification, documentation, or other rich annotations [3][22].

Furthermore, while some recommendations are based on uses of components by programmers [15][21], we are particularly interested in early evaluation of component reusability *prior* to any actual reuse by other programmers. The reason is that early evaluation would facilitate repository cultivation (not just filtering and ranking of search results). For example, if a programmer is creating code that is unlikely to be reusable, the repository might interactively recommend changes to improve reusability. Conversely, another potential application is to flag new components that have high likelihood of reuse, so an administrator can review them and add some to a "New and Interesting" recommendation list. Our focus on early evaluation of reusability further restricts the information that is available for making recommendations.

Thus, supporting applications like these requires us to find information that (1) is available in the end-user programming context, (2) is available when programs are initially created, and (3) actually corresponds to reusability. In this paper, we identify information meeting these criteria by examining the information requirements of existing recommendation systems, evaluating if and how we can extract analogous information in the context of a particular end-user scripting platform [12], and using web server logs to test if the information empirically corresponds to script reuse.

Our results show that several end-user script characteristics available at script creation-time do indeed correspond to reusability. Examples include the presence of comments and the number of script parameters. Surprisingly, some characteristics do not correspond to reuse as expected. For instance, although research shows that defect count generally rises with code size [10][17], we find that long scripts are actually more likely than short scripts to be reused by their respective authors. Identifying script characteristics is a first step toward developing a predictive model of reuse that will support new recommendation features.

2. CONTEXT FOR STUDY: COSCRIPTER

As a concrete case for study, we consider the CoScripter web macro tool (formerly called Koala), which records end-user programmers' actions in the Firefox browser as re-playable "macro scripts" [12]. For example, a script might instruct the browser to visit www.southwest.com and submit a form to look up a flight's status. The author can edit the script, possibly giving it a title, modifying recorded instructions, or adding comments. The author

This paper is not to be included in the ACM Digital Library. The authors retain the copyright. The preferred citation for this non-archival paper is:

C. Scaffidi, C. Bogart, M. Burnett, A. Cypher, B. Myers, and M. Shaw. *Characterizing Reusability of End-User Web Macro Scripts*, Presentation at the Intl. Workshop on Recommendation Sys. for Software Engineering, co-located with FSE 2008, 10 Nov 2008, unpublished.

can change literal strings to variable references (which are read at runtime from a per-user configuration file called the Personal Database) and can insert “mixed-initiative” instructions that cause CoScripter to pause at runtime while the user makes decisions and performs a few actions manually. CoScripter stores all scripts on a wiki so other users can run, edit, or copy-and-customize them. By default, each script is publicly visible and modifiable, but its author can mark it “private” so that it is not visible to others.

Interviews of IBM employees who use CoScripter “show that CoScripter has addressed many user automation and sharing needs” by capturing procedural knowledge in a reuse-conducive form [11]. Users’ enthusiasm is particularly remarkable given that CoScripter does not support basic computational primitives provided by most languages, such as conditionals, loops, or structured data. Personal Database variables are untyped and read-only (scripts cannot update variable values). Scripts can call one another, but they cannot pass parameters. Moreover, very few users have discovered how to implement a call.

3. SCRIPT CHARACTERISTICS

CoScripter represents an extreme form of the situation outlined in the introduction: scripts lack inter-script calls, inheritance, types, simultaneous script modification in version histories, functional specifications, documentation, and detailed annotations. Given these differences from traditional languages, we searched the literature for kinds of information that have proven useful for identifying reusable components created by professional programmers, and we considered how analogous information might be reflected in characteristics of CoScripter scripts (summarized in Table 1).

Findability: To be reused, a component must first be found. At a minimum, most repositories for professional programmers support keyword-based search, which recommends components based on the text of their source code (eg: [6]) or category labels and other meta-data specified by the component author (eg: [13]). Keyword-based queries can be generated automatically by examining the source code of a component that a programmer already knows about [6], yielding a “show me similar classes” recommendation.

