



Switching Between Tools in Complex Applications

Will Schroeder

Principal Usability Specialist
The MathWorks
3 Apple Hill Dr.
Natick, MA
USA
Will.schroeder@mathworks.com

Abstract

Large software applications are made up of many specialized tools. In Microsoft Word the document editor is supported by tools to create and fix drawings and tables. Programming environments have custom views (difference editors) and analyses (performance reports) to help developers make robust code. Every application has tools to help users sift the documentation.

In usability, we usually test a tool at a time, yet complex work requires many tools, and this brings a new set of issues. How do I know when I should be using a different tool? What tool do I need when the one I am using is not working? How do I get to it? How quickly can I start using it?

In complex or creative work, our observations show that users seldom choose the correct tool as soon as work progress dictates. This erodes productivity and creativity and is a prime target for improved designs.

Usability practice needs a procedure to identify, record, count, and highlight tool switch events for study. This paper describes one that supports the trained observers on which User-Centered Design relies to detect problems and causes, and evaluate design changes.

Keywords

Activity theory, situated actions, logging, workflow, user-centered design, context switching

Introduction

User-Centered Design for tool switching is a big topic, and we're just beginning to work out a technique for gathering, examining, and evaluating data. A study of users moving from tool to tool (and the design changes intended to improve it) will not fit easily into a usability practitioner's normal practice. This work generates large amounts of data that the practitioner has little time to analyze or reduce. Tool switching appears in time patterns that the facilitator is too close to the process to observe. Critical tool changes (those that merit immediate analysis) can't always be identified at the moment they happen, and when they are identified, retrospective analysis (of a recording) is necessary to evaluate them. To improve design in this area, new techniques based on an understanding of the problems tool switching gives users are needed.

Conventional usability practice is organized around tasks, because a task framework supports observation, testing, and measurement. Work is sectioned into testable units that are concrete steps with quantifiable goals. This tacitly assumes that chaining the right steps in the proper order adequately models how work is actually done. In fact, only simple and repetitive work can be reduced to such a step-by-step model. The production line model is not adequate for complex or creative work. These are not production tasks. When starting them, users have only the most general idea what the workflow will be or which tools they will bring to bear on the work.

Individual tools in Visual Studio, Photoshop, and Word (toolkits) may work well on their own, but not in concert or in sequence. The following inefficiencies arise when users move between tools:

- Time and effort involved in transition (switching) does not advance the task.
- Moving work from tool to tool dissipates focus.
- Information is difficult to transfer when focus changes.
- The right tools aren't used if they can't be smoothly integrated into users' work.

Tool switching is the task of choosing and changing from one tool to another. Tool switching work is not productive—it does not alter the product. Users switch to increase productivity by applying whatever is, at the time, the most productive tool. But gains from using the new tool must offset losses from the work of switching.

Switching between tools adds users' responsibilities to the production-line, step-by-step task model: monitoring task progress, deciding when to change tools, and choosing the best next tool. The usability of complex software toolkits includes users successfully applying these functions as well as usable individual tools and planned workflows. Proficient toolkit users apply tools in different ways, with different frequency, in a different sequence, at different times. Some must be better than others. Their switching patterns should be reinforced with appropriate improvements in design.

Usability and productivity of software toolkits peak when users switch to the right tool at the right time, every time and steadily focus on the work, not the tools.

The problem

For a complex application (toolkit), the best design results in timely application of correctly chosen tools, each for as long as it remains the best choice. This means not only that we optimize individual tools, but also (a) minimize the labor of switching, (b) facilitate correct tool selection, and (c) arrange switching at the right time. Options *a* and *b* imply either planning of tool use, monitoring of task progress, or both to ensure that correct and timely choices be made. However, optimal work with a single tool requires neither option *b* nor *c*.

Long term goal

Tool switching has the following three aspects we would like to improve with better design:

- Accomplish each switch and choice smoothly and efficiently. Software's effectiveness suffers when users move between tools. Time is lost. Energy is wasted. Focus is dissipated. Information is difficult to transport from one tool to another.

- Facilitate the best tool choice. Users may be unaware of the “best” tool, or just not think of it, or switching might be too costly.
- Encourage tool change at the right point in the task. Transitions between tools can occur too soon (resulting in inefficient extra usage of the subsidiary tool) or too late (time and effort ineffectively spent before transition). Users need design support to make these decisions more effectively.

But we must walk before we can run. First we must be able to compare and evaluate patterns of tool choice and switch timing and determine critical incidents for study in realistic (complex) usability tasks or in actual work. Then we will know where to focus design effort and be able to measure how well our new designs foster more effective tool switching strategies.

Where to begin?

Before we get into analysis, and then into design (UCD practitioner) or experimentation (academic), we must ask, can we compare tool switching and patterns of switching effectively? This takes a lot of specific data and fresh ways of looking at it. The amount of raw data needed to study the subtasks and transitions that make up a complex “real” task can overload normal usability techniques. Test observations and facilitator’s notes do not suffice. We need “unattended data capture for portions of a long-term evaluation, used along with observations and interviews” (Redish, 2007, p. 107). This data is useless without effective tools and metrics that make it possible to study and to compare the patterns of use and prioritize the design effort.

Simple and complex tasks

In simple tasks, work steps and tool choices are known beforehand. Deviations from expected path and usage are very likely to be errors or inefficiencies. In a complex task, the workflow is not known ahead of time and generally cannot be planned. It is driven by unique or unfamiliar problems users encounter. In response, users apply tools chosen from the set they are aware of as best they can. To pose a complex task, we either (a) give users a problem without a formulaic solution (e.g., a programming problem) or (b) give users a simple task amid a large array of possible tools (e.g., plotting and formatting data). Complex applications evolve to facilitate complex tasks.

Background

Kuutti says, “[The focus of] mainstream HCI research is narrow, covering most adequately the area of error-free execution of predetermined sequences of actions” (1996, p. 37). Theoretical formulations that describe users’ engagement with complex tasks have not been coupled with workable evaluative techniques.

