AppMap: Exploring User Interface Visualizations

Michael Rooke, Tovi Grossman, George Fitzmaurice

Autodesk Research, 210 King St E, Toronto, ON, M5A 1J7

ABSTRACT

In traditional graphical user interfaces, the majority of UI elements are hidden to the user in the default view. Application designers and users desire more space for their application data and thus want to minimize the user interface footprint. We explore the benefits of dedicating additional screen space for presenting an alternative visualization of an application's user interface. Some potential benefits are to assist users in examining complex software, understanding the extent of an application's capabilities, and exploring the available features. Thus, we propose *user interface visualizations*, alternative representations of an application's interface augmented with usage information. We first introduce a design space for UI visualizations and describe some initial prototypes and insights based on this design space. We then present AppMap, our new design, which displays the entire function set of AutoCAD and allows the user to interactively explore the visualization which is augmented with visual overlays displaying analytical data about the functions and their relations. In our initial studies, users welcomed this new presentation of functionality, and the unique information that it presents. We conclude by summarizing some potential benefits of UI visualizations.

KEYWORDS: AppMap, User Interface, Visualization.

INDEX TERMS: H5.2 [Information interfaces and presentation]: User Interfaces. Graphical user interfaces.

1 INTRODUCTION

Today's large computer software applications can expose an overwhelming amount of functionality to its users [16]. With so many functions, the majority of a UI is hidden in the default view, to maximize screen real estate for the working document (see Figure 1). Graphical user interfaces (GUIs) serve as tools for access, providing mechanisms for users to navigate through ribbons, dialog boxes, tabs and menu systems [28]. While this provides a logical structure for interaction, users may have trouble locating desired functionality and may not be aware of certain features that are hidden away in these nested UI components [14].

Figure 1. UI screenspace consumption for the default views of Microsoft Word (13%), Autodesk AutoCAD (28%) and Adobe Photoshop (29%) running at 1280x1024 resolution.

mike.rooke@gmail.com
tovi.grossman@autodesk.com
george.fitzmaurice@autodesk.com

While it may not be their primary purpose, a GUI also inherently serves as a visual representation of what features the system has to offer, and as a tool for exploring and becoming aware of those features [28]. But, because this is not their primary purpose, we argue that traditional user interfaces are not optimally designed to provide awareness and exploration of features. For instance, users rarely have the ability to graphically view the entire scope of the software's functionality, and how those functions relate to one another. As such, becoming aware of relevant functionality, and establishing an overall familiarity with an application can be challenging, especially for complex software applications. For example, Grossman et al. found that functionality awareness was a specific learnability problem impeding both novice and expert users from completing tasks efficiently or at all [14].

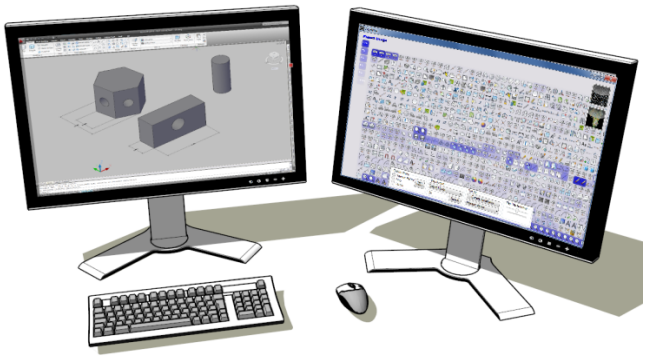

Figure 2. AppMap UI Visualization.

Just as cities have tour books and web pages have sitemaps [5, 19, 23], we contend that software applications could provide *user interface visualizations* – alternative representations of the UI for improving the examination of an application. We first describe related work and then introduce a design space for UI visualizations. Guided by informal lessons learned from initial mockups, we present AppMap, an interactive user interface visualization system (Figure 2).

Specifically, AppMap provides an interactive visualization of an *entire* program's functionality to support exploration and awareness of complex end-user interfaces. AppMap allows users to grasp what a program does and how it is structured without having to find and make sense of explicit functionality through program usage. Information is laid out in either a grid view (Figure 2) or map view (Figure 3). To help users indentify the specific commands that are relevant to their own usage of the software, AppMap provides analytical information about commands, through interactive visual overlays and spatial layouts. In initial user observation sessions, we found that AppMap was well received and easy to use, delivering information which users found to be useful.