To test whether the findability of a script corresponds to reuse, we evaluate several script characteristics related to how prominently the script appears in the repository user interface: whether the script was created for publication on the tutorial list, whether it appears early in search results when sorted by author name, and whether the domains in its URLs and the tokens in its content reflect the general interest of the CoScripter community.

Expertise of author: To be reused, a component must not only be relevant to a reuse context but must also be reliable enough for use in that context. Several repositories for professionals provide indicators of component quality. One indicator is the number of previous downloads and various schemes that allow users to rate components [1]. However, predicting a component’s reuse based on its use suffers from the chicken-and-egg “cold-start” problem at code creation-time: some programmers must try to reuse the component before the system can predict reuse by other people.

Some systems address this limitation by tracking the reputation of component creators, rather than just components themselves [20]. We consider several characteristics that might relate to script authors’ expertise. Early adopters might be savvier than later adopters and more likely to produce reusable scripts, so the

chronologically-assigned author and script ids might relate to reusability. Higher reusability might also be associated with active forum participants, users from IBM (who might have even been on the team of professional programmers who created CoScripter), script authors who use advanced scripting keywords, and those with a history of creating reusable scripts.

Defect prediction: To be reliable, a program should contain few bugs. Consequently, the numerous defect-prediction models might offer insight into the reusability of professionals’ components. These machine learning models combine version histories and code-based measures to predict the likelihood of defects [10][17]. In addition, researchers have specifically proposed coupling and cohesion measures for evaluating component *reusability* [5][8].

Version histories are not available at script creation time, and the absence of inter-script calls prevents using coupling and cohesion measures for CoScripter scripts. However, we can test measures of code length as possible indicators of reusability. In addition, under the expectation that experimental scripts are more likely to contain defects, we evaluate if scripts with “test” or “Copy of” in the title—or even no title at all—are less likely to be reused.

Understandability: Prior repository research suggests that findability and reliability alone do not ensure reusability. Reuse also depends on professional programmers’ *perceptions* of the reusability of components [19]. At a basic level, evaluating a component requires understanding its functionality and preconditions for use. To assist in this, professional programmers provide comments, documentation, and readable variable names [7][6].

Designers of a spreadsheet repository for end-user programmers also noted the impact of understandability on reusability [22].

We consider script characteristics related to the readability of script titles, URLs, literals, and form field references, as well as the presence of comments and mixed-initiative instructions, which explain what a user must do to help the script proceed at runtime (and also offer a way for users to help scripts through difficult steps, potentially reducing brittleness).

Preconditions: Finally, prior analysis of the CoScripter repository suggested that implicit preconditions seemed to reduce reusability [4]. For example, less-reused scripts often required that the user had access to intranet URLs or was logged into a site prior to replaying scripts. Such preconditions could reduce the range of cases where users might be able and interested in reusing scripts.

We test this statistically by considering several kinds of preconditions, such as requiring that the browser is at a certain URL prior to execution, that the browser’s cookies are in a certain state, that multiple servers are currently online and accessible to the user, and that hardcoded literals in the script are suitable in the reuse context (rather than needing substitution through a variable).

4. EXPERIMENT

We extracted six months of web server logs from the CoScripter web server, which is primarily used by non-IBM employees (as IBM employees mainly use a small company-internal repository instead). In addition, we retrieved the initial source code from version control for each public script in that period (N=937). We performed statistical tests to determine whether the candidate script characteristics corresponded to significantly higher likelihood of script reuse during the three months after each script was created.

Table 1. Each row shows one script characteristic that we hypothesized would correspond statistically to four kinds of reuse. Characteristics are sorted in the order of their introduction in Section 3. A + (-) hypothesis indicates that we expected higher (lower) levels of reuse to correspond to the characteristic (e.g.: we hypothesized that if a script is created for publication on the tutorials list, then it is more likely to be reused).