Usability testing limitations

Literature on testing of tool switching, because of its specific focus (multi-tasking) and limited scope (complex software), is sparse. To one side is conventional usability work (Dumas & Parsons, 1995; Rubin, 1994), and academic studies (Chapuis & Roussel, 1999; Card and Henderson, 1987), in which the analysis of simple tasks remains within the production line paradigm. This serves because workflows are known and tool options limited or equivocal. To the other side we find ambitious projects (Kline & Saffeh, 2005; Singer, Lethbridge, Vinson, & Anquetil, 1997; Weiderman, Habermann, Borger, & Klein, 1986) proposing to manage the entire software engineering life cycle. These papers are non-starters. They recognize the need for this kind of study without proposing strategies, let alone techniques, to gather and manage the enormous amount of data and observations that User-Centered Design requires.

Studying user-driven workflows is daunting. “Real” multi-tool tasks have an untestably large number of valid workflows, branching with every opportunity for tool change. Complexity of creative tasks and idiosyncratic work practices pull together the components of integrated development environments (IDEs) into what are, in effect, unique configurations each time they are used (Sy, 2006; Redish, 2007). Reported tests (Hanson & Rosinski, 1985; Leich, Apel, Marnitz, & Saake, 2005) treat IDEs as single entities with fixed workflows by evaluating them with simple tasks despite the fact that each user applies the components differently.

Tool choice and switch timing, when they are considered at all, are taken as either an outcome of training or acquired expertise (Card, Moran, & Newell, 1983; Kuutti, 1996), or the result of

the user's awareness of the task and the environment (Engestrom, 1999; Suchman, 2007). To date, neither approach has produced results that nourish effective design strategies for toolkits, which is what we are after.

Theories (Bødker, 1996; Engestrom, 1999 Green & Petre, 1996; Kuutti, 1996; Suchman, 2007) attempt to clarify the problem by more subtle description of what we observe without sufficient data and concrete examples to make them usable for the practitioner. This body of work sustains itself on elaborate schemas and thoughtfully studied examples that have undeniable explanatory and heuristic power. Unfortunately, when schemas conflict, debate collapses into whose example is more relevant.

Activity theory maintains that experienced users move unconsciously from step to step in a learned process until a problem (contradiction) intrudes, at which point the user is open to change and improvement of the process. Suchman and others suggest that step choices are "situated", that is, based on factors in the current problem state. Tension between these views remains because neither has been tested to determine the extent and actual conditions of its validity. Klein (1999), in a series of detailed interview studies of situated decisions and actions, gives evidence that the situation driven thinking described by Suchman is typically evoked by emergency or crisis situations (contradictions that cause process breakdowns). Klein's detailed study, drawing from many subjects engaged in widely different complex tasks, suggests that these two stances may be the same underlying behavior observed under different conditions.

The question of whether the "layered task execution choices" (Wild, Johnson, & Johnson, 2004) of this planning, monitoring, and decision making work can be supported, at least in part, by a user interface has also not been systematically tested. If transitions are marked and measured, their effects can be separated from the effects of the tools, and both can be studied in the context of complex tasks. The framework and vocabulary used by activity theory to describe how workflows change over time with experience provides a workable starting place. The issue of tool switching needs this depth of information to drive the design of better tools and toolkits.

Theoretical formulations of complex tasks

Activity theory calls the use of a tool an *operation* (Card et al., 1983; Kuutti, 1996). For the novice, an operation is a discrete series of actions, each of which must be, at first, chosen and planned. With practice, the need for planning and choice diminishes, and a chain of actions becomes one smooth operation. A tool works well if it allows productive work on an object. Two types of events take users out of the flow of productive work:

[Breakdowns] occur when work is interrupted by something; perhaps the tool behaves differently than was anticipated, thus causing the triggering of inappropriate operations or not triggering any at all. In these situations the tool as such, or part of it, becomes the object of our actions.... A focus shift is a change of focus or object of the actions or activity that is more deliberate than those caused by breakdowns.... Now the operations that she normally does become actions to her... (Bødker, 1996, p.150).

Breakdowns and focus shifts, although different in character, result from contradictions:

Activity theory uses the term contradiction to indicate a misfit within elements, between them, between different activities, or between different development phases of a single activity.... Activity theory sees contradictions as sources of development ... (Kuutti, 1996, p. 34).

These are opportunities for changing tools and opportunities for learning and, potentially, opportunities for reworking and improving the chain of actions that activity theory terms the operation.

Planning and switching

According to activity theory, an expert beginning an operation initiates a preset chain of actions. The operation's internalized chain of action is a form of planning. Contradictions result when these plans fail. In activity theory terms a contradiction signaling a breakdown or focus shift should occur at any point where the tool is not "doing its best." The contradiction alerts the user that the current tool or its usage is (at the moment, at least) suboptimal. The user must then decide whether to change tools or persevere.

These plans (chains of actions) may develop from experience (trial and error) or they may result from training, or simply arise from users striving to implement best practices. The result is the same. Actions proceed without evaluation until a contradiction intrudes.

The importance of tool switching

As soon as task success includes an aesthetic evaluation or quality goal, Kuutti's definition of success as "error-free" must be replaced by "optimal" or at least "satisfactory." Error-free is not enough. Steps must be of adequate quality, performed at an acceptable rate.

Having assembled actions into an operation that may be carried out without considering what to do next, the expert must remain aware of the quality of the work and the rate of progress in order to detect problems and act on them. These problems include not only contradictions, but also inefficiency from using the wrong tool or missing information. Any or all of these problems may require modification of the sequence of actions or a search for a more effective tool. It is precisely in this area that theories clash. Suchman warns that the planned selection of actions of an expert's operation do not account for choices and decisions made in specific situations that advance the work, but that they result from the user's engagement with the task(s) and could not be planned in detail in advance. Activity theory holds that operations disintegrate into actions separated by evaluation and choice when the operation encounters a contradiction, problem, or difficulty.

