Dimensions Characterizing Programming Feature Usage by Information Workers

Christopher Scaffidi, Amy J. Ko, Brad Myers, Mary Shaw
School of Computer Science, Carnegie Mellon University
{cscaffid, ajko, bam, mary.shaw}@cs.cmu.edu

Abstract

Information workers such as administrative staff, consultants, and their managers constitute one of the largest groups of end users, yet little research about their usage of programming features is available to guide development of end user programming tools. In this paper, we describe our survey of over 800 information workers and our analysis of their feature usage in applications such as spreadsheets, browsers, and databases. Our factor analysis reveals three clusters of features—macro features, linked structure features, and imperative features—such that information workers with an inclination to use a feature in each cluster also were inclined to use other features in that cluster, even though each cluster spans several tools. We discuss the implications for research aimed at providing end user programming tools for information workers.

1. Introduction

Many applications include features enabling customization and extension by end user programmers. Such features facilitate creating macros and dynamic charts in spreadsheets, scripts and forms in web pages, and stored procedures and tables in databases.

Researchers have performed surveys, ethnographies, and other studies of certain populations' use of programming features. These studies are essential to successful improvement on those features. For example, Rosson et al. ran a survey of over 300 web developers to document their work practices and to identify opportunities for better tool support [3]. Likewise, Rode et al. conducted an ethnography to characterize abstraction creation by people using domestic appliances, informing design of automation features [2], and Pane et al. performed studies to identify features needed in a programming system for children [1].

In this paper, we present a survey of programming feature usage by another group of end users: information workers, such as managerial and administrative staff, whose jobs involve managing data with spreadsheets, browsers, and other common applications. These workers are one of the largest end user groups [5], yet basic questions remain about their use of programming features: What kinds of features are most heavily used by this group? What feature usage patterns span across the applications that are used?

To address these questions, we surveyed over 800 Information Week magazine readers concerning their use of 23 programming features within five applications. We focused on features related to abstraction creation, since prior research showed that professional programmers rely on abstraction to help enhance the quality of their work (as discussed in detail in [4]).

Usage rates of features varied widely. For instance, only 16% reported use of server-side "include" files (which contain reusable HTML chunks like web sites' headers), whereas 89% reported use of simple spreadsheet functions (like "sum") that link cells.

To characterize feature usage patterns across applications, we applied factor analysis, which revealed three feature clusters. These clusters dealt with macros, textual imperative code, and linked structures (such as database tables linked by keys). For each cluster, users who were inclined to use one feature also were inclined to use that cluster's other features. For example, users who were inclined to use one imperative feature such as JavaScript functions were generally inclined to use other imperative features such as stored procedures. Overall, the features used by the most people were linked structure features.

2. Data collection method

We collaborated with *Information Week* to develop, test, and run a web-based survey containing 96 questions. To recruit respondents, *Information Week* published a link to the survey on their web site and emailed 125,000 randomly selected subscribers who had previously indicated their willingness to do surveys. The email mentioned our goals to learn about feature usage and to improve software flexibility, and it noted that we would enter respondents into a drawing for one prize of $500 and five of $100 each. Within two months, 831 people completed the survey (which *Information Week* reports is a typical response rate for their surveys).

In the first set of questions, we asked respondents whether they or their subordinates had used five classes of applications in the past three months: databases, web page editors, web server scripting environments, spreadsheets, and word processors / slide presentation editors. For each application used, we asked about usage of that application's features during those three months; for example, if databases were used, then we asked whether tables were created. In total, we asked about 23 features, as shown in Table 1. For each feature, respondents could answer "Yes," "No," or "Don't Know" (or skip the question).

Next, to assess respondents' knowledge about programming terms, we asked if they were familiar with certain terms: *variables, subroutines, conditionals, and loops*. For each term, we provided a definition and, if appropriate, a list of synonyms. If respondents were familiar with the term, then we asked if they had personally created the corresponding construct in the past year. For the "subroutines" term, we asked two usage questions to distinguish between creation of subroutines and usage of subroutines created by other people.