Non-empty empirical results indicate statistically significant differences in reuse, with + (-) indicating that higher (lower) reuse corresponds to higher levels of the characteristic. One + or - indicates one-tail significance at $p < 0.05$, ++ or -- indicate $p < 0.01$, and +++ or --- indicate $p < 0.00036$ (which is the cutoff corresponding to a Bonferroni correction of $p < 0.05$). A characteristic name is bolded if its z-tests generally supported our hypothesis.

	Charac- teristic name	Meaning	Hypo- thesis	Empirical Result			
				Self Exec	Other Exec	Other Edit	Other Copy
Findability	tutorial	bool: true if script was created for tutorial list	+		+	+++	+++
	auloname	bool: true if script author's name starts with punctuation or 'A'	+		-		
	keydom	real: normalized measure of how many other scripts contain the same URLs (domains) as this script	+		+++	++	+
	keycnt	real: normalized measure of how many other scripts contain the same tokens as this script	+	+			
Expertise of creator	aid	int: id of the user who authored the script	-	+++	---	---	---
	sid	int: id of the script	-		---	---	---
	nforum	int: # of posts by the script author on the CoScripter forum	+	++		++	
	ibm	bool: true if script's author was at an IBM IP address	+	++		+++	+++
	ctlclick	bool: true if script contains "control-click" or "control-select" keywords	+		+		+++
	npcreated	int: # of scripts by same author that were created prior to this script	+	+++	---		
	npselfexec	int: # of scripts by same author that were executed by author prior to this script's creation	+	+++	---	--	
	npoexec	int: # of scripts by same author that were executed by other users prior to this script's creation	+	---	+++	-	-
	npoedit	int: # of scripts by same author that were edited by other users prior to this script's creation	+	---	+++	-	-
	npocopy	int: # of scripts by same author that were copied by other users prior to this script's creation	+	---	+++	-	-
Defect pre- diction	tloc	int: total # of lines (sloc + cloc)	-	+			
	sloc	int: total # of non-comment lines in script	-	++			
	ndloc	int: total # of distinct non-comment lines in script	-	+++			++
	titlts	bool: true if script title contains the word "test"	-	--			
	titlcp	bool: true if script title contains the phrase "Copy of"	-	--	-		
Understandability	titled	bool: true if script has a title	+	+++	++		
	titlpn	bool: true if script title contains punctuation other than periods	-	-			
	nonroman	pct: % of non-whitespace chars in title or content that are not roman	-		-	-	
	nmchost	int: # of URLs in script that use numeric IP addresses	-				
	enus	int: # of US URLs in script	+	+	+		
	nonenus	int: # of non-English words in literals + # of URLs outside USA	-	-		-	--
	unklang	bool: true if nonenus == enus == 0	-		---		
	ctnordin	bool: true if script uses ordinals (eg: "second") to reference form fields	-	+++			
	cloc	int: # of comment lines	+	+	++	+++	+++
	nyloc	int: # of mixed-initiative "you manually do this" instructions	+		+	+	++
Preconditions	assmurl	bool: true if first line of script is not a "go to URL" instruction	-		---		
	ctnclstt	bool: true if script contains "log in", "logged in", "login", or "cookie"	-				
	nhosts	int: # of distinct hostnames in script's URLs	-		+++		
	niinet	int: # of hosts referenced by script that seem to be on intranets	-		-		
	npar	int: # of parameters (configuration variables) read by script	+	++	+	+++	+++
	nlit	int: # of literal strings hardcoded into script	-	+++			

4.1 Measures of reuse

We consider four forms of reuse:

Self-exec: Did the script author ever execute the script between 1 day and 3 months after creating the script? (We omit executions within 1 day, since such executions might be associated with the initial creation and testing of the script, rather than reuse per se.) {17% of scripts meet this criterion}

Other-exec: Did any other user execute the script within 3 months of its creation? {49%}