Unfortunately, "expert users, when asked, cannot report reliably about such cases of breakdown of their expectations, because they happen in the unconscious middle phase of an action" (Raeithel & Velichovsky, 1996, p. 201). In other words, the expert is unaware of any monitoring process until the operation breaks down, so the expert feedback reports only on the problem solving activity that it collapses into. We have no cognitive description of "checking for breakdown." As a practical matter we are left with the following:

- Operations are carried out with (novices) or without (experts) deliberation between actions until users detect contradictions (flaws in the state of the work). At this branch-point adjustments may or may not be made, depending on what tactics appear to be available or applicable.
- The signals that users detect may originate in the workflow, or from the user's monitoring of the work, or from a plan, or from an interruption by a co-worker or software agent. *Detect* means that users take themselves out of flow and choose the next action.
- Choice is constrained by the task environment, users' experience with the task, users' experience with the tools, and the user interface (UI).

Tool switching is the only contradiction-generated events that can be reliably detected. We know that a contradiction has been considered and a choice made.

Contradictions and interruptions

Activity theory is a theory of learning. Contradictions "are not the same as problems or conflicts. Contradictions are historically accumulating structural tensions within and between activity systems" (Engestrom, 1999, p. 3). For our purposes it is not important, at least initially, to distinguish between contradictions whose resolution generates a new form of operation and interruptions that cause the user to notice that the workflow has bogged down. Our interest begins when any event initiates consideration of a tool change tactic.

Interruptions have received more concrete study than contradictions. Mark, Gonzales, and Harris (2005) call interruptions from within the working sphere "interactions," those from outside the working sphere are "disruptions" that reflects a negative impact on task progress. Adamczyk and Bailey (2004) show that the cognitive impact of interruption varies according to which stage of the task (beginning, middle, end) it occurs. Cutrell, Czerwinski, and Horvitz (2000) attribute more impact to interruption of chunking behaviors (highly focused subtasks) than to interruptions happening between them. Speier, Valacich, and Vessey (1997) demonstrate that the disruptive impact of an interruption increases with the complexity of the task. These definitions and characterizations are useful in describing what is observed in the tool switching process.

Methods of observation

Studies of multi-tasking and task switching in the literature provide a framework for categorizing data and controlling the test environment. A general taxonomy for modeling multitasking (Wild et al., 2004) enumerates possible sources of error and distortion in our test environment and protocol. The test environment provides that test participants do not inhabit different roles (developer, manager, leader, or contributor), that there are no interruptions from outside the task, that there are no proximate external forces such as job or peer pressure, that there are no events hindering one tool in favor of another, and that there is no change in work environment. The following are the most important points to consider:

- No distinct goals separate operations and actions into different but interwoven, workflows. Tasks in this study have a single goal. No other tasks compete for attention.
- The character of the goal and how it affects activity does not differ from participant to participant. The goal in this study—to finish as many subtasks as possible in the allotted time—engages all equally.

Compared to most multi-tasking studies, usability testing is clean-room control.

Learning about tool switching begins with an accurate picture of what actually happens. So, to perform the necessary basic studies, we need a technique that extracts and summarizes pertinent (to tool changes) data and metrics from testing and focuses on critical incidents with little or no “dog work” to encumber the practitioner’s day job.

Logging and reporting

Microsoft used silent logging extensively and famously to gather design requirements for Vista (Harris, 2008). A web site, Hackystat (<http://code.google.com/p/hackystat/>), provides an open source framework that enables anyone to log their own activity on a remote server and retrieve regular reports.

Recording operating system focus and user input captures time spent with each tool and an indication of the activity. The log record’s time signal indexes into a recording of screen activity and users’ commentary (if any). Additionally, success in tasks and subtask steps can be measured using completion, time to completion, user satisfaction, and the System Usability Scale (Brooke, 1996; Tullis & Stetson, 2004). We use Techsmith’s Morae®, which captures some tool switching information, to record screen and voice.

A Java applet logs window activity and user input from within MATLAB® capturing all activity that can be precisely distinguished. A Visual Basic for Applications (VBA) script on the back end of the logger produces timeline pictures of tool use and changes, and statistical descriptions (counts and charts) that summarize the patterns with exhibits such as those presented in this paper.

How to read the charts

Task 1 is a problem without a formulaic solution (e.g., a programming problem). Task 2 is simple plotting task amid a large array of possible tools (e.g., plotting and formatting data). In the programming Task 1, users create, step-by-step, a function or script that can find palindromes in text. They employ very few of the available tools. The complexity in this case lies in deciding when to switch from one tool to another. In the plotting Task 2 users load data from a file and create plots for export or printing. In this case the complexity is tool choice; there are many ways to accomplish Task 2 in MATLAB®.

We recorded 14 users doing each of the two tasks. This gave a feeling for the variation in tool use across users (see below) and a baseline against that we can measure our next set of design changes. The charts show the user’s progress through the task in terms of tool use as well as indicating frequency and direction of tool changes so as to give a quick, overall picture of what tools were applied, when the tools were applied, and how long the tools were applied. The following is a description of how to read each line in the figures that follow:

- The x-axis is a timeline in minutes. Each plot begins when the user finishes the first reading of the task and begins to work with the software.
- Row 1 (the bottom row) contains the helper tools that support both the command window and the editor (file browser, several ways of looking at variables, and a

command history). It's a measure of the simplicity of the programming task that they get so little use.

- Rows 2 and 3 (from the bottom) of the timeline charts show the (legacy) Command Window (row 2), from which MATLAB® was originally run, and the Editor (row 3), where users write more elaborate code to manipulate the same component mathematical functions. Each of these makes its own contribution to the product's capability. Working between them and using each most effectively has been difficult for users from the start.
- Row 4 is Help and Documentation. Switching to help is present (in varying degrees and with different severity) at all levels of expertise, and in every task. Users' aversion to this instance of tool switching is well documented (Grayling, 2002). We anticipate new design work in this area. There is no problem accumulating large amounts of data.
 - Success of the help-seeking step is (relatively) easy to measure. (The user's evaluation is likely to be accurate.)
 - The contradiction that provokes switching from a work-piece to help is comparatively easy to observe—users discover a need for information that they must then search for—and therefore to measure and evaluate.
- Row 5 (the top row) shows a group of GUI tools used for plotting, including the window where the actual plot appears.