Finally, we asked about respondents' company and individual background. See [4] for the full text of the survey, in which we also asked about usage of the web, suggestions for software improvement, and several other issues not relevant to this paper.

3. Characteristics of the sample

Information Week advertises itself as "exclusively for information managers… involved in computer, communications and corporate planning," so unsurprisingly, 76% of respondents managed at least one subordinate. Only 10% worked for IT vendor firms; instead, respondents generally came from business services, education, government, manufacturing, and finance/insurance. Although 99% had attended college, only 22% majored in computer science or computer-centric fields; the others mostly majored in business, sciences, or engineering. Only 23% were IT or networking staff; the remainder were consultants, managers, or administrative staff.

Yet respondents generally displayed certain characteristics of programmers. As shown in Table 1, they used applications' programming features quite heavily. Moreover, 79% were familiar with all four programming terms (variables, subroutines, conditionals, and loops); in the past year, 35% actually created all four of the corresponding constructs in their own code. Thus, our sample's information workers could generally be characterized as skilled end user programmers.

4. Analysis method

To evaluate feature usage, we applied factor analysis, a technique that summarizes correlations among observed variables. A "factor loading" is a vector

Table 1. Most (but not all) features were used by many people; the top ten usage rates are bolded. In addition, factor loadings exceeding an arbitrary cutoff of 0.15 are also bolded.

Application	Application Usage (% of all users)	Feature Description	Feature Usage (% of all users)	Factor Loadings (see Section 5)		
				Macro	LinkStruct	Imperative
Slide Editors and Word Processors	96.1	Creating document templates	**73.3**	-0.59	-0.02	-0.17
		Making inter-document hyperlinks	**53.7**	-0.63	-0.04	0.07
		Recording desktop editor macros	33.3	**0.86**	0.03	0.07
		Creating/editing desktop editor macros	30.8	**0.87**	0.07	0.09
Spreadsheets	93.1	Using functions ("sum") linking cells	**89.3**	-0.21	**0.37**	-0.10
		Creating charts in spreadsheets	**80.1**	-0.40	**0.27**	0.14
		Creating inter-spreadsheet references	**66.2**	omitted	omitted	omitted
		Creating spreadsheet templates	50.4	-0.24	-0.19	0.12
		Recording spreadsheet macros	42.3	**0.60**	-0.35	-0.13
		Creating/editing spreadsheet macros	38.6	**0.68**	-0.18	0.02
Databases	79.3	Referencing records by key	**74.0**	-0.02	**0.68**	-0.06
		Creating tables	**71.7**	-0.01	**0.54**	-0.21
		Creating database views	**68.1**	omitted	omitted	omitted
		Creating stored procedures	49.5	0.04	-0.72	**0.22**
Web Pages	68.6	Creating hyperlinks	**65.2**	-0.05	0.06	-0.62
		Creating web forms	**52.2**	0.15	**0.18**	-0.50
		Creating web page behavior scripts	42.9	omitted	omitted	omitted
		Creating JavaScript functions	34.3	-0.04	-0.09	**0.50**
		Using JavaScript "new" function	23.3	-0.03	-0.07	**0.64**
Web Server Script Hosts	52.3	Creating web server script functions	40.9	omitted	omitted	omitted
		Referencing PHP/Perl libraries	27.9	-0.06	-0.41	-0.06
		Using PHP/Perl "new" function	25.4	0.07	0.01	**0.42**
		Using Web server-side static includes	16.3	-0.09	**0.25**	-0.26

whose cells each reflect the correlation of that factor with an observed variable; if a factor has large values in two cells, then the corresponding two data variables highly correlate with the factor (and with each other). For example, an ethnography might record the amount of time that programmers spend in sixty activities, ranging from typing code to talking on the phone; factor analysis might reveal a "collaborativeness" factor showing that people who often talk on the phone also tend to perform other collaboration activities, such as writing emails and attending meetings.

To prepare for factor analysis, our data cleaning involved the following stages.