Other-edit: Did any other user edit the script within 3 months of its creation? {5%}

Other-copy: Did any other user copy the script to create a new script within 3 months of the original script's creation? {4%}

We chose binary measures of reuse rather than absolute numbers of reuse events for two reasons. First, unless a repository user chooses to sort scripts by author, the user interface sorts search results by the number of times that a script is run. This appears to result in a “pile-on” effect: oft-run scripts tend to be reused very much more in succeeding weeks. While this helps confirm that *findability* corresponds to reuse, it interferes with using absolute reuse counts as a measure of reuse for testing script characteristics. Second, scripts can recursively call themselves (albeit without parameters), and some users also apparently have set up non-CoScripter programs that run CoScripter scripts periodically (e.g.: once per day), which clouds the meaning of absolute counts.

Although none of the scripts in our time interval were recursive, some were run periodically. When computing our binary reuse measures, we considered a script to have been reused if it was called from a periodic program. However, we carefully examined the logs to identify automated spiders that walked through the site and executed (or even copied) very many scripts in a short period of time, so that we could filter out those events. Most of these spiders were running from IBM IP addresses, apparently owing to automated analyses run by colleagues at IBM.

The 3 month post-creation observation interval that we selected was a compromise. Choosing a short interval runs the risk of incorrectly ruling a script as not-reused, if it was only reused beyond the interval. Selecting a longer interval reduces the number of scripts whose observation interval fell within the 6 months of available logs. Selecting half of the log period as the observation interval resulted in few cases (20) of erroneously ruling a script as not-reused, yet it provided over 900 scripts for study.

4.2 Testing correspondence

For each script characteristic, we divided scripts into two groups. For Boolean characteristics, we grouped scripts based on truth value. For numeric characteristics, we grouped based on whether each script had an above-average or below-average level of the characteristic.

For each group's scripts, we then computed the four reuse measures. Finally, for each combination of characteristic and reuse measure, we report the results of a one-tailed z-test of proportions ($p < 0.05$). In cases where the correspondence between script characteristic and reuse measure was actually *opposite* what we expected, we also report whether a one-tailed z-test would have been significant in the opposite direction.

4.3 Results

As shown in Table 1, 21 of the 35 script characteristics generally corresponded to reuse as we hypothesized—that is, for each of these 21 characteristics, more tests were significant in the direction that we anticipated than in the opposite direction. Many more tests were significant in the anticipated direction than would have been expected by chance (7 of 140 at $p < 0.05$).

In terms of robustness, we noted that many candidate script characteristics are count integers that could be normalized by the overall script length. Consequently, we performed our experiment twice, once with the characteristics shown in Table 1, and again with length-normalized characteristics as appropriate. In virtually every case, results were identical.

Finding: Two of four findability characteristics corresponded to higher reuse by other users but not to reuse by the author (perhaps because script authors could uniformly find their own scripts through a particular screen provided by the user interface).

Expertise of author: Half of the expertise-related characteristics corresponded to more than one form of reuse. However, in the other half, we found that while script authors with high levels of prior experience tended to produce scripts that were executed by other users, those scripts were *less* likely to be *edited or copied* by other users. Perhaps such scripts generally worked well and were reusable without needing edits or copy-paste customization.

Defect prediction: Tests showed that if a user took the time to title a script with something other than “test” or “Copy of...”, then that script was more likely to be reused by the author, consistent with our belief that some un-reused scripts are experimental. However, no characteristic based on script length consistently corresponded to reuse by other users. Moreover, it appears that longer scripts (which we suspected would be more defect-prone than shorter scripts) were actually *more* likely to be reused by their respective *authors*, perhaps for reasons discussed below.

Understanding: The most successful set of characteristics dealt with the readability of scripts. Comments were particularly important, as were “mixed initiative” instructions.