This visual overview of test progress allows us to locate switching problems in a test recording, and affords a description of tool use that we can compare with tests of the same task using other designs.

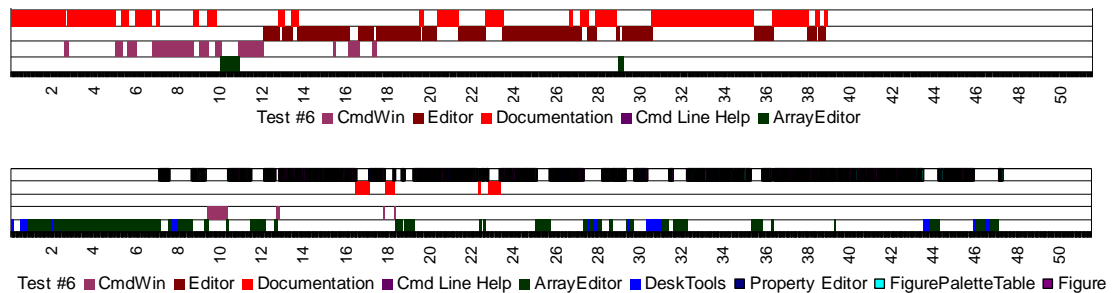


Figure 1. User 6 working through the programming task (top) and the plotting task (bottom).

Figure 1 shows User 6 doing Task 1 and then Task 2, using different tools for each task. In Figure 2, user 2 plots with the same toolset user 6 used for programming. User 2's style (in the period from 10:00 to 50:00) demonstrates that user 6 might have done the plotting task with the same tool use tactics used to complete the programming task, but chose not to.

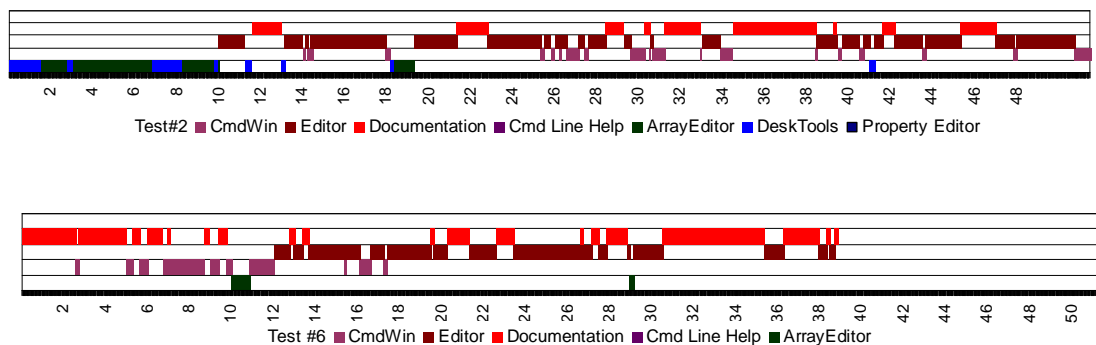


Figure 2. User 2 does the plotting task (top). User 6 does the programming task (bottom).

Interpreting the timelines

With practice, we can pick out problem areas that might have been overlooked as the test was running. For example, the brief visit to the documentation that user 6 made at about 22:30 into the plotting task (see Figure 1) resulted in misleading information, requiring another visit to the documentation a minute later. Of course, the cause of this problem can't be read off the timeline. But investigating brief visits to documentation is almost always valuable from a usability viewpoint. When the trip is successful you can take the time to figure out what worked. We see in the timeline when they must repeat the visit (usually because they have not been able to bring back enough information).

Long visits to helper tools (row 1) may also be of interest. For example, why is user 6 spending so much time in the Array Editor? In addition, the statistical data on which this picture is based will quickly show whether other users also spend large amounts of time with a relatively non-productive tool. But the main value of the timelines here is their depiction of different users' tool strategies.

All of this paper's key questions appear in Figures 1 and 2. The tools (level 5 in the timeline) user 6 applied to the plotting task are designed for plotting. User 6 completed 10 of 13 segments of the task with them. User 2 applied general-purpose programming tools to the same task, completing 5 segments in about the same length of time.

We might say that user 6 was where we wanted him to be (in terms of tools), and user 2 was not. Yet user 2's Standard Usability Scale (SUS) score was almost twice that of user 6 (72 to 40). In relative terms, then, user 2 was comparatively unaware of how poorly the tools he chose were performing. To compound the problem, user 6 did much less well on the programming task than on the plotting task, finishing only one segment, yet his SUS score for that task was ~50% higher than for the plotting task (57.5 to 40).

The timeline view enables flagging of critical points in the testing (previous section), and grouping of workflows by patterns or styles, as shown here and in the discussion. The following sections present the limitations of the current version.

Drawbacks and advantages of the logged data

This automatically captured data provides us with information about users' workflow which we could obtain in no other way, but it is neither perfect nor complete. We can make most effective use of it by remaining alert to the following limitations:

- We cannot identify and measure periods of inactivity. We can't say whether users are puzzled by the work, waiting for the software, or sipping coffee. We can't distinguish waiting and thinking from just waiting.
- Window focus (which is what we detect) is not always equivalent to user focus, although users must act to change focus. Activity in a period of focus without logged user input may be of little or no significance to the work.

While these limitations may lead to misinterpretation of individual usages and switches, they are more likely to be associated with statistical outliers—overlong dwells in the first case, and very rapid changes in the second.

These problems should be seen against the main advantage of the method, which is to gather and manage large amounts of data during conventional usability testing with little or no additional effort. Large amounts of data are essential to this investigation in order to identify and compare patterns of behavior that are, even without the noise generated by inactive periods and rapid focus shifts, statistical. This logging, counting, and plotting extend conventional usability practice by enabling consideration of new problem areas.

Rating

We rate tool switching and tool switch patterns in two ways. First, switches that statistically dominate successful task execution (defined both as task completion and satisfaction with the task experience, as measured by SUS) are rated as preferred by users. Secondly, however, the choices made in the course of the work can be evaluated as regards timing (by observing whether task progress before the switch point was satisfactory) and in terms of choice (by observing whether users choose the most effective tool).

