Learnability and Personalization

Peter Beshai

March 11, 2013

Introduction

When designing and evaluating user interfaces, the learnability of the interface needs to be taken into consideration. Due to the lack of a consensus definition of learnability, Grossman et al. tackled the problem of surveying the literature to find out all the ways the term was being used and consequently produced a taxonomy of learnability to be used in future work [6]. They make a clear distinction between two different types of learnability: initial and extended. Initial learnability has to do with users learning how to do tasks for the first time, while extended learnability is about the learning that takes place as a user gains expertise with a system.

With this understanding of learnability, we investigate how a variety of systems have made use of personalization techniques to facilitate both initial and extended learning. Three different ways of supporting learnability are discussed: reduced-functionality interfaces, context-aware help, and command recommendations.

Reduced-Functionality Interfaces

A reduced-functionality interface is one where features have been stripped out for the purposes of simplification. In this section, we will examine one method in particular: the use of Multi-Layered (ML) interfaces, where users can progress from a basic version of the interface into the full-featured version over time. Motivations behind such interfaces are to adapt to a user's cognitive limits, such as working memory capacity, as well as to make common tasks easier for novices to learn [7].

Inspired by previous work that showed significant improvements to learnability for desktop software [1, 2, 4], Leung et al. investigated the effectiveness of using an ML interface for a mobile software application designed for older users [7]. They discovered that while novices were faster and less-error prone when learning basic tasks in the reduced-functionality layer, they did not experience any positive transfer effects when switching to the full-functionality layer, contrary to the results of [2]. The authors speculate that this may be due to the difficulty their older participants had in producing mental models of how the menus worked. Another possible source for the discrepancy was the way layers were reduced. The systems used in [2] had the advanced functions blocked in the reduced layer, while in [7] they were completely hidden. The authors suggest as future work an investigation of the impact on extended learnability caused by the method of feature reduction, as well as research into which tasks benefit most from ML interfaces.

While there were no positive transfer effects for learnability, there were also no negative effects, and many of the users in [7] were happy solely using the features in the reduced inter-

face. This is important because the improved initial learnability resulted in a less frustrating interface, a highly determining factor in older user adoption rates of new technologies.

However, an important flaw with reduced-functionality interfaces is that by removing functionality, awareness of other features can be negatively affected [4, 5], compromising extended learnability. The performance gains in studies looking at the improved initial learnability of an interface often overlook this component, so to combat this, Findlater and McGrenere have devised metrics to measure awareness distinct from feature findability [4].

Context-aware Help

Beyond simply finding features quickly and completing tasks with minimal errors, another aspect of learnability is the ease with which users can find solutions to problems and questions they have with the system: learning by reading help documentation. It was only in 2011 that work began being done on how help systems can be improved to better support users by taking advantage of contextual information [10, 3]. Prior to this work, help systems were either built-in, typically offering descriptions of the features in the application, or were found on the web through discussion boards, which often have richer answers than the in-software help, but lack contextual information, putting the onus on the user to construct accurate queries to find relevant information. This becomes especially difficult for users who are new to an application and unfamiliar with how to describe their context.

The work of Matejka et al. involved bringing the richness of web-based answers into the software itself [10]. They did this through their system IP-QAT (In-Product Questions, Answers, and Tips), a context-aware help panel in AutoCAD (Fig. 1). While most discussion boards are organized by date of last modification, IP-QAT extends this idea by including the user's recent activity and help browsing history to produce a list of ten potentially useful topics. The main benefits of this approach lie in the integration with the software: the help topics, visible within the main program, are passively updated based on what the user is doing. The push nature of this system allows for opportunistic learning, in that a user may notice an interesting topic that they would not have otherwise searched for.

In contrast, the work by Ekstrand et al. involved integrating the software context directly into a web browser to aid users in finding relevant answers to their problems [3] (Fig. 2). The solution tracks user behaviour along with the application's context (e.g., opened dialogs, document contents, recent commands) in order to augment web search queries and results. The user study showed mixed results, with many users feeling confused by the interface due in part to the lack of transparency of the queries being made.

Beyond the user confusion shown in [3], both systems had problems related to privacy

and confidentiality. Automatically including contextual information in public forums may violate company policies, and it's not clear whether users are aware of what information is being shared. The two systems also both rely heavily on having a diverse amount of quality content in the help databases they are indexing. By adding contextual keywords to search queries in [3], the number of results returned is reduced, requiring well described, good quality results for success. This is less of a problem in [10], as there was a 47% increase in the amount of voluntary posting compared to their existing discussion board.

Future work in this domain includes the embedding of machine-readable contextual information in help documents, allowing fast and accurate retrieval. For example, if a tutorial video is created, along with the video could be information about which menus and dialogs were activated that could then be searched by the help system. There is also room for investigating how collaborative filtering can be applied to these systems, extending them beyond the simplicity of recently used commands to determine the results. This would involve comparing the context and history of the current user against a larger community and using what similar users found to be helpful when determining the list of topics to share.

Command Recommendations

The final area we will discuss is about systems that recommend new commands to users to facilitate the development of expertise. In particular, these systems make use of personalization techniques to provide relevant recommendations, based off of the user's history. Linton et al.'s OWL (Organization-Wide Learning, 2000) was the first of such systems [8]. They collected Microsoft Word usage data of all employees across a company and compared individuals against the whole. If a user under-used a command compared to the organization's average, a recommendation would be made. The major oversight with this project was the assumption that all users across the organization should fit to the same usage patterns. Subsequent work progressed along this vein, producing more personalized results [9, 11].

CommunityCommands (CC), the first related work to follow OWL, leveraged collaborative filtering techniques to produce new, more personalized recommendation algorithms [9] (Fig. 3). The system worked by comparing the command history of a single user against the histories of thousands of other users. The results are filtered based on other users with similar usage histories who also use additional commands (user-based filtering), or based on commands that are similar to the active user's history (item-based filtering). Both approaches produced significantly more good recommendations than the algorithm used in [8].

Murphy-Hill et al. investigated augmenting the algorithms developed for CC with the addition of command discovery information [11]. By modelling how users discover new com-

mands historically, relevant recommendations can be made corresponding to the order in which commands are discovered. They propose four new algorithms, all of which are less effective than the methods used in previous work when dealing with novice users. With experts, however, their collaborative filtering based on discovery modelling works well, producing the best balance of useful to not useful recommendations. This suggests that a hybrid approach may be necessary, as novices do not have large enough discovery histories to make the collaborative filtering effective.

A major issue in this area is the resistance existing users have in accepting recommendations, as they are content with their current usage patterns. As such, it is important that recommendations take place soon after introduction to the system to set up a positive track record that encourages use after expertise is acquired. Another issue is that temporality of use is not explored by current systems, introducing a problem where a command is recommended and used once, and then forgotten about by the user. The command is now in the user's history, and consequently will not be recommended again.

Future work remains to be done exploring the use of temporality to support forgetfulness, as well as to aid in the modelling of sequences. The prospect of recommending new sequences of commands to replace or augment existing behaviour seems promising and is something that needs to be researched further. There is also work to be done investigating different weighting schemes for the recommendation algorithms, including adding new information about adoption rates and user preferences (e.g., preferring commands that are used by people you know). Finally, inefficiency-based recommendations, where a series of commands can be replaced by a single one, are currently unexplored by the automated systems. They typically require an expert to setup the models, but it may be worth investigating how this can be done automatically.

Conclusion

We have discussed three areas where personalization has been used to aid in improving learnability in systems: reduced-functionality interfaces, context-aware help, and command recommendations. Each area has its own advantages, and much of the research has only been done in recent years, leaving significant room for future work. The systems involving collaborative filtering seem to be the most promising, but they are still immature. There is currently a need to validate the existing research with longitudinal studies that provide insight into how these techniques truly affect people over an extended period of time. All users are different, and with the complex nature of learnability, personalizing our systems to fit individual needs is an excellent approach.

References

[1] J. M. Carroll and C. Carrithers. Training Wheels in a User Interface. *Communications of the ACM*, 27(8):800–806, 1984.
An early attempt at using a reduced-functionality interface to reduce errors by novices.

[2] R. Catrambone and J. Carroll. Learning a word processing system with training wheels and guided exploration. In *CHI '87 Proceedings of the SIGCHI/GI Conference on Human Factors in Computing Systems and Graphics*, pages 169–174, 1986. ISBN 0897912136.
Extends the results of Training Wheels to show positive transfer effects when users migrate to the full featured interface.

[3] M. Ekstrand, W. Li, and T. Grossman. Searching for software learning resources using application context. In *UIST 2011 Conference Proceedings: ACM Symposium on User Interface Software & Technology*, pages 195–204, 2011. ISBN 9781450307161.
Describes a system for integrating application context into web search to aid user's in finding relevant help documents. Also includes a context taxonomy.

[4] L. Findlater and J. McGrenere. Evaluating reduced-functionality interfaces according to feature findability and awareness. *Human-Computer Interaction - INTERACT 2007*, pages 592–605, 2007.
Breaks down performance into two metrics, findability and awareness, and runs a study demonstrating their strengths and weaknesses.

[5] L. Findlater and J. McGrenere. Beyond performance: Feature awareness in personalized interfaces. *International Journal of Human-Computer Studies*, 68(3):121–137, Mar. 2010. ISSN 10715819. doi: 10.1016/j.ijhcs.2009.10.002.
Demonstrates the importance of measuring awareness when evaluating personalized interfaces by showing user's have negatively impacted performance on new tasks.

[6] T. Grossman, G. Fitzmaurice, and R. Attar. A survey of software learnability: metrics, methodologies and guidelines. In *UIST 2009 Conference Proceedings: ACM Symposium on User Interface Software & Technology*, 2009.
Constructs a taxonomy of learnability and gives evidence supporting a new evaluation technique: question-suggestion.

[7] R. Leung, L. Findlater, and J. McGrenere. Multi-layered interfaces to improve older adults' initial learnability of mobile applications. *ACM Transactions on Accessible Com-*

puting, 3(1):1–30, 2010. doi: 10.1145/1838562.1838563.http.
Tests the effectiveness of a reduced-functionality interface on a mobile device with older users, discovering positive initial learning effects, but no transfer effects when switching to the full-featured interface.

[8] F. Linton, D. Joy, H. Schaefer, and A. Charron. Owl: A recommender system for organization-wide learning. *Educational Technology & Society*, pages 65–69, 2000.
Describes a year-long study of a system used to recommend commands to users based on their usage compared to the average across the organization.

[9] J. Matejka and W. Li. CommunityCommands: command recommendations for software applications. In *UIST 2009 Conference Proceedings: ACM Symposium on User Interface Software & Technology*, pages 193–202, 2009. ISBN 9781605587455.
Describes a system for recommending new commands to users based off of collaborative filtering, effectively doubling the number of good suggestions made over previous work.

[10] J. Matejka, T. Grossman, and G. Fitzmaurice. IP-QAT: in-product questions, answers, & tips. In *UIST 2011 Conference Proceedings: ACM Symposium on User Interface Software & Technology*, pages 175–184, 2011. ISBN 9781450307161.
Describes an integrated help system that makes use of the user's current context to suggest relevant topics.

[11] E. Murphy-Hill, R. Jiresal, and G. Murphy. Improving Software Developers' Fluency by Recommending Development Environment Commands. In *SIGSOFT 2012/FSE-20 Conference Proceedings: ACM International Symposium on the Foundations of Software Engineering*, pages 1–11, 2012. ISBN 9781450316149.
Compares eight algorithms for recommending commands to users, with emphasis on using command discovery as input. Results show a hybrid between previous work and the new algorithms is ideal.

Appendix: Figures

Figure 1: IP-QAT panel displaying tooltip with an image on mouseover

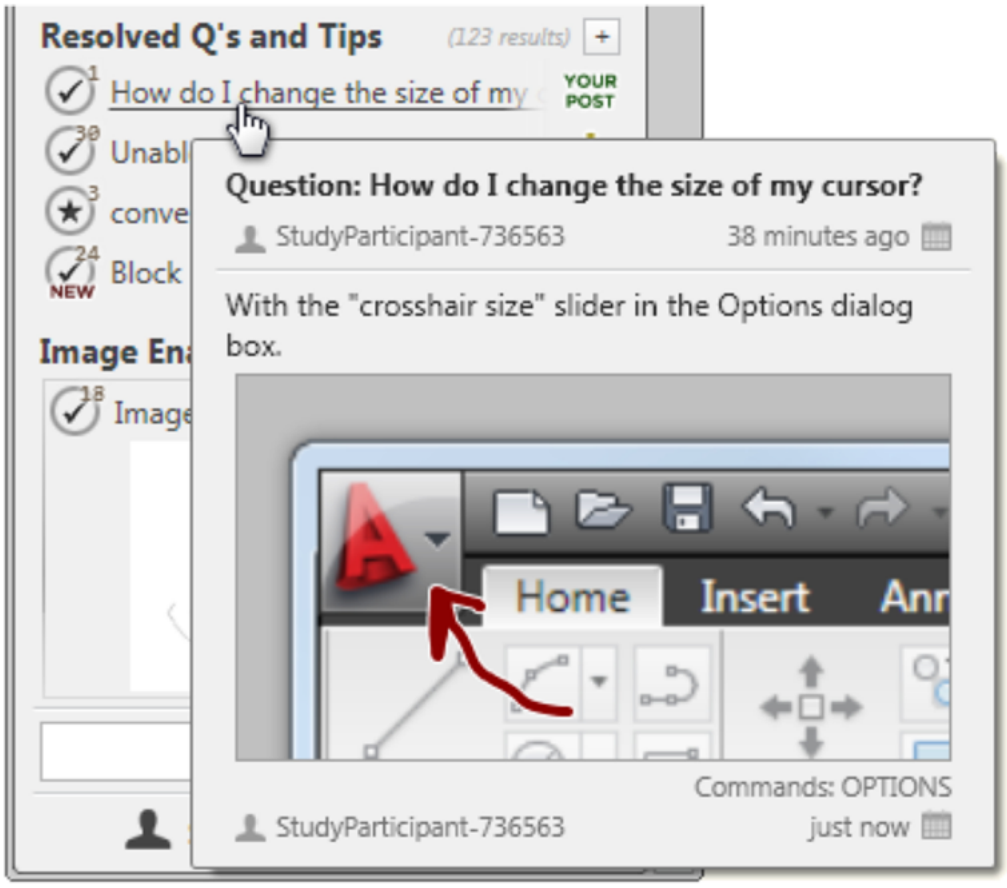

Figure 2: Context augmented web search results

Figure 3: CommunityCommands panel listing recommended commands with notes