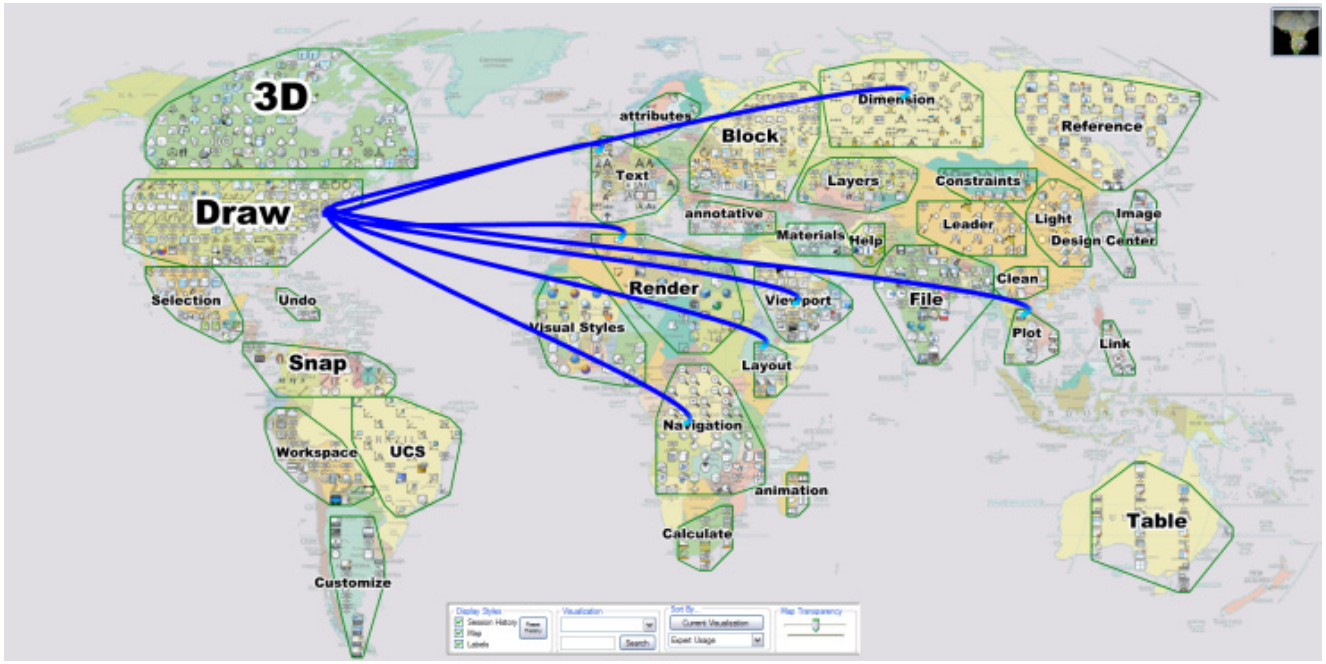

Figure 3. AppMap – an application *User Interface Visualization* exposing and organizing all command functionality. In the *map view,* the visualization is grounded to a world atlas. Blue lines point to likely next used commands.

2 Related Work

2.1 The Problem of Feature Awareness

Many computer programs of moderate complexity hold more functionality than single users actually use [22]. In their paper on software learnability, Grossman et al. highlight the important issue of *awareness* of functionality. In particular, even an "expert," who has mastered certain aspects of a system may be completely unaware of other tools that could improve their overall efficiency [14]. In a recent study, Matejka et al. found that users greatly overestimated the true fraction of functionality they used [21]. An abundance of functionality is often labeled "bloat" if it reaches a subjectively daunting size [22].

For a beginner, a fully featured program often requires trial and error to locate desired components since many of these components are not directly visible. In researching word processor learning, Carroll and Carrithers [3] found that "new users often recklessly tried out menu choices in their early encounters with the system." Their investigation demonstrated that showing the full-featured program to a beginner was not the optimal approach to learning it.

However, it was found by McGrenere and Moore that most users do not want a stripped-down version to suit their own needs – they prefer to have functionality that they can discover, even if they never use it. They explained, "this discovery of a set of unused features, both wanted and unwanted, that is subjectively defined by each user, opens the design space and raises new challenges for interface designers" [22]. However, a balance must be struck at some level of complexity. Hsi and Potts found that adding functionality to an existing program generally increases complexity, especially when this process is repeated version after version [16]. These contrasting finding demonstrates the challenge in designing an interface that is discoverable but not complex. AppMap attempts to address this challenge by providing a UI visualization: an alternative representation of the user interface which promotes discovery, so the primary representation can maintain a low level of complexity.

2.2 Adaptive User Interfaces

One line of research, which could potentially combat the challenge of software bloat and user confusion, is to design adaptive user interfaces, which adapt to the user's behavior [11, 12]. Most relevant to software bloat are multilayered user interfaces [9] or "training wheels" [3] which gradually reveal functionality to the user as they progress in expertise. Unfortunately, there are disadvantages associated with adaptive interfaces. In particular, Hui et al. argue that adaptive interfaces can induce a *disruption* to a user's mental model of an application [17]. Furthermore, in a comprehensive study on the impact of personalized interfaces, Findalter and McGrenere found that "personalization can negatively impact the user's overall awareness of features" [10]. AppMap addresses this limitation, by instead providing an alterative visualization of the UI that can potentially *improve* a user's mental model of the interface.

2.3 Software Feature Visualizations

Before GUIs were popular, it was important to indicate a system's functionality to the user. For example, menu map visualizations were explored [24], and physical keyboard overlays[1] were heavily relied upon. We propose to bring back such strategies, but to make them more interactive and exhaustive.

Our work is also inspired by sitemaps, which provide alternative views to help aid navigation and cognition of online websites [5, 19, 23]. We adapt such representations to user interface components, which to our knowledge has never been explored. Another relevant visualization technique which served as inspiration are Mind maps [31]: structured diagrams used to represent words, ideas, tasks, or other items arranged using radial hierarchies and tree structures denoting relationships with a central governing concept. This structured approach has shown to have positive effects on learning [8]. AppMap applies these concepts to visualize the full scope of an application's user interface elements and the relations between those elements.

[1] http://www.vintage-computer.com/images/kaypro10keyboard.jpg (Retrieved 20/12/10).