These ratings are concrete data descriptive of users' cognitive behavior, based on observation, consistent with other summative work in usability studies. Their quantity and quality are made possible by automated detections and pre-processing of all events in this category, and artificially extending (and compressing) the facilitator's observation time through the use of recordings and flagging of events. The technique remains qualitative; its foundation is the experienced observer's analysis of critical incidents. However, quantitative methods (plots and statistics) identify patterns and clustering, which suggest where those critical incidents lie.

Analysis

The analysis has four steps: choice of event types (or perhaps design changes) for study, parsing and processing of logs into statistics or graphics that arrange the data into useful views, pairing events and patterns with their metrics, and checking conclusions against test recordings. The final step connects the statistical abstraction with actual test observations, and also warrants that conclusions are not distorted by outliers.

We have already observed problem patterns prior to the study that stands out in the timelines and data. We expect to discover further orderings of subtasks (groupings or effective sequences) that, although they alter the naïve flow of the task, are so evidently more productive that design should reinforce them. One example of such a technique is copying and pasting if, then, and else constructs as a block, and then updating the contents of the new block. Speed, accuracy, and neatness are achieved by "going out of order." These sometimes unexpected strategies are fruitful because of dependencies between subtasks that users discover and exploit (Card et al., 1983). Graphical presentation of workflows (Figures 3 and 4) help us detect patterns and dependencies.

Results

At this time, we have refined generation of the timeline and automated some charts that summarize and compare data. However, the data so far is only a baseline sample. The acid test of the technique and approach will come when we run the same tasks under the same conditions with improved designs. We expect that differences in tool usage will be easy to study and evaluate using this approach. Meanwhile, the results to date do support some observations of interest and potential value. (This discussion is limited to the programming task, which Users 3 and 5 did not attempt.)

We asked each user to fill out the Standard Usability Scale (SUS) after the task, and we recorded the number of task steps each user finished. The users also supplied information about themselves, rating themselves expert (E) or novice (N), on level of skill and comfort with MATLAB (1-5 low to high), and whether they thought of themselves as programmers or not. They also reported the number of files they wrote per month, and years they had used the product.

Table 1. User Data from the Programming Task

User	SUS Score	Completed	Self-Rating				Experience		Editor-Command Window Switches
			Expert?	Skill with MATLAB	Comfort with MATLAB	Programmer	Files/Month	Years Using MATLAB	
1	60	2	E	3	4	Y	1	6	61 (High)
2	82.5	1	N	2	4	N	7	0.5	17 (Middle)
4	87.5	5	E	4	5	Y	24	5	90 (High)
6	57.5	1	N	2	3	N	9	2	4 (Low)
7	45	0	N	2	3		1	8	6 (Low)
8	40	1	N	1	2	Y	2	0.5	9 (Low)

9	70	1	N	1	3	N	4	5	8 (Low)
10	67.5	3	E	3	3	Y	12	3	29 (Middle)
11	90	3	E	1	3	Y	0	4	0 (Low)
12	100	1	N	3	4	Y	15	1	28 (Middle)
13	70	2	N	3	5	N	5	1.5	18 (Middle)
14	80	2	E	1	4	N	20	0.2	26 (Middle)

No correlation between user scores and their self-descriptions was apparent, nor was it expected. However, it was possible to group users meaningfully by their tool switching behavior, which is the content of the last column. How this was done appears in Figure 3.

Tool switching styles of 11 users who completed the programming task are presented as point data connected by lines in Figure 3. This view accentuates user differences and similarities around individual transitions. We can immediately pick out users 6 and 13 moving between the Editor and the Documentation, user 1's frequent use of the Array Editor, and so forth. We can also separate the users into three groups based on their frequent switching in and out of the Editor and the Command Window (the first two data on the x-axis). We can see what this grouping implies by comparing the "styles" depicted by the users' timeline diagrams. (This grouping is analyzed in Table 2.)

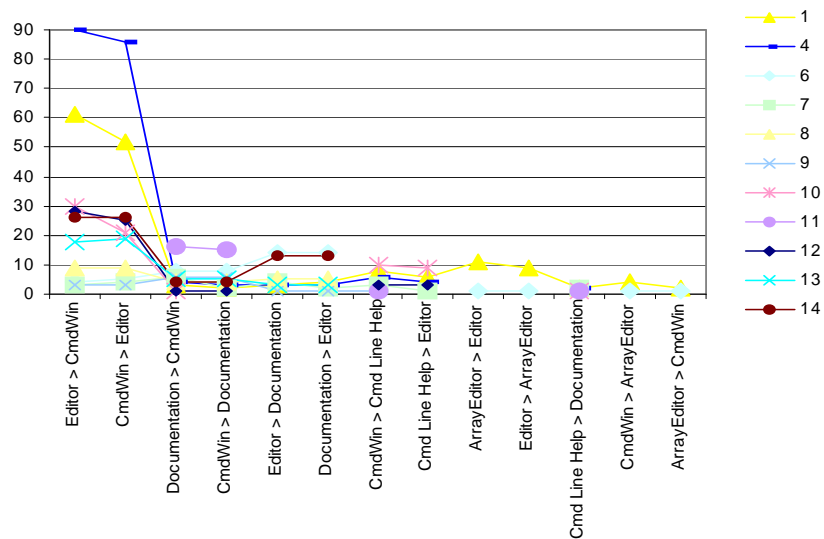


Figure 3. Transition statistics by user for the programming task, the frequency of switching from Editor to Command Window suggested the division of users into three groups: High > 50, Middle 18-30, and Low < 10. (Not all users tried both tasks; user 2's log was corrupted.)

Figure 4 shows a timeline from each of the three groups. There are other visible differences, but sample sizes are too small for anything but speculation at this point.