Preconditions: The use of Personal Database variables corresponded to all four forms of reuse. Our tests statistically confirmed the prior work's observation that execution by other users was lower when scripts accessed intranet URLs or presumed that the browser was at a certain URL. Interestingly, editing and copying by other users was not associated with these characteristics.

5. DISCUSSION

Our experiment confirmed that it is possible to find plenty of end-user script characteristics that are available at script creation time and that correspond to likelihood of reuse.

Perhaps the most surprising result was that longer scripts were more reused by their authors than shorter scripts. This stands in stark contrast to prior work where researchers found that the likelihood of defects increases with code length and complexity [10][17]. One possible explanation for our results is that longer scripts save more manual labor, in which case our results are consistent with the Attention Investment Model, which states that programmers are more likely to construct an abstraction if they are confident that it will save effort on net in the future [2]. But perhaps for other users, the added costs of understanding, defect checking, and adaptation cancel out this benefit.

This highlights the fact that recommendation systems should take account of the usefulness of components, not just the probability of defects. Many of our script characteristics could be interpreted as possible indicators of script usefulness. In addition to measures of script length and the number of literals (*nlit*), examples include whether the script contains text that matches the interests of other repository users (*keydom* and *keycnt*) and whether the script was created for the tutorial list (*tutorial*). Usefulness might also be inhibited if the script assumes preconditions, and perceived usefulness might be inhibited if users cannot understand the script.

In many cases, each script characteristic corresponded to different measures of reuse. For example, prior experiences of a script author corresponded to executions by other people but not to edits by other people. This highlights the fact that different kinds of reuse occur for different reasons and therefore might correspond to different script characteristics.

6. FUTURE WORK

The information embodied in our script characteristics has proven useful in the past for designing successful recommendation systems, so our results suggest that there is a range of end-user programming contexts where this information could be useful for developing new quality models and recommendation features.

Specifically, this paper constitutes first steps toward a model that would combine script characteristics in order to predict reusability. Such a model would probably generalize to other web macro platforms, since few of the script characteristics are particular to CoScripter (specifically, *nforum*, *ibm*, *click*, and *nyloc*). Further evaluations might show that some aspects of the model generalize to other end-user programming platforms such as spreadsheets, since few of the characteristics are particular to web macros. (Exceptions are characteristics relating to browsers, such as *keydom*, *assmurl*, and *niinet*.) Ultimately, it would also be extremely desirable to integrate this model with existing models—such as those based on collaborative filtering and code versioning—that take advantage of valuable information about usage history.

We or other researchers might use the resulting model to design new repository features, such as the automated script critique function that we mentioned in the introduction. We would gratefully welcome the opportunity to meet workshop participants with similar interests to discuss other possible directions for our work.

7. ACKNOWLEDGMENTS

This work was supported by the EUSES Consortium via NSF ITR-0325273, by NSF grants CCF-0438929 and CCF-0613823, and by an IBM International Faculty Award. Opinions, findings, and recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