Finally, while Software Visualization is an active subfield of software engineering [25, 30], the field focuses on a developer's perspective of the program's architecture and operation. Little attention has been given to visualizing the UI components of a software system for the benefit of an end user. In our work, we adapt the concept of software visualization to aid navigation and exploration of user interface components for end users.

2.4 Summary

To summarize, feature awareness and software bloat are still open and important problems in user interface design. The commonly proposed approach of adapting or personalizing interfaces may actually be detrimental to user's mental model of the applications features space. As such, we take inspiration from work on sitemaps and Mind Maps, and will introduce the concept of user interface visualizations to aid in awareness, discovery, and exploration of a GUI's feature space.

3 User Interface Visualizations

To motivate our work further, we provide an analogy to using maps while driving a car. Most drivers do not use a map if they know exactly where it is they are going, and how to get there. However, if a tourist has just rented a car in an unfamiliar city, and wants to explore, they would almost certainly invest in a map. Without a map, they may be unsure of where they are, and more importantly, where they have yet to explore. The experience of using a traditional GUI is similar to driving a car without a map. This may be acceptable if the user knows exactly what it is they want to do. However, if the user wishes to explore the interface, there are no analogous "map" tools for our "tourist" users. Using the traditional graphical interface itself as this "map" may significantly constrain the extent and value of the explorations. Instead, we propose interactive *user interface visualizations*, which serve as a platform for end-user software exploration.

The main idea is to display, in a single view, the entire scope of an application's user interface components, which the user can navigate and interact with. However, it is not our goal to make the user aware of every command in a system. On the contrary, most components of the UI may be irrelevant to any single user, based on their specific uses of the system [21]. Instead, we provide interactive tools and visualizations to help users identify components that may have particular relevance to them, amongst the scope of the entire system's functionality. This functionality could be useful to both novice and expert users, as both are susceptible to the awareness learnability challenge [14].

4 Design Space

Because of the lack of any previous comprehensive investigation into *user interface visualizations*, we map out a general design space by describing various properties worth considering.

4.1 Represented UI Elements

The design must primarily determine what user interface elements to include in the visualization. These elements could include individual tool icons, toolbars, menus, menu items, dialog boxes, tool parameters, etc. Also worth considering are tools or functions that do not have a visual representation in the existing user interface. For example, it may be worthwhile to have visual representations of the hotkey accelerators that are available for the system, as users are often unaware of these [13].

4.2 Element Granularity

The granularity of the UI elements must also be considered. For example, a coarse grain visualization could represent dialog boxes or menus, while a detailed visualization could depict individual elements within a dialog box, or individual menu items.

4.3 Element Visualization

Each element could be represented by an actual screenshot of itself, since users may be able to identify familiar dialog boxes by their look. Elements could also be visualized by representative icons or textual descriptions.

4.4 Layout

It has been argued that spatial layout can be one of the best methods to facilitate graphical perception [1]. There are numerous ways the individual components of a user interface visualization could be laid out. One option is to cluster the elements by usage or functional category. Another alternative is to have the layout reflect the UI elements' location in the primary UI or how deeply they are nested in the UI access hierarchy. Alternatively, the layout could have less semantic meaning, and be arranged in a table, indexed by name or some other ordered feature.

Within the layout, it may be worthwhile to include real-world landmarks, as this has been shown to improve spatial cognition [6, 26]. For our purposes, this may help users form a cognitive model of the user interface.

4.5 Sizing and Coloring

Sizing and coloring are both important features for graphical visualizations [4]. The size or color of the elements could be used to represent any quantitative data associated with the element, such as its frequency of use or number of child elements.

4.6 Interactivity

In its simplest form, the visualization could be a static representation. However, to best support exploration, the visualization should be interactive. Potential interactions include navigation, annotations, filtering, sorting, and grouping. Furthermore, the visualization could serve as a learning tool by linking individual elements to extensive help articles or tutorials, to help users explore and learn to use new features.

4.7 Application Link

While the visualization could run independently of the application, it is important to consider real-time communication between the application and the visualization. For example, the visualization could represent the user's current usage patterns, and highlight the tools that were used most recently. Alternatively, the user could trigger functionality from within the visualization.

5 Initial Prototypes And Insights

Through iterative design, we developed a series of mock-ups and prototypes in order to explore the design space discussed above. We briefly discuss these initial designs here, to help highlight some of the challenges in designing UI visualizations and to also provide rationale for our final design.

Our first exploration looked at visualizing the actual dialog boxes of an application using a TreeMap style layout [18]. Figure 4 shows a mock-up with purely simulated data of our design intent. Large regions of functionality would be visible (e.g., Modeling and Animation) and the relative size of each region could represent a corresponding proportion of functionality contained within the application. In addition, the size of each

dialog box could reflect the frequency of use. A problem with this design is that the layout conveys little semantic meaning to the user.

We next explored an experimental node-and-stick style visualization, which was manually created to show Microsoft's WordPad user interface elements (Figure 5). The elements are arranged according to the pull-down menu that they are accessed from. The prototype was interactive, allowing the user to pan and zoom the static image with the mouse.

From our initial prototypes we formed the following design guidelines:

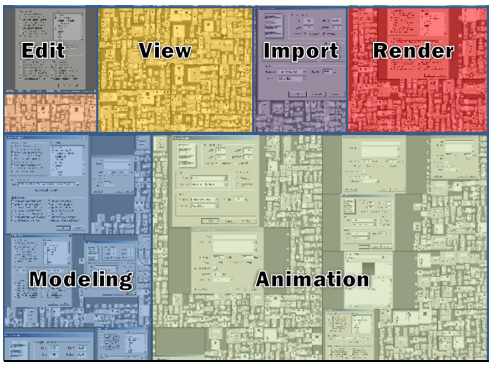

Figure 4: TreeMap mock-up using simulated data.