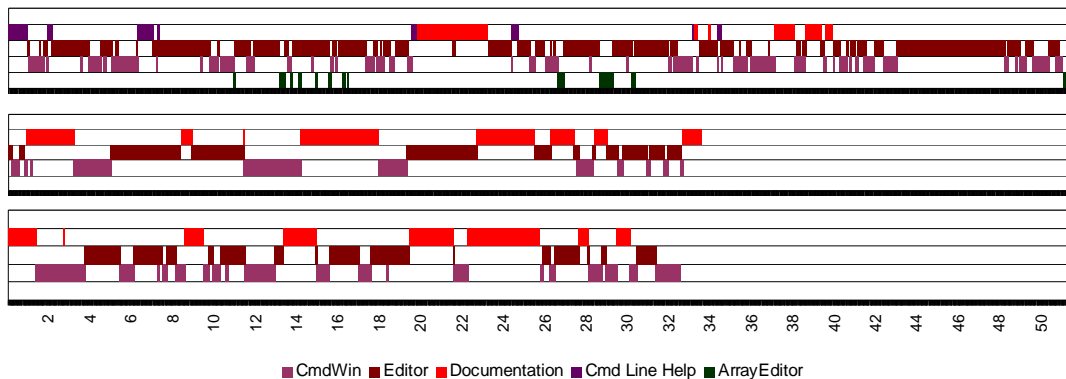


Figure 4. User 1 (High group, top), user 8 (Low group, middle), and user 13 (Middle group, bottom) working on the programming task.

Discussion

We revisited whether some of the factors given in Table 1 contribute to differences in tool switching behavior such as those shown in Figures 3 and 4. Table 2 collapses Table 1 into three groups, either by averaging individual results or by expressing class (expert or novice) representation as ratios. These are the same groups that appear in the last column of Table 1.

Before examining Table 2, the experimenter naively assumed that the Low group had the best tool switching strategy and the High group the worst, based on a gut feeling that switching should be minimized. He could not have been more wrong. Not only did the Low group perform the worst (fewest segment completions, Finished (of 5) section in Table 2), but they were the most unhappy with the tools (SUS), and had the lowest in exposure to the product (3.2 files written per month). The High group, whose timelines appear almost dysfunctionally scattered, were in fact all-expert, and they proved it by completing by far the most task segments. Not only did they score the best, they also rated themselves the best and most comfortable with the software.

Table 2. Metrics by Style Grouping of Switching Statistics

Group	Expert: Novice	SUS	Finished (of 5)	Skill (MATLAB)	Comfort (MATLAB)	Years	Programmer: Plotter	Files/ Month
High	2:0	73.75	3.5	3.5	4.5	5.5	2:0	12
Mid.	2:3	80	1.8	2.4	4	1.24	2:3	11.8
Low	1:4	60.5	1.2	1.4	2.8	3.9	2:2	3.2

The manner in which this grouping of users by tool switching style organizes scores, subjective evaluations of MATLAB® (SUS), and users' self-ratings and experience is the most important finding in the study thus far. It counter-intuitively suggests that more keystrokes and more switching "work" can be better, which goes to show how much we have yet to learn about tool switching.

Recommendations (What's Next)

Making tool switching painless and quick may not be enough, and perhaps not even the highest design priority. Experienced users employ techniques—and choose tools—that appear rapid and efficient because using them has come to seem easy. Some are effective and some are not. For example, every user has his or her way to access help and support information. These patterns can't all be optimal, even if users think they are. Some users get better results than others; their tool choices and switching strategies will be the ones to reinforce in design.

Development of new designs to support tool switching highlights some areas of further investigation that are likely targets for work carried out with the techniques that are discussed in the following sections.

User self-monitoring

The notion of best practices and much of the content of training hinge at least in part on the assumption that users can monitor the quality of their work. In terms of this study, users discover through evaluation of work progress that they need to change tools. Our more narrow interest would be whether design could support this monitoring activity or not.

Studies of pair programming shed interesting light on this area: Experimental work pairing experts with novices shows experts' awareness of task barriers (contradictions) when observing novices encountering them (Raeithel & Velichovsky, 1996). It is not clear, however, that these experts monitor their own activity with the same care. Pair programming, in which one user codes while a teammate takes a supervisory view and (theoretically) makes good switching decisions, should model any system that notifies users of contradictions in their ongoing work (Domino, Collins, Hevner, & Cohen, 2003). An ethnographic study of pair programming in practice suggests that this is not so easy, asking at one point "how it would even be possible for two people working at different levels of abstraction to successfully sustain a conversation at all" (Chong & Hurlbutt, 2007, p. 2). Also, the authors repeatedly observed that the best-functioning pairs are also peers at skill level and experience (Chong & Hurlbutt, 2007). Advice, however good, administered injudiciously, can be counterproductive. Experiments harnessing the joint attention of two users (Raeithel & Velichovsky, 1996) appear to be more encouraging, suggesting that appropriately designed software agents might support the monitoring task better than peers hampered by their social baggage.

Task specificity of tools

Nardi (1996) suggests classification of tools as more or less task specific according to the variety of tasks that a task is suited for. As an example, the task of comparing two files can be carried out in most editors, but awkwardly; the same task can be handled much more quickly and efficiently in a difference editor. It's plausible that users resist switching to a task-specific difference editor simply because the comparison is only a small part of the overall task, and the difference editor is not hospitable to most of what the user is trying to accomplish. The effect of task specificity in tool choice appears in frequency of switching in tasks where use of the tool is appropriate, and in delay (inefficient attempts to complete the work) before moving to the tool, and then moving back.

A more productive understanding of expertise

Although complex software applications must be accessible to novices, for users to remain novices is undesirable. Yet the definition of expert remains elusive. Since user 4 scored as an expert, and rated himself as an expert, I reviewed his test recording to see if there was anything concrete I could say about his expertise. What I saw was an expert as defined by activity theory. No time was spent deciding whether or not to change tools, or which tool to change to. And with each tool his actions were swift and sure, at times almost too fast for me to follow. He was undeniably expert (smooth and rapid) executing the steps of his workflow, and expert (economical and effective) with the tools he chose.

But I could not rate his workflow or his choice of tools so highly. Having watched many programmers try this task, I know that there are better orderings of steps, and better choices of tools than the ones that this user had internalized. Coleridge (1827) defined prose as "words in their best order" and poetry as "the best words in their best order." User 4 produced eloquent prose. The highest level of expertise must be more like poetry.

