Inferring Reusability of End-User Programmers' Code from Low-Ceremony Evidence

Chris Scaffidi, Mary Shaw
Institute for Software Research
Carnegie Mellon University
Pittsburgh, PA 15213
{cscaffid, mary.shaw}@cs.cmu.edu

ABSTRACT
While end-user programmers sometimes combine, learn from, or otherwise reuse existing code to quickly create new programs, not all code is equally reusable. Some code is reused by its creator or by others, but other code simply languishes on servers and never provides any help in the creation of subsequent programs. In this paper, we draw on numerous empirical studies of end-user and professional programmers to show that the reusability of code can be inferred on the basis of "low-ceremony" evidence. This evidence is information that is often informal, possibly unreliable, but that can be quickly gathered, interpreted and synthesized without the investment of substantial effort or skill by code producers or consumers. In the studies considered here, it includes information about code's mass appeal, flexibility, understandability, functional size, authorship, and prior reuses. We summarize a simple machine learning model that has successfully predicted reuse of web macros based on this low-ceremony evidence.

INTRODUCTION
End-user programmers have many opportunities to reuse code in programming tasks. For example, they might download a script and use it to validate web form inputs, or they might download and run a web macro to scrape data from web sites into a spreadsheet.

Yet for every reusable piece of end-user programmer code, there are typically many pieces of code that are never reused by anyone [6, 14, 16]. Actually identifying the reusable pieces of code within the mass of other code can be like looking for a needle in a haystack.

Conventional software engineering provides methods for assessing or ensuring the quality of code. These methods include formal verification, code generation by a trusted automatic generator, systematic testing, and empirical follow-up evaluation of how well the software works in practice. We have used the term "high-ceremony evidence" to describe the information produced by these methods [15], since applying them requires producers or consumers of code to exert high levels of skill and effort, in exchange for strong guarantees about code quality.

But end-user programmers (and some professional programmers) often lack the skill, time, and interest to apply these methods. What they need instead are methods based on "low-ceremony" evidence: information that may be informal, imprecise, and unreliable, but that can nevertheless be gathered, interpreted, and synthesized with a minimal amount of effort and skill in order to generate *confidence* (not a guarantee) that code is reusable.

Low-ceremony evidence would be particularly appropriate for end-user programmers because they rarely need the resulting program to perform perfectly. For example, in one study, teachers reported that their gradebook spreadsheets were "not life-and-death matters" [17], and in another study, web developers "did not see their efforts as 'high stakes' and held a correspondingly casual view of quality" [12]. For these people, the strong quality guarantees of high-ceremony methods are probably not worthwhile enough to justify the requisite effort. But low-ceremony evidence might suffice, if it is possible to make reasonably accurate assessments of code's reusability based on whatever low-ceremony evidence is available about that code at a particular moment in time.

In this paper, we empirically show that the reusability of end-user programmers' code can indeed be inferred from certain kinds of low-ceremony evidence. We proceed in two stages.

First, we draw on numerous empirical studies to develop a catalog of low-ceremony evidence that is known to relate to reuse of end-user programmers' code. We categorize the evidence based on its source: evidence based on the code itself, evidence based on the code's authorship, and evidence based on prior uses of the code. For example, code is more likely to be reusable if it contains variables rather than hard-coded values, if it was authored by somebody who has been assigned to create reusable code, and if it was previously used by other people who rated it highly. Our catalog of kinds of evidence can be extended in the future if additional studies empirically document new sources of low-ceremony evidence that are indicative of code reusability.

Second, we summarize a recent study that combined the first two categories of low-ceremony evidence to accurately predict whether web macro scripts would be reused [14]. Our results open several research opportunities aimed at further exploring the range and practical usefulness of low-ceremony evidence.

EMPIRICAL STUDIES OF END-USER PROGRAMMERS

As shown in Table 1, we draw on 11 empirical studies of end-user programmers:

- Retrospective analyses of a web macro repository [3, 14]
- Interviews of software kiosk designers [4]
- A report about running a Matlab code repository [6]
- Observations of college students in a classroom setting [7]
- An ethnography of spreadsheet programmers [10]
- Observations of children using programmable toys [11]
- Interviews [12] and a survey of web developers [18]
- Interviews of consultants and scientists [16]
- Interviews of K-12 teachers [17]

Where relevant, we supplement these with one simulation of end-user programmer behavior [2], as well as empirical work related to professional programmers [1, 5, 9, 13]. In the sections below, we underline citations of work related to professional programmers in order to make it clear when a statement is only supported by research on professionals.

Table 1. Many studies mention evidence that is based on the code itself. This evidence contains information about mass appeal, flexibility, understandability, and functional size. A few studies mention evidence based on authorship or prior uses.