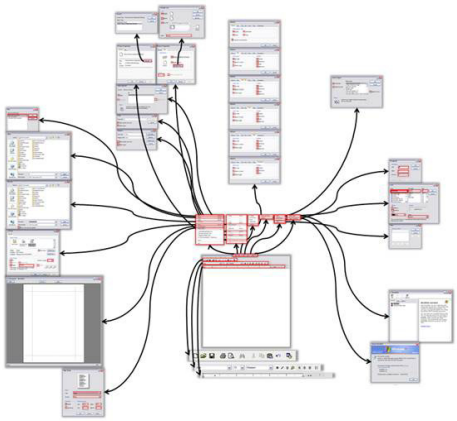

Figure 5: Wordpad expanded into all its visual UI elements. Highlighted regions indicate interactive areas of functionality.

Utilize a combination of manual and automated layout algorithms: One of the first issues we noticed was that it was surprisingly time consuming to manually layout the visual elements in a meaningful way – even for an application with limited functionality. Furthermore, we found it challenging to come up with a compact design. Even though the Wordpad node-and-stick prototype was carefully laid out, it still took up a lot of space. Switching to an entirely automatic layout could possibly make the visualization more compact, but it could also result in a less organized layout. We believe that a more useful approach may involve a combination of manual and automated layout algorithms. This would allow designers to specify high-level structures and then the algorithms could fill these structures with the UI elements.

Provide spatial landmarks: The panning and zooming interactions seemed very promising, and in our own experiences, were very useful for exploring the data. However, we found that in some cases there was very little relationship between neighboring UI elements in the visualizations. Thus, the user could become disoriented in the information landscape. As a result, we felt spatial landmarks could be considered.

Use uniformly sized iconic representations of the UI elements: With regards to element representation, we found that using screenshots of the actual elements added little value to the designs. They were sometimes useful for identifying known components, but this was not a primary goal for the visualization. Furthermore, the varying size and aspect ratio of these elements added to the challenge of creating compact and structured layouts. Thus, we felt it would be worthwhile to pursue uniformly sized iconic representations of the UI elements.

6 APPMAP

Based on the lessons learned from our initial prototypes, we built a more robust and interactive system. We called this system AppMap. We decided to use AutoCAD as a target application. AutoCAD is a complex program used for 2D and 3D design and drafting, containing over a thousand commands. Through the customization features of AutoCAD we were able to obtain graphical iconic representations of nearly all of its commands, and through AutoCAD's *Customer Involvement Program*, we were able to obtain associated command usage data. AppMap and AutoCAD ran in tandem, and could communicate with one another via a custom written AutoCAD plug-in.

6.1 Implementation

AppMap was programmed in C++ using OpenGL, specifically the WGL system interface. Implementation and evaluation occurred on a dual-monitor HP xw4600 workstation. AutoCAD ran on one monitor while AppMap ran on the other. This represents our envisioned usage of having the visualization on a secondary, or ambient, display. The AutoCAD plug-in was programmed in Microsoft's C# using the ObjectARX plug-in architecture.

6.2 Visualization Elements

For AppMap we decided to use commands as the primary visual component for the visualization, regardless of how they are accessed through the UI. This would give users a visualization of almost all actions that could be carried out with the system. In the dataset we collected, there were 1031 commands, with 840 of them having an associated icon (Figure 6a). For the remaining 191 commands, we used the same placeholder icon (Figure 6b).

Figure 6: a) Example iconic representations. b) Placeholder AppMap icon used for commands without associated icons.

6.3 Item Layout

AppMap supports two layouts: *map view,* and *grid view.*

6.3.1 Map View

Our initial prototypes indicated that the automatic layouts could be somewhat arbitrary, causing users to lose context when navigating, especially when zooming in. Motivated by research showing the benefits of spatial landmarks [6], we developed a map view, where icons are overlaid on a map of the world. This design is similar to the approach used in Robertson et al.'s Data Mountain, whose research showed beneficial recall and understanding when grounding icons with spatial landmarks [26].

The icons were grouped into 33 functional categories, and each group was automatically arranged over a specific region or country (Figure 7). We specified the region for each category heuristically, based on group size and relatedness. It is also conceptually possible to place groups according to size, popularity, etc. By default, icons are ordered alphabetically within each category.

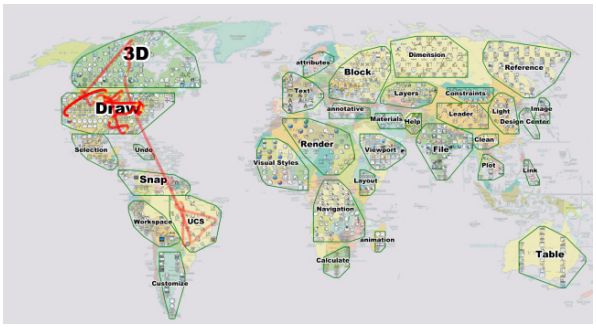

Figure 7: The Map View overlays icons on a world map. A "Flight path," or recent command history, is shown in red.