Activity theory describes the process by which novices learn to produce satisfactory prose. When a workflow is derailed by one of Engestrom's contradictions the same process heals and improves it. But users with operations that employ less effective tools, or switch between tools at less than optimal times, don't run into contradictions. And yet a new and better tool (within the application) that users believing themselves to be functioning at the level of expert do not employ makes them, in some important sense, less than expert.

Adding tools to an application's mix will not likely help users. Even if they (a) become aware of the new features (not easy), and (b) try to weave the new tool into their workflow (lots of extra work). Determining which tool (old or new) works better is neither straightforward nor foolproof.

Extensions of Engestrom's "contradictions" are more likely to succeed if users are made aware of more effective alternative tools at the moment of choice, or notified of ineffective performance at a point when they feel the need to "upgrade" their process. The trick, of course, is making users aware of the information at that pivotal moment.

Interruptions by design

Because of their focus on task work, users sometimes miss points where they ought to be branching. This is why study of interruptions is vital; if users are to make the best choices at the best points in the workflow, can they be prompted to do so?

We want to be able to inject task interruptions in the form of alerts and reminders to influence better tool switching and to improve the timing of tool switching by helping users decide when to switch. The most interesting studies on interruptions deal with interruptions that are distractions from the task (Cutrell et al., 2000; Mark et al., 2005; Speier, Valacich, & Vessey, 1997).

In our testing, interruptions originate either with the participants when the task flow breaks because questions they can't answer come up in the progress of the work ("How do I finish this for loop?"), when the computer throws an error or performs an unexpected action, or when reaching the end of a subtask. Interactions occur at natural activity boundaries and are adequately characterized in our data by the before and after activities and time measurements, frequency, and positioning along the length of subtask and task.

Interruption statistics, within-task completion times, and users' satisfaction with their work indicate the relative magnitude of impact due to interruption position in the task in user input. We can also observe whether interruptions are preceded by a tapering off of productivity as Speier (1997) predicts.

Design for more timely tool changes requires affordances in the user interface that alert users to the pace and effectiveness of their work.

Conclusion

Tasks that are the work of large software applications are typically complex. Neither the user nor the usability tester knows either the path the workflow will take or which tools will be brought to bear. Users' awareness of points where they change tools, and the choice of tools they make at those points, directly affects productivity and creativity. Training and the inculcation of best practices may well improve this awareness and decision making ability, but most users never get either. Improved design for making these choices and transitions, on the other hand, reaches every user every time a tool needs changing.

A model of user action made up of steps or segments that smoothly merge into an efficient operation as expertise increases falls short as a design goal for complex software because it omits the following:

- Switching from one tool to another at times that depend uniquely on task content and progress. Users must become or be made aware that the work is not advancing as well as it might be, and that a tool change is the right next move.
- Consideration and study of complex, "real" tasks wherein the user discovers or invents the workflow. The switch from one optimal tool to another at the appropriate time is a key problem in this discovery and invention of a workflow.

- Discovery and adoption by experts of better, more appropriate tools—simply tools that are newly available. As soon as the task is fresh, or in any way different, the expert's polished workflow, however impressively effective, is potentially ripe for improvement. Experts must cope with this possibility without losing their effectiveness.

We have only vague notions of how design may help users to use the best tools in the best order, and how to test proposed improvements. We need to replace these notions with studies and observations.

Practitioner's Take Away

Methods presented in this paper serve as a foundation for the following:

- Comparison of users' performance in complex tasks by their choice of workflow path elements (tool use) and strategies (tool choice and decision timing).
- Methods of visualizing and analyzing logged computer-user interaction data to study switching behavior.
 - Transition statistics (traffic between tools, time on support tools) locate intense switching activity and compare one user's tool strategies to another's.
 - Transition timelines (graphics) highlight problematic patterns of activity.
- Use of transition statistics as metrics to identify, zero in on, and characterize successful and unsuccessful design features.
- Use of the Standard Usability Scale (SUS) and task completion scores to indicate users' awareness of their tool switching patterns and the effectiveness or efficiency of these patterns.
- Use of logging, counting, and plotting (visualizing) test data to extend conventional usability practice by enabling consideration of new problem areas.
- A set of baseline data against which design changes in a specific area of MATLAB®, or any other complex software toolset, can be evaluated for their impact on tool switching behavior, alongside their effect based on standard usability metrics.

Acknowledgements

Andrew Wirtanen and Mike Ryan, who facilitated the testing, Amy Kidd, who helped me plan it, Jared Spool, who taught me how to design *real* tasks, Donna Cooper, the MATLAB LCT Usability team, Chauncey Wilson, the excellent peer reviewers, and others who grappled so supportively with ideas out of order, and, of course, the participants.

References

- Adamczyk, P. D. & Bailey, B. P. (2004, April). If Not Now, When?: The Effects of Interruption At Different Moments in Task Execution, *CHI 2004*, 24-29 (pp. 272-278) Vienna, Austria.
- Blackwell, A., Robinson, P., Roast, C., & Green, T. (2002, April). Cognitive Models of Programming-Like Activity, *CHI 2002*, 20-25 (pp. 910-911) Minneapolis, MN, USA.
- Bødker, S. (1996). Applying Activity Theory in Video Analysis: How to Make Sense of Video Data. In HCI in Bonnie A. Nardi, (Ed.) *Context and Consciousness: Activity Theory and Human-Computer Interaction* (pp. 147-174). Cambridge, Massachusetts: The MIT Press.
- Brooke, J. (1996). SUS: A Quick and Dirty Usability Scale. In P.W. Jordan, B. Thomas, B.A. Weerdmeester, & I.L. McClelland (Eds.) *Usability Evaluation in Industry*. London: Taylor & Francis. (Also see <http://www.cee.hw.ac.uk/~ph/sus.html>)
- Card, S. K. & Henderson, A. Jr. (1987). A Multiple Virtual-Workspace Interface to Support Task Switching (pp 53-59) *CHI 1987*.
- Card, Stuart K., Moran, T. P., & Newell, A. (1983). *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Chapuis, O. & Roussel, N. (1999). Copy-and-Paste Between Overlapping Windows. *CHI 2007 Proceedings-Navigation & Interaction, April-May 2007* (pp. 201-210) San Jose, CA, USA.

