Evaluation of a Role-Based Approach for Customizing a Complex Development Environment

Leah Findlater[1] **Joanna McGrenere**[1] **David Modjeska**[2]
[1]Department of Computer Science
University of British Columbia, Canada
{lkf, joanna}@cs.ubc.ca

[2]Interactive Media Lab
University of Toronto, Canada
modjeska@acm.org

ABSTRACT
Coarse-grained approaches to customization allow the user to enable or disable groups of features at once, rather than individual features. While this may reduce the complexity of customization and encourage more users to customize, the research challenges of designing such approaches have not been fully explored. To address this limitation, we conducted an interview study with 14 professional software developers who use an integrated development environment that provides a role-based, coarse-grained approach to customization. We identify challenges of designing coarse-grained customization models, including issues of functionality partitioning, presentation, and individual differences. These findings highlight potentially critical design choices, and provide direction for future work.

Author Keywords
Role-based interface, adaptable and adaptive interfaces, customization, interview study.

ACM Classification Keywords
H.5.2 User Interfaces: Evaluation/methodology, graphical user interfaces.

INTRODUCTION
Complex software applications often provide more features than are used even by expert individual users [5,8]. To manage this complexity, customization methods to reduce functionality have been proposed by several researchers, either for regular usage or for a limited training period. Evaluations have been limited in number and scope, but have shown that reduced-functionality applications can make novice users faster, more accurate and more satisfied [1], and that they can be preferred by a large proportion of intermediate and advanced users [7]. Despite these advances, evaluations have focused on the benefits of such designs, while drawbacks have largely been ignored.

CHI 2008, April 5–10, 2008, Florence, Italy.

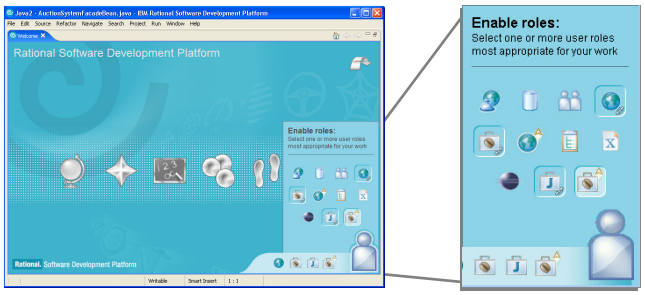

Figure 1. Screenshot of RAD's mechanism to change user role. A short description is presented for each role on mouseover.

In particular, research on coarse-grained approaches to reducing functionality, such as layered interfaces [10], has been limited to relatively simple applications or customization models [2,3,9,10]. A coarse-grained approach allows large groups of features to be enabled or disabled at once; in contrast, a fine-grained approach enables or disables individual features, as is done with Microsoft Office 2003's adaptive menus, or with multiple interfaces [7]. Since lack of time and difficulty are among the factors that inhibit customization [6], coarse-grained approaches have the potential to provide the benefits of customization while reducing the burden on the user. However, due to the lack of evaluation of such approaches, we do not fully understand their effectiveness.

The role-based customization model found in IBM Rational Application Developer 6.0 (RAD) is an example of a coarse-grained approach for a complex, feature-rich application. This approach, shown in Figure 1, allows the user to select from a set of user roles, such as *Java Developer* and *Web Developer*, and only functionality associated with those roles is enabled in the user interface. Although CSCW applications have on occasion provided user roles to support collaboration, the research literature does not contain examples of using roles to filter functionality in complex user interfaces. An additional difference is that RAD's customization model offers flexibility through multiple levels of granularity, unlike the restrictive definitions of roles that have been found to be problematic in CSCW [4,11].

To address the limitations discussed above, we conducted an interview study with 14 users of RAD. The findings highlight challenges of coarse-grained approaches,

including partitioning of features, presentation, and individual differences. These issues should be considered by designers of reduced-functionality systems, and offer potentially fruitful areas for further research.

IBM RATIONAL APPLICATION DEVELOPER

RAD extends and inherits all user interface components from Eclipse, a popular IDE (http://www.eclipse.org). Shown in Figure 2, the key components of RAD are as follows:

Workspaces hold one or more development projects. Users can create more than one workspace, but can only work in a single workspace at a time. Customization changes are only persistent within a workspace.

Perspectives group functionality by task (e.g., *Debug Perspective*). The user controls which menu and toolbar items as well as views on the code appear in a perspective, and can also control switching between perspectives. There is often functionality overlap between perspectives.

Capabilities are groups of features that correspond to user tasks on a higher level than perspectives. The features associated with a capability can range from entire perspectives to individual menu and toolbar items within a perspective. When a capability is disabled, the features associated with it are no longer visible. For example, enabling the *Java Development* capability enables features for creating and testing Java projects, such as a Java-specific text editor, and a menu item to create a new class.

Roles are groups of capabilities that are potentially overlapping. RAD provides 11 roles on a *Welcome* screen when the user creates a new workspace. By default, 2 roles are enabled (*Java Developer* and *Web Developer*), but the user can disable these and/or enable additional roles. When the user enables a role, this enables the set of capabilities associated with that role; in turn, the specific interface elements associated with those capabilities are made available in the interface. For example, enabling the *Tester* role will enable 3 capabilities: *Core Testing Support*, *Probekit*, and *