Evidence based on	End-user												Professionals			
Information about	2	3	4	6	7	10	11	12	14	16	17	18	1	5	9	13
Code itself																
Mass appeal					x				x			x	x			x
Flexibility		x			x		x	x	x		x	x	x			
Understandability			x		x	x		x	x	x	x		x			x
Functional size	x								x				x	x	x	
Code's authorship						x			x							
Code's prior uses				x												

EVIDENCE BASED ON THE CODE ITSELF

It is widely believed that professional programmers' code is more reusable if it has certain traits [1]. In particular, the code must be *relevant* to the requirements of multiple programming tasks, it must be *flexible* enough to meet those varying requirements, it must be *understandable* to the people who would reuse it, and it must be *functionally large* enough to justify reuse rather than coding from scratch.

These traits also apparently contribute to the reusability of end-user programmers' code, since all 11 end-user programming studies produced findings of the form, "Code was hard to reuse unless it had X," where X was a piece of low-ceremony evidence related to one of these four traits. For example, unless code contained comments, teachers had difficulty understanding and reusing it [17]. In this example, the evidence is the presence of comments in the code, and the trait is understandability. Thus, the evidence was an indicator of a trait, and thus an indicator (but not a guarantee) of reusability.

Evidence about Mass Appeal / Functional Relevance

The presence of keywords or other tokens in a certain piece of code appeared to be evidence of whether the code was relevant to many peoples' needs. For instance, web macros that operated on web sites with certain tokens in the URL (such as "google") were more likely to be reused by people other than the macro author [14]. Perhaps one reason why keywords were so predictive of reuse is that repositories and programming environments usually provide a search interface where users can type keywords to locate code [7, 8]. Thus, the presence of certain keywords can be evidence of mass appeal, suggesting a higher potential for reuse.

But when programmers seek reusable code, they are looking for more than certain keywords. Keywords are just a signal of what the programmer is really looking for: code that provides functionality required in the context of the programmer's work [5, 7, 13]. Typically, only a small amount of code is functionally relevant to many contexts, so a simple functional categorization of code can be evidence of its reusability. For example, 78% of mashup programmers in one survey created mapping mashups [18]. All other kinds of mashups were created by far fewer people. Thus, just knowing that a mashup component was related to mapping (rather than photos, news, trivia, or the study's other categories) suggested mass appeal.

Evidence about Flexibility and Composability

Reusable code must not only perform a relevant function, but it must do it in a flexible way so that it can be applied in new usage contexts. Flexibility can be evidenced by use of variables rather than hardcoded values. In a study of children, parameter-tweaking served as an easy way to "change the appearance, behaviour, or effect of an element [component]", often in preparation for composition of components into new programs [11]. Web macro scripts were more likely to be reused if they contained variables [3, 14].

Flexibility can be limited when code has non-local effects that could affect the behavior of other code. Such effects reduce reusability because the programmer must carefully coordinate different pieces of code to work together [7, 17]. For example, web page scripts were less reusable if they happened to "mess up the whole page" [12], rather than simply affected one widget on the page. In general, non-local effects are evidenced by the presence of operations in the code that write to non-local data structures (such as the web page's document object model).

Finally, flexibility can be limited when the code has dependencies on other code or data sources. If that other code or data become unavailable, then the dependent code becomes unusable [18]. Dependencies are evidenced by external references. For example, users were generally unable to reuse web macros that contained operations which read data from intranet sites (i.e.: sites that cannot be accessed unless the user was located on a certain local network) [3].

Evidence about Understandability

Understanding code is an essential part of evaluating it, planning any modifications, and combining it with other code [7]. Moreover, understanding existing code can be valuable even if the programmer chooses not to directly incorporate it into a new project, since people often learn from existing code and use it as an example when writing code from scratch [12, 13]. This highlights the value of existing code not only for verbatim blackbox or near-verbatim whitebox reuse, but also for indirect conceptual reuse.

Many studies of end-user programmers have noted that understandability is greatly facilitated by the presence of comments, documentation, and other secondary notation. Scientists often struggled to reuse code unless it was carefully documented [16], teachers' "comprehension was also slow and tedious because of the lack of documentation" [17], office workers often had to ask for help in order to reuse spreadsheets that lacked adequate labeling and comments [10], and web macros were much more likely to be reused if they contained comments [14]. End-user programmers typically skipped putting comments into code unless they intended for it to be reused [4]. In short, the presence of comments and other notations can be strong evidence of understandability and, indirectly, of reusability.