While the choice of using an atlas for the spatial grounding image provides a nice metaphor for the goals of the system, it is not a requirement. An alternative image could be used, but it should be familiar and contain explicit "areas" for the categorization. Similarly, there is no "correct" mapping between category and geographic location. Most important is that the categories have specific and constant locations to help users form a mental map of the system over time.

6.3.2 Grid View

In addition to the map view, we included a grid view layout structure (Figure 8), arranging icons in a square grid. This compact view is used for applying visualizations and sorting arrangements across all icons regardless of category, without restricting their placement to specific regions. By default, the icons are ordered alphabetically.

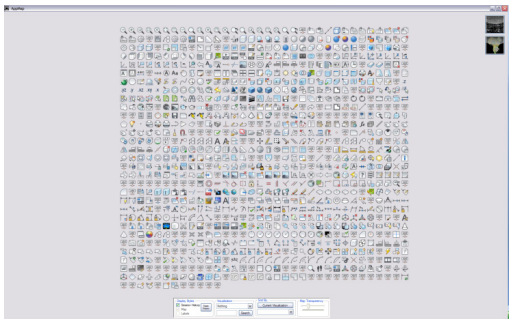

Figure 8: The Grid View organizes items into arrays of icons.

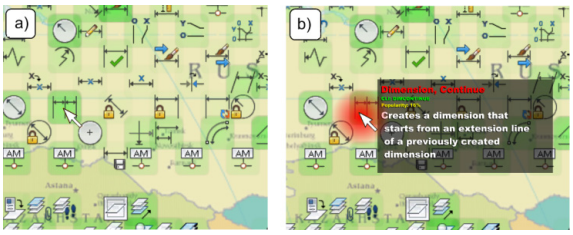

Figure 9: a) Zooming into the Text category of the map view. b) Displaying a tooltip for a command with a right-click.

6.4 Navigation and Interaction

AppMap uses the same zoom and pan interface as in the earlier prototypes. The mouse-wheel is used to zoom, and the left button is used to pan. This allows users to zoom in to specific regions to see detailed information (Figure 9a). In addition, right-clicking an icon invokes a contextual tooltip displaying additional information of the tool (Figure 9b). Double-clicking executes the command within AutoCAD.

6.5 Control Panel

The AppMap has a control panel (Figure 10) to allow the user to (1) set and adjust display styles, (2) control visual overlays, (3) sort the data given a selected criterion, (4) search for individual commands, and (5) adjust the transparency of the background map image. We now discuss each of these options in more detail.

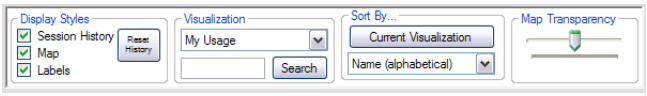

Figure 10: The AppMap control panel.

6.5.1 Display Styles and Map Transparency

A "Session History" checkbox controls the visibility of what we call "flight paths", which are lines between recently used commands (Figure 7). The data used to display this is the real-time command history that is collected by the AutoCAD plug-in. The purpose of these lines is to allow quick recognition of areas of the map view that have been "visited." This could help a user identify entire categories of functionality that they do not use. Alternatively, it may reveal to the user that they have used a command that exists in an entire category of related commands for which they were unaware. To avoid clutter, flight paths fade out over time, so that only 10 curves are ever visible at a time.

Additional controls include a "Map" checkbox, which shows or hides the background map image, a "Labels" checkbox, which controls the visibility of the category labels in the map view, and a slider, which can be used to adjust the map transparency.

6.5.2 Command Usage Visualizations

One of the potential benefits of user interface visualizations is indicating the relative importance of the commands to the user. This can be especially important for a system like AutoCAD, where only a certain subset of the entire feature set may be relevant to any single user. To this end, we acquired command usage histories gathered from AutoCAD's Customer Involvement Program (CIP) [21]. We used a data set of 4075 users, which contains a log of approximately 17,000,000 command activations. For the sake of prototyping, we labelled 17 of these users as "experts," since these particular users were known professionals in 2D drafting. We chose a single user's command stream to represent the command usage information of the AppMap user. In an actual implementation, this personal usage data could be real.

From this usage data, we defined four data sets to be associated with the commands, which could be visualized with a pull-down menu in the control panel:

Community Usage: The overall frequency, relative to all other commands, in which the command is used. For example, the "Delete" command makes up 17% of the entire user population's command stream. Command frequency is an important indication of the relative importance of a command [20, 21].

Community Popularity: The fraction of users who have used the command. For example, 95% of all users have used the *Erase*

command (Figure 11). This is an alternative indication of a command's importance.

Expert Usage: The frequency for which the command is used by the expert users. Viewing this information allows a user to see "what the experts are doing".

My Usage: The frequency which the AppMap user uses the command. This allows a user to see and reflect upon their own usage of the system.

Each data set is visualized by highlighting the icons with a colored box. The data value is mapped to the opacity of the color. A legend indicates the current visualization and the range in values (Figure 11a). Applying the visualization while zoomed out in the map view can allow users to obtain an overview of the relative importance of categories (Figure 11b). Zooming in would give detailed information on individual commands (Figure 9).