8. REFERENCES

- [1] C. Bellettini, E. Damiani, M. Fugini. User Opinions and Rewards in a Reuse-Based Development System. *Proc. 1999 Symp. Softw. Reusability*, 1999, 151-158.
- [2] A. Blackwell. First Steps in Programming: A Rationale for Attention Investment Models. *Proc. 2002 Symp. on Human Centric Computing Lang. and Environments*, 2002, 2-10.
- [3] J. Brandt, P. Guo, J. Lewenstein, S. Klemmer. Opportunistic Programming: How Rapid Ideation and Prototyping Occur in Practice. *Proc. 4th Workshop on End-User Softw.Eng.*, 2008, 1-5
- [4] C. Bogart, M. Burnett, A. Cypher, Chris Scaffidi. End-User Programming in the Wild: A Field Study of CoScripter Scripts. *Proc. 2008 Symp. Visual Lang. and Human-Centric Computing*, 2008, to appear.
- [5] G. Caldiera, V. Basili. Identifying and Qualifying Reusable Software Components. *Computer* (24), No. 2, 1991, 61-70.
- [6] D. Čubranić, G. Murphy. Hipikat: Recommending Pertinent Software Development Artifacts. *Proc. 25th Intl. Conf. Softw. Eng.*, 2003, 408-418.
- [7] G. Fischer, S. Henninger, D. Redmiles. Cognitive Tools for Locating and Comprehending Software Objects for Reuse. *Proc. 13th Intl. Conf. Softw. Eng.*, 1991, 318-328.
- [8] G. Gui, P. Scott. Coupling and Cohesion Measures for Evaluation of Component Reusability. *Proc. 2006 Intl. Workshop on Mining Software Repositories*, 2006, 18-21.
- [9] R. Hall. Generalized Behavior-Based Retrieval. *Proc. 15th Intl. Conf. Softw. Eng.*, 1993, 371-380.
- [10] M. Lanza, M. Godfrey, S. Kim. MSR 2008 – 5th Working Conference on Mining Software Repositories. *Companion Proc. 30th Intl. Conf. Softw. Eng.*, 2008, 1037-1038.
- [11] G. Leshed, M. Haber, T. Matthews, T. Lau. CoScripter: Automating & Sharing How-To Knowledge in the Enterprise. *Proc. 26th SIGCHI Conf. on Human Factors in Computing Sys.*, 2008, 1719-1728.
- [12] G. Little, T. Lau, A. Cypher, J. Lin, E. Haber, E. Kandogan. Koala: Capture, Share, Automate, Personalize Business Processes on the Web. *Proc. 25th SIGCHI Conf. Human Factors in Computing Sys.*, 2007, 943-946.
- [13] V.F. Lucena Jr. Facet-Based Classification Scheme for Industrial Automation Software Components. *6th Intl. Workshop on Component-Oriented Programming*, 2001.
- [14] D. Mandelin, et. al. Jungloid Mining: Helping To Navigate the API Jungle. *Proc. 2005 SIGPLAN Conf. Programming Lang. Design and Implementation*, 2005, 48-61.
- [15] F. McCarey, M. Cinneide. Rascal: A Recommender Agent for Agile Reuse. *Artif. Intell. Rev.* (24), No. 3-4, 2005, 253-276.
- [16] A. Michail. Data Mining Library Reuse Patterns Using Generalized Association Rules. *Proc. 22nd Intl. Conf. Softw. Eng.*, 2000, 167-176.
- [17] R. Moser, W. Pedrycz, G. Succi. A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction. *Proc. 30th Intl. Conf. Softw. Eng.*, 2008, 181-190.
- [18] C. Scaffidi, M. Shaw, B. Myers. Estimating the Numbers of End Users and End User Programmers. *Proc. 2005 Symp. Visual Lang. and Human-Centric Computing*, 2005, 207-214.
- [19] S. Schach, X. Yang. Metrics for Targeting Candidates for Reuse: An Experimental Approach. *Proc. 1995 Symp. Applied Computing*, 1995, 379-383.
- [20] G. Suryanarayana, M. Diallo, J. Erenkrantz, R. Taylor. Architectural Support for Trust Models in Decentralized Applications. *Proc. 28th Intl. Conf. Softw. Eng.*, 2006, 52-61.
- [21] T. Tsunoda, T. Kakimoto, N. Ohsugi, A. Monden, K. Matsumoto. Javawock: A Java Class Recommender System Based on Collaborative Filtering. *Proc. 17th Intl. Conf. Softw. Eng. and Knowledge Eng.*, 2005, 491-497.
- [22] R. Walpole, M. Burnett. Supporting Reuse of Evolving Visual Code. *Proc. 1997 Symp. Visual Lang.*, 1997, 68-75.
- [23] T. Zimmermann, P. Weißgerber, S. Diehl, A. Zeller. Mining Version Histories to Guide Software Changes. *Trans. Softw. Eng.* (31), No. 6, 2005, 429-445.