Evidence about Functional Size

When asked about whether and why they reuse code, professional programmers made "explicit in their verbalisation the trade-off between design and reuse cost" [5], preferring to reuse code only if the effort of doing so was much lower than the effort of implementing similar functionality from scratch. In general, larger components give a larger "payoff" than smaller components, with the caveat that larger components can be more specialized and therefore have less mass appeal [1]. Empirically, components that are reused tend to be larger than components that are not reused [9].

Simulations suggest that end-user programmers probably evaluate costs in a similar manner when deciding whether or not to reuse existing code [2], though we are not aware of any surveys or interviews which show that end-user programmers evaluate these costs consciously. Nonetheless, there is empirical evidence that functional size does affect reuse of end-user programmers' code. Specifically, web macros that were reused tended to have more lines of code than web macros that were not reused [14].

EVIDENCE BASED ON THE CODE'S AUTHORSHIP

In some organizations, certain end-user programmers have been tasked with cultivating a repository of reusable spreadsheets [10]. Thus, the identity of a spreadsheet's author might be evidence about the spreadsheet's reusability.

Even when an author's identity is unknown, certain evidence about the author can be useful for inferring code's reusability. For example, CoScripter web macros were more likely to be reused if they were uploaded by authors located at internet addresses belonging to IBM (which developed the CoScripter platform) [14]. In addition, web macros were more likely to be reused if they were created by authors who previously created heavily-reused macros.

EVIDENCE BASED ON THE CODE'S PRIOR REUSES

Once someone has tried to reuse code, recording that person's experiences can capture information about the code's reusability. Repositories of end-user code typically record this information as reviews, recommendations, and ratings [6, 8]. In the Matlab repository, capturing and displaying these forms of reusability evidence has helped users to find high-quality reusable code [6].

COMBINING LOW-CEREMONY EVIDENCE

As a first step toward finding effective models for combining low-ceremony evidence into predictions of reusability, we have designed and evaluated a machine learning model that predicts reuse of CoScripter web macros [14].

Before applying this model, it must be trained using information about macros (low-ceremony evidence) and about whether those macros were ever reused. In particular, while evaluating this model, we used low-ceremony evidence based on the code itself and authorship. For example, we used the number of comments in each web macro.

The training produces a set of simple arithmetic constraints that we call "predictors". For example, one predictor might be a constraint on the number of comments ≥ 3, and another might be that the number of referenced intranet sites ≤ 1. Ideally, all such predictors would be true for every reused macro and false for every un-reused macro.

After the training process produces this set of predictors, a second algorithm uses this set to predict if some other macro will be reused. The algorithm counts the number of predictors matched by the macro and predicts that it will be reused if it matches at least a certain number of predictors.

The model predicted reuse quite accurately, even using just the low-ceremony evidence collected directly from code or from information about the code's author. Specifically, the model predicted with 70-80% recall (at 40% false positive rate) whether other end-user programmers would reuse a given macro. The most useful predictors related to mass appeal, functional size, flexibility, and authorship.

These results show that low-ceremony evidence can be combined in a simple manner to yield accurate predictions of web macro reuse. While there may be other equally-accurate methods of combining evidence, our model has the advantage of being relatively simple, which might make it possible to automatically generate explanations of why the model generated certain predictions. Moreover, the model is defined in such a way that it does not require that the programs under consideration must be web macros. Thus, we are optimistic that it will be possible to apply the model to other kinds of end-user code.

CONCLUSION AND FUTURE DIRECTIONS

In this paper, we have argued that the reusability of end-user programmers' code can be characterized on the basis of low-ceremony evidence. This conclusion is supported by the fact that diverse empirical studies have reported a qualitative or quantitative relationship between code reuse and certain information that is based on the code itself, the code's authorship, or the code's prior uses. This information can be collected, interpreted, and synthesized without substantial investment of skill or effort by producers or consumers of the code. Our conclusion is also supported by the success of our machine learning model in predicting reuse of web macros based on low-ceremony evidence. This conclusion suggests three directions for future work.

First, we have found relatively few studies showing that code reuse is related to the code's authorship or prior uses. This was somewhat surprising, since evidence about prior uses has been incorporated into many repositories in the form of rating, review, and reputation features. Thus, one direction for future work is to perform more studies aimed at empirically identifying situations where this and other low-ceremony evidence helps to guide end-user programmers to highly reusable code. Further empirical studies might also help to extend our catalog by identifying new sources of low-ceremony evidence, beyond the code itself, authorship, and prior uses.

Second, it will be desirable to empirically confirm the generalizability of our machine learning model. This will require amassing logs of code reuse in some domain other than web macros (such as spreadsheets), collecting low-ceremony evidence for that kind of code, and testing the model on the data. At present, except for the CoScripter system, we are unaware of any end-user programming repository with enough history and users to support such an experiment. Ideally, just as we have drawn on research from studies performed by many teams, the machine learning model would be confirmed on different kinds of code by different research teams.