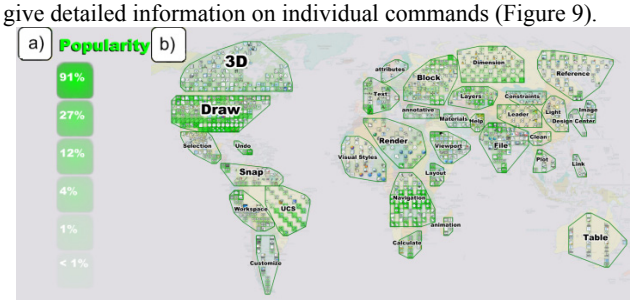

Figure 11: a) The legend for visualizing command popularity. b) The associated visualization, shown in Map View.

In addition to highlighting by a single data set, the drop down box contains options to visualize two usage data sets, comparing their differences. For example, the community usage can be compared to the expert's usage, to quickly view what commands experts use more often than the community on average, and vice versa (Figure 12a).

6.5.3 Release Version Visualization

We also included a data field that identifies what release year a given AutoCAD command was introduced. The 13 discrete release versions are illustrated by 13 different colors (Figure 12b). Viewing this information could help a user identify commands, which were recently released, that they have not yet adopted.

6.5.4 Sorting

The spatial arrangement of items is of great importance to cognition [1]. In addition to highlighting icons, their locations can be sorted. This can allow users to obtain additional information. For example, if a user is interested in what commands are popular in AutoCAD, he or she may select the community popularity visualization and judge which commands are popular by inspection. However, if the user is interested in the single, most popular command, this method is not effective. To do this more efficiently, the user can sort the commands according to the desired data field (Figure 12a). When the sort is performed, icons are sorted from top-to-bottom, left-to-right. If in map view, each category is sorted within its region.

A powerful feature in AppMap is the ability to visualize the commands by one data set and sort by the other. This can be a fruitful feature for understanding certain relationships between data sets. For example, Figure 12b shows the grid view where the icons are highlighted by release version and sorted by popularity. Figure 13 shows a zoomed in view of the grid view when sorted by community popularity, and viewed by "my usage." This arrangement could allow a user to identify commands to adopt.

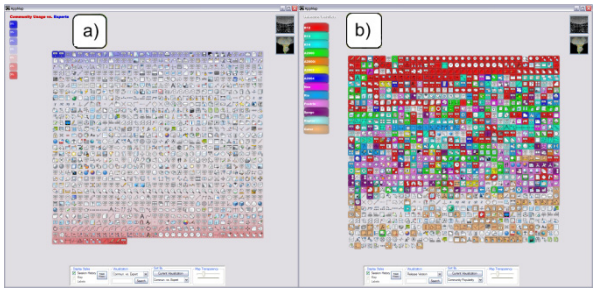

Figure 12: The Grid View, a) visualized and sorted by the comparison between expert and community usage. b) visualized by release version, and sorted by popularity.

Figure 13: The Grid View, zoomed into the top-left. The view is sorted by community popularity, and visualized by, "My Usage."

Sorting criteria are selected from a pull down menu. Alternatively, the user may click the "Current Visualization" button, to sort the commands by the current visualization setting (Figure 12a).

6.5.5 Searching and Executing Commands

The search box can be used to find commands within the visualization. As each key is typed, AppMap searches for commands that have the current substring in either their name or tooltip text. Icons not containing this text fade out, to emphasize the matching commands. Clicking "Search" will search for an exact command name match. If the command is found, the icon is highlighted, and AppMap zooms the view onto that icon. Once a command is found, the user can double-click it to activate its function within AutoCAD. Double clicking the icon also highlights commands that it is often followed by, as determined from our collected command usage data (see Figure 3).

7 Initial User Observation Sessions

In order to understand how users may approach and use a UI visualization, we conducted a set of exploratory user observation sessions of AppMap, used in conjunction with AutoCAD (Figure 2). This study was *not* meant to measure a formal or quantitative benefit of AppMap. We feel the true value of AppMap would occur after a prolonged exposure, allowing time for a user's mental model to develop. It would thus be difficult to measure this in our initial observation sessions. Instead, we felt at this stage, it would be more useful to gather initial impressions on the design aspects and features of the system.

Participants performed a series of AppMap tasks, using both the grid and map views, while a single observer sat nearby. Participants were then given a short questionnaire to ask about their experiences with the system.

The sessions were conducted with six participants (5 male, 1 female) aged 24-25. Three participants were "novice" users,

defined as users with less than one year experience, and three were "expert" users, who had been using AutoCAD for more than three years. Each session lasted approximately one hour.

The session started with a 5-minute introduction to the system. Participants were then given a list of eleven basic tasks, which exhausted the majority of the AppMap functionality. For the observations sessions, the *My Usage* data set was mocked-up with data not belonging to the participant. Two example tasks are as follows:

- *Give three commands that experts use more often than community users.*
- *What is the exact popularity of the VPMAX tool?*

After completion of these tasks, two AutoCAD tasks were carried out, with the "flight path" being visualized in real time on the AppMap.

7.1 Observations and Results

The observations made during the usage sessions were encouraging. In general, participants were able to quickly learn and understand how to use the various features of the system. Of the 66 tasks performed across the six participants, 56 were completed quickly and independently by the participants. In the other 10 tasks, minor help had to be provided by the experimenter.

The questionnaire demonstrated that users enjoyed using AppMap (6.0/7), and found AppMap easy to use (5.2/7). In addition, users indicated that they would likely recommend AppMap to friends or colleagues that were both novice users (5.8/7) and expert users (5.7/7). Participants' comments during execution of AppMap tasks indicate that they reacted positively to the visualization (5.8/7), search (5.7/7), and navigation (5.3/7) aspects of AppMap. During each session, participants were never observed as being disoriented or confused.