Although most questions, including feature usage questions, were multiple-choice, some demographic items such as college major and industry required a coding system. In general, these were fairly unambiguous to code (and none of these variables were statistically significantly related to our results), so we did not implement inter-rater checks.

The 15 respondents with more than 1000 subordinates reported extremely high feature usage, so after plotting feature usage versus subordinate count, we opted to discard their data, which left 816 data records.

We flagged questions that were skipped and "Don't Know" answers as missing values. Since factor analysis models relationships among observed data, it cannot be applied directly to unobserved/missing values. Our survey only asked about features for applications that were actually used, so our factor analysis used only the 168 records from respondents who reported usage of all five applications and who did not skip any feature usage questions. To verify that the resulting factor structure generalized to all 816 respondents with 1000 or fewer subordinates, we used the factors' structure to construct a traditional scale for each factor, and we then checked each scale's Cronbach alpha (discussed below).

We were able to cross-check some answers. For example, we repeated a question about creating web forms (once in the context of web page editors, and once in the context of web server scripting). If the user skipped the question once but also answered once, then we used the available answer rather than flagging the question as a missing value; in contrast, if the user answered twice but in a contradicting way, then we flagged the item as missing. These cross-checks modified answers for less than a dozen users.

We recognized that not every instance of feature usage was equally "significant." For example, 66% of respondents reported creating hyperlinks, whereas only 16% reported creating static server-side includes. To standardize features' contributions to the factor analysis, we scaled each feature usage variable to a mean of 0.0 and standard deviation of 1.0, thereby ensuring that a "Yes" for a commonly used feature was coded as a value less than 1.0, while a "Yes" for an uncommonly used feature was coded as more than 1.0.

Moreover, feature usage co-occurs a great deal within each application: for example, creating JavaScript highly correlates with creating web forms. This "bundling" has nothing to do with abstraction, but rather with the application. To remove this effect, for each feature, we subtracted the average feature usage by that respondent for that application, thus revealing inclinations to use features rather than applications.

Finally, it is essential to avoid trying to extract more factors than the data support. Each factor loading is an eigenvector of the data's covariance matrix, and one commonly practiced threshold accepts only factors that have eigenvalues exceeding 1.0. In addition, variables lacking significant communality with the other variables should be omitted and the analysis redone, as including outliers may lead to spurious factors.

Following these guidelines, we performed an initial factor analysis with all 23 feature usage items. Three eigenvalues exceeded 1.0, so we reran the analysis with a constraint allowing only three factors. Checking items' communality revealed that four were very low. (Specifically, each had a total squared factor loading under 0.1, whereas that of the other 19 items exceeded 0.1 and averaged over 0.4.) So as flagged in Table 1 with the label "omitted," we eliminated the following items from further analysis: creating database views, inter-spreadsheet references, web server script functions, and web page behavior scripts. Finally, we ran our factor analysis with the remaining 19 items.

5. Results

The analysis yielded three factors that each correlated strongly with usage of different features. (Factor loadings appear in the right column of Table 1; in addition, loadings that exceeded an arbitrary cutoff of 0.15 appear in parentheses below.)

The first factor correlated most positively with recording (0.60) and textual editing (0.68) of spreadsheet macros as well as recording (0.86) and textual editing (0.87) of macros in other desktop applications. This "Macros" factor indicated that people with an inclination to use one macro feature also were inclined to use other macro features. In addition, this factor negatively correlated with creating document templates and hyperlinks in word processors and slideshow software.

The second factor correlated most positively with creating database tables (0.54) and linking them via keys (0.68). It correlated positively with creating web forms (0.18) and "static includes" shared by web pages (0.25). Finally, it correlated positively with creating spreadsheet charts (0.27) and linking spreadsheet cells using simple functions like “sum” (0.37). Because the factor correlated so strongly with this cluster of features related to creating and linking structures, we termed it the “LinkStruct” factor. It negatively correlated with creating database stored procedures.

The third factor, which we called the “Imperative” factor, correlated with using Perl and PHP’s “new” command in web server scripts (0.42), using JavaScript’s “new” command (0.64), creating JavaScript functions (0.50), and creating stored databases procedure functions (0.22). It correlated negatively with creating web forms and hyperlinks.

Template and hyperlink features fell outside all three clusters yet were heavily used. This may suggest the existence of undiscovered yet important clusters.

To verify these three factors’ robustness, we reran the factor analysis using various subsets of the data, such as only including respondents with no subordinates or using alternate factor extraction techniques. In each case, the same qualitative structure appeared.

Finally, because the factor analysis could only use the 168 respondents with no missing feature usage answers, we verified that the factors’ qualitative structure generalized to our entire sample by using that structure to construct a traditional scale for each of the factors. We then checked each scale’s Cronbach alpha. For example, the first factor positively correlated with four items and negatively correlated with two items, so our first scale equaled the sum of these four items minus the other two. The Cronbach alphas for these scales were 0.82, 0.62, and 0.64, respectively, indicating that the patterns revealed by the factor analysis applied fairly consistently throughout our entire data set.

6. Discussion and implications

As we design programming tools, we want their features to be useful to many people, but if users differ widely from one another in their feature usage, then we will be hard pressed to create broadly useful features. Fortunately, our survey’s primary implication is that skilled information workers do not differ so much in their feature usage as to discourage creating new programming features intended for large numbers of users.

Moreover, our results offer guidance on what new programming features might be used by the most people. Specifically, the most popular features generally are “linked structure” features (particularly those that represent data in 2D grids), highlighting the value of providing such features to this population.

Almost all respondents were familiar with imperative programming terms such as “loop,” so we suspect that the low usage of imperative features generally resulted from lack of desire, need, or ability to apply these features, rather than from unfamiliarity with the features. This suggests that researchers providing imperative programming features should carefully evaluate those features’ *utility* and not just their *visibility*.

We close by raising the central question prompted by our results: Why do these clusters exist? One possible hypothesis is that perhaps common work processes require using certain features together; in that case, contextual inquiry of the users’ work may offer clues for how to make programming features more useful. Alternatively, once people learn one cluster’s features, perhaps they are less willing to learn features of other clusters, even when those other features are highly relevant to a work task; in that case, studying how to motivate programming may be more necessary than studying users’ work context. In short, determining why these clusters exist is essential to helping information workers benefit from end user programming.

7. Acknowledgements

We thank Rusty Weston and Lisa Smith at *Information Week* for their partnership, as well as James Herbsleb, Irina Shklovski, and Sara Kiesler at Carnegie Mellon University for design and analysis assistance. This work was funded in part by the EUSES Consortium via the National Science Foundation (ITR-0325273), by an NDSEG fellowship, by the National Science Foundation under Grant CCF-0438929, by the Sloan Software Industry Center at Carnegie Mellon, and by the High Dependability Computing Program from NASA Ames cooperative agreement NCC-2-1298. Opinions, findings, and conclusions or recommendations expressed in this material are the authors’ and do not necessarily reflect sponsors’ views.

8. References

[1] Pane, J., Myers, B., and Miller, L. Using HCI Techniques to Design a More Usable Programming System, In *Proceedings of IEEE Symposia on Human Centric Computing Languages and Environments*, 2002, pp. 198-206.

[2] Rode, J., Toye, E., and Blackwell, A. The Fuzzy Felt Ethnography - Understanding the Programming Patterns of Domestic Appliances. In *Proceedings of the International Conference on Appliance Design*, 2004, pp. 10-22.

[3] Rosson, M.B., Ballin, J., and Rode, J. Who, What, and How: A Survey of Informal and Professional Web Developers, In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing*, 2005, pp. 199-206.

[4] Scaffidi, C., Ko, A., Myers, B., and Shaw, M. Identifying Categories of End Users Based on the Abstractions That They Create, Technical Report CMU-ISRI-05-110, Carnegie Mellon University, 2005.

[5] Scaffidi, C., Shaw, M., and Myers, B. Estimating the Numbers of End Users and End User Programmers, In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing*, 2005, pp. 207-214.