Finally, our results create the opportunity to collect and exploit low-ceremony evidence in new system features aimed at supporting reuse. For example, code that matches many reuse predictors could be ranked higher in search results, potentially making it easier to discover reusable code. Having already shown that low-ceremony evidence is predictive of reusability, implementing features like these would show that it is possible to put these predictions to good practical use in a real system.

ACKNOWLEDGEMENTS

We thank the members of the EUSES Consortium for constructive discussions. This work was supported by the EUSES Consortium via NSF ITR-0325273, and by NSF grants CCF-0438929 and CCF-0613823. Opinions, findings, and recommendations are the authors' and not necessarily those of the sponsors.

REFERENCES

1. T. Biggerstaff and C. Richter. Reusability Framework, Assessment, and Directions. *IEEE Software Vol. 4*, No. 2, March 1987, 41-49.
2. A. Blackwell. First Steps in Programming: A Rationale for Attention Investment Models. *2002 IEEE Symp. Human Centric Computing Lang. and Env*., 2002, 2-10.
3. C. Bogart, M. Burnett, A. Cypher, and C. Scaffidi. End-User Programming in the Wild: A Field Study of CoScripter Scripts. *2008 IEEE Symp. Visual Lang. and Human-Centric Computing*, 2008, 39-46.
4. J. Brandt, P. Guo, J. Lewenstein, and S. Klemmer. Opportunistic Programming: How Rapid Ideation and Prototyping Occur in Practice. *Proc. 4th Intl. Workshop on End-User Software Engineering*, 2008, 1-5.
5. J. Burkhardt and F. Détienne. An Empirical Study of Software Reuse by Experts in Object-Oriented Design. *Proc. 5th Intl. Conf. Human-Computer Interaction*, 1995, 38-138.
6. N. Gulley, Improving the Quality of Contributed Software and the MATLAB File Exchange, *Proc. 2nd Workshop on End User Software Engineering (WEUSE II)*, 2006, 8-9.
7. A. Ko, B. Myers, and H. Aung. Six Learning Barriers in End-User Programming Systems. *2004 IEEE Symp. Visual Lang. and Human-Centric Computing*, 2004, 199-206.
8. G. Little, T. Lau, A. Cypher, J. Lin, E. Haber, and E. Kandogan. Koala: Capture, Share, Automate, Personalize Business Processes on the Web. *Proc. 25th SIGCHI Conf. Human Factors in Computing Sys.*, 2007, 943-946.
9. P. Mohagheghi, R. Conradi, O. Killi, and H. Schwarz. An Empirical Study of Software Reuse vs. Defect-Density and Stability. *Proc. 26th Intl. Conf. Software Engineering*, 2004, 282-291.
10. B. Nardi. *A Small Matter of Programming*, MIT Press, 1993.
11. M. Petre and A. Blackwell. Children as Unwitting End-User Programmers. *2007 IEEE Symp. Visual Lang. and Human-Centric Computing*, 2007, pp. 239-242.
12. M. Rosson, J. Ballin, and H. Nash. Everyday Programming: Challenges and Opportunities for Informal Web Development. *2004 IEEE Symp. Visual Lang. and Human-Centric Computing*, 2004, 123-130.
13. M. Rosson and J. Carroll. The Reuse of Uses in Smalltalk Programming. *Trans. Computer-Human Interaction (TOCHI), Vol. 3*, No. 3, 1996, 219-253.
14. C. Scaffidi, C. Bogart, M. Burnett, A. Cypher, B. Myers, and M. Shaw. Predicting Reuse of End-User Web Macro Scripts. Submitted to *31st Intl. Conf. Software Engineering (ICSE 2009)*.
15. C. Scaffidi, M. Shaw. Toward a Calculus of Confidence. *1st Intl. Workshop on Economics of Software and Computation*, 2007.
16. J. Segal. *Professional End User Developers and Software Development Knowledge.* Tech. Rpt. 2004/25, Dept. of Computing, Faculty of Mathematics and Computing, The Open University, Milton Keynes, United Kingdom, Oct 2004.
17. S. Wiedenbeck. Facilitators and Inhibitors of End-User Development by Teachers in a School Environment. *2005 IEEE Symp. Visual Lang. and Human-Centric Computing*, 2005, 215-222.
18. N. Zang, M. Beth Rosson, and V. Nasser. Mashups: Who? What? Why?. *Proc. 26th SIGCHI Conf. Human Factors in Computing Sys. - Work-in-Progress Posters*, 2008, 3171-3176.