The post-session questionnaire indicated that the most popular features was the *command usage visualization.* This indicates users had a positive reaction to the type of information that AppMap exposed. Informal comments made during the study also indicate users were genuinely interested in this information. We are encouraged that this type of data is welcomed, and in particular, in the form we presented it in.

One expert was surprised by the number of commands immediately after beginning to use AppMap for the first time, and believed that a lot of these new commands would be useful to him. This was achieved by providing a single, non-hierarchal, compact, graphical view of all available commands in the application. Another expert took particular interest in viewing the icons shaded by release version, taking several minutes to explore this visualization closely.

A feature that seemed to be less popular was the real time visualization of "flight paths" while using AutoCAD. Participants did appreciate the ability to review this information once their AutoCAD tasks were complete, but they rarely viewed the flight paths as they appeared during the AutoCAD task completion.

Participants provided positive remarks regarding both the grid view and the map view. Participants felt the grid view was an effective way to visualize the data, but also liked the persistent spatial grounding that the map view provided.

7.2 Discussion

We felt the informal usage observation sessions would be a more appropriate form of evaluation than a controlled laboratory study, given the exploratory nature of this research. This allowed us to observe some important initial experiences. The initial observations indicate our tool is promising, but additional long-term studies, and formal methods, would need to be used to quantitatively measure the benefits.

Having an alternative user interface view, or supporting dual user interfaces, seems promising, as the user can choose a presentation based on intent – one oriented for quick command access and a second for function exploration and awareness. This may ultimately lead to improved command awareness, increased understanding, confidence or performance with the system.

Finally, it is important to mention that awareness and exploration is only one aspect of software learnability, other important challenges, such as task knowledge and novice to expert transition [14] are equally significant. We foresee AppMap complementing, not replacing, existing help mechanisms that tend to other aspects of software learnability.

8 Potential Benefits Of UI Visualizations

Through our investigation, prototypes and user observation sessions, we have been able to identify numerous potential benefits for a user interface visualization. These relate to the characteristic information visualization tasks suggested by researchers in the field of information visualization [2, 29]. Examples include providing overviews of a collection, zooming in on areas of interest, filtering out less relevant items, acquiring addition details for items when desired, highlighting relationships among items, and providing interactive histories. Below we identify potential benefits and discuss how these tasks specifically relate to user interface visualizations.

Exploring Features: A user interface visualization would allow a user to explore the UI features of a target application. Exploring the visualizations could amplify a user's cognition of the user interface [2]. This may improve the confidence of a novice user, and motivate an expert user to learn new features.

Locating Features: A known problem related to software learnability is locating functionality [14]. If features of the application are logically structured within the visualization, users may be able to locate specific components.

*Understanding Feature Relations***:** Through appropriate interactive visualization, users could better understand how features relate to one another [29], such as tools which accomplish similar goals, or tools that are often used together.

Discovery of Features: Exploring a user interface visualization could result in the discovery of new features which the user was unaware of [27]. This is especially important given that awareness is a recognized barrier to software learning [14], and in particular can prevent a user's transition to expert usage.

Usage Reflection: Exploring the features of a system may also give users the opportunity to reflect on how they use the program, how much of the program functionality they have used and how their usage might relate to other users. Provided with the right information, users may be able to identify specific tools or whole categories of features that they should learn.

Communication Tool: Information visualizations can be a useful platform for collaboration and communication [15]. In particular, if the visualization tool has a static view that is shared among users, it could serve as a platform for communication. Users could informally refer to areas of the visualization to direct one another to tools or categories of interest.

Comparison Tool: Similar visualizations of two or more applications can be used to compare the relative functionality of each application. This potentially would allow users to more easily transition from one well-known application to similar applications, to support "subsequent learning" [7].

9 Future Work & Conclusions

There are many areas for future investigation. One promising direction we see is coordinating user interface visualizations with other help and search techniques. AppMap currently provides tooltips with short descriptions of each command. This could be extended to link commands to help articles or learning videos related to that command, or even community comments about the nature or usefulness of that feature. This would help users not only discover new tools, but also help user learn how to use those new tools.

Another promising direction is to consider how our command-centred visualizations relate to a goal-oriented user. Our intended use was not to support the completion of specific goals, but to instead support informal exploration and discovery. However, it may be useful to segment the visualization based on a user's common goals or tasks, rather than on the categorizations which we used. Here, it is important to remember that learning commands is only one aspect of software learnability.

It would also be interesting to consider other usage domains. While our focus was to assist end-users, our experiences indicate that this may not be the only target user. For usability engineers, AppMap may give a fast, visual look at traditional usability data. AppMap could also be used as a tool for interface designers to iterate on a UI based on how it is being used by its customers.

Overall, we would like to continue to explore alternative representations and structures to present an application's user interface to the user to facilitate application examination and understanding. Both our design space, and the lessons learned from our prototypes, should not be considered exhaustive. We hope our work will inspire future efforts to develop upon these contributions.

In conclusion, we have defined and explored the concept of *user interface visualizations*. Through the development of multiple prototypes and AppMap, we illustrated novel methods of viewing, accessing and analyzing a computer program's functionality. We have been able to identify and present both a design space and potential benefits for developing UI visualizations. Our experiences indicate that user interface visualization is a fruitful area for continued explorations. We are encouraged by the results of our initial user observations, and hope our work will inspire future developments in user interface visualizations.

10 References

[1] Bertin, J. (1967). *Semiologie graphique : Les diagrammes - Les reseaux - Les cartes, Les reimpressions*. Editions de l'Ecole des Hautes Etudes en Sciences. 1967.

[2] Card, S. K., Mackinlay, J. D. and Shneiderman, B., *Readings in information visualization: using vision to think*. 1999: Morgan Kaufmann Publishers Inc. 686.

[3] Carroll, J. M. and Carrithers, C. (1984). Training wheels in a user interface. *Commun. ACM*. 27(8):800-806.

[4] Cleveland, W. S. and McGill, R. (1984). Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods. *Journal of the American Statistical Association*. 79(387):531-553.

[5] Danielson, D. R. (2002). Web navigation and the behavioral effects of constantly visible site maps. *Interacting with Computers*. 14(5):601-618.

[6] Darken, R. P. and Sibert, J. L. (1996). Navigating large virtual spaces. *Int. J. Hum.-Comput. Interact.* 8(1):49-71.

[7] Davis, S. and Wiedenbeck, S. (1998). The effect of interaction style and training method on end user learning of software packages *Interacting with Computers*. 11(2):147-172.

[8] Farrand, P., Hussain, F. and Hennessy, E. (2002). The efficacy of the mind map study technique. *Medical Education*. 36(5):426-431.

[9] Findlater, L. and McGrenere, J. (2007). Evaluating reduced-functionality interfaces according to feature findability and awareness. *INTERACT*. 592-605.

[10] Findlater, L. and McGrenere, J. (2010). Beyond performance: Feature awareness in personalized interfaces. *Int. J. Hum.-Comput. Stud.* 68(3):121-137.

[11] Gajos, K. Z., Czerwinski, M., Tan, D. S. and Weld, D. S. (2006). Exploring the design space for adaptive graphical user interfaces. *AVI*. 201-208.

[12] Greenberg, S. and Witten, I. H. (1985). Adaptive personalized interfaces: A question of viability. *Behaviour and Information Technology*. 4(1):31-45.

[13] Grossman, T., Dragicevic, P. and Balakrishnan, R. (2007). Strategies for accelerating on-line learning of hotkeys. *ACM CHI*. 1591-1600.

[14] Grossman, T., Fitzmaurice, G. and Attar, R. (2009). A Survey of Software Learnability: Metrics, Methodologies and Guidelines. *ACM CHI*. 649-658.

[15] Heer, J., Vigas and Wattenberg, M. (2007). Voyagers and voyeurs: supporting asynchronous collaborative information visualization. *ACM CHI*. 1029-1038,

[16] Hsi, I. and Potts, C. (2000). Studying the Evolution and Enhancement of Software Features. *International Conf. on Software Maintenance (ICSM'00)*. 143-151.

[17] Hui, B., Partridge, G. and Boutilier, C. (2009). A probabilistic mental model for estimating disruption. *Proceedings of Intelligent user interfaces*. 287-296.

[18] Johnson, B. and Shneiderman, B. (1991). Tree-Maps: a space-filling approach to the visualization of hierarchical information structures. *IEEE Visualization*. 284-291.

[19] Lin, J., Newman, M. W., Hong, J. I. and Landay, J. A. (2000). DENIM: finding a tighter fit between tools and practice for web site design. *ACM CHI*. 510-517.

[20] Linton, F., Joy, D., Schaefer, H. P. and Charron, A. (2000). OWL: A recommender system for organization-wide learning. *Educational Technology & Society*. 3(1):62-76.

[21] Matejka, J., Li, W., Grossman, T. and Fitzmaurice, G. (2009). CommunityCommands: command recom-mendations for software applications. *ACM UIST*. 193-202.

[22] McGrenere, J. and Moore, G. (2000). Are We All In the Same "Bloat"? *Graphics Interface*. 187-196.

[23] Newman, M. W. and Landay, J. A. (2000). Sitemaps, storyboards, and specifications: a sketch of Web site design practice. *ACM conference on Designing interactive systems*. 263-274.

[24] Parton, D., Huffman, K., Pridgen, P., Norman, K. and Shneiderman, B. (1985). Learning a menu selection tree: Training methods compared. *Behaviour & Information Technology*. 4(2):81-91.

[25] Price, B. A., Baecker, R. M. and Small, I. S. (1993). A Principled Taxonomy of Software Visualization. *Jour. of Visual Languages and Computing*. 4(3):211-266.

[26] Robertson, G., Czerwinski, M., Larson, K., Robbins, D., Thiel, D. and Dantzich, M. v. (1998). Data mountain: Using spatial memory for document management. *ACM UIST*. 153-162.

[27] Russell, D. M., Stefik, M. J., Pirolli, P. and Card, S. K. (1993). The cost structure of sensemaking. *ACM CHI*. 269-276.

[28] Shneiderman, B. (1983). Direct Manipulation: A Step Beyond Programming Languages. *Computer*. 16(8):57-69.

[29] Shneiderman, B. (1996). The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. *IEEE Symposium on Visual Languages*. 336 - 343.

[30] Stasko, J. *Software visualization: programming as a multimedia experience*, T.M. Press. 1998.

[31] Wycoff, J., *Mindmapping: Your personal guide to exploring creativity and problem solving*. 1991, New York, New York: Berkeley Books.