- Chong, J. & Hurlbutt, T. (2007). The Social Dynamics of Pair Programming. *29th International Conference on Software Engineering (ICSE'07)* 0-7695-2828-7/07.
- Coleridge, S. T. (1827). *Table Talk – July 12, 1827, in Collected Works*, vol.14. In Kathleen Coburn 1990 (Ed.).
- Cutrell, Edward B., Czerwinski, M., & Horvitz, E. (2000, April). Effects of Instant Messaging Interruptions on Computing Tasks. *CHI 2000* (pp. 99-100).
- Domino, M. A., Collins, R. W., Hevner, A. R., & Cohen, C. F. (2003). Conflict In Collaborative Software Development. *SIGMIS '03* (pp. 44-51) Philadelphia, PA.
- Dumas, J. & Parsons, P. (1995, June). Discovering the Way Programmers Think About New Programming Environments, *Communications of the ACM*, 38 (6), pp. 45-58.
- Engeström, Y. (1999). Innovative learning in work teams: analyzing cycles of knowledge creation in practice. In Y. ENGESTRÖM et al (Eds.) *Perspectives on Activity Theory* (pp. 377-406). Cambridge: Cambridge University Press. Retrieved April 27, 2008, from <http://www.bath.ac.uk/research/liw/resources/Models%20and%20principles%20of%20Activity%20Theory.pdf>.
- Grayling, T. (2002). Fear and Loathing of the Help Menu: A Usability Test of Online Help, *STC Technical Communication* 45 (2) 168-179.
- Green, T. R. G. & Petre, M. (1996, June). Usability Analysis of Visual Programming Environments: A cognitive dimensions framework, *Journal of Visual Languages & Computing*, 7 (2) 131-174.
- Hackystat Development Site. Retrieved on May 14, 2008, from <http://code.google.com/p/hackystat/>.
- Hanson, S. J. & Rosinski, R. A. (1985). Programmer Perception of Productivity and Programming Tools, *Communications of the ACM*, 28 (2) 180-189.
- Harris, J. (2005, October 31). Inside Deep Thought (Why the UI, Part 6). Retrieved on May 14, 2008, from <http://blogs.msdn.com/jensenh/archive/2005/10/31/487247.aspx>
- Harrison, W., Ossher, H., & Tarr, P. (2000). Software Engineering Tools and Environments: A Roadmap, *Future of Software Engineering* Limerick Ireland.
- Klein, G. A. (1999). *Sources of Power: How People Make Decisions*. Cambridge, MA: MIT Press.
- Kline, R. R. & Saffeh, A. (2005). Evaluation of integrated software development environments: Challenges and results from three empirical studies, *Int. J. Human-Computer Studies* 63, 607-627.
- Kuutti, K. (1996). A Framework for HCI Research. In Bonnie A. Nardi, (Ed.) *Context and Consciousness: Activity Theory and Human-Computer Interaction* (pp.17-44). Cambridge, Massachusetts: The MIT Press.
- Leich, T., Apel, S., Marnitz, L., & Saake, G. (2005, October). Tool Support for Feature-Oriented Software Development: FeatureIDE: An Eclipse-based Approach, *eclipse '05*, San Diego, CA.
- Mark, G., Gonzalez, V. M., & Harris, J. (2005, April). No Task Left Behind? Examining the Nature of Fragmented Work, *CHI 2005* (pp. 321-330) Portland, Oregon, USA.
- Nardi, B. (1996). Reflections on the Application of Activity Theory. In Bonnie A. Nardi (Ed.) *Context and Consciousness: Activity Theory and Human-Computer Interaction* (pp.235-246) Cambridge, Massachusetts: The MIT Press.
- Raeithel, A. & Velichovsky, B. M. (1996). Joint Attention and Co-Construction of Tasks. In Bonnie A. Nard, (Ed.) *Context and Consciousness: Activity Theory and Human-Computer Interaction* (pp.199-233). Cambridge, Massachusetts: The MIT Press.
- Redish, J. (2007, May). Expanding Usability Testing to Evaluate Complex Systems, *Journal of Usability Studies*, 2, (3), 102-111.
- Rubin, J. (1994). *Handbook of usability testing: how to plan, design, and conduct effective tests*. New York, NY: John Wiley and Sons.

- Singer, J., Lethbridge, T., Vinson, N., & Anquetil, N. (1997). An Examination of Software Engineering Work Practices, GASCON 97.
- Speier, C., Valacich, J. S., & Vessey, I. (1997, December). The Effects of Task Interruption and Information Presentation on Individual Decision Making, *ICIS '97: Proceedings of the eighteenth international conference on Information systems*.
- Suchman, L. A. (2007). *Human-Machine Reconfigurations: Plans and Situated Actions 2nd Edition*. Cambridge: Cambridge University Press.
- Sy, D. (2006). Formative usability investigations for open-ended tasks, *UPA 2006 Conference Proceedings*.
- Tullis, T.S. & Stetson, J. N. (2004). A Comparison of Questionnaires for Assessing Website Usability, *Usability Professional Association Conference*.
- Weiderman, H. H., Habermann, A. N., Borger, M. W., & Klein, M. H. (1986). A Methodology for Evaluating Environments, *ACM 1986*, 0-89791-212-8/86/0012/199.
- Wild, P. J., Johnson, P., & Johnson, H. (2004). Towards a Composite Modeling Approach for Multitasking, *TAMODIA '04* (pp.17-24) Czech Republic.

About the Author



Will Schroeder

Will Schroeder has seventeen years designing and testing hardware and embedded software at Foster-Miller, Inc. He holds or co-holds five patents. Twelve years as Principal at User Interface Engineering—usability consulting and research on hardware, software and the web. Two-time CUE participant. Principal Usability Specialist, The MathWorks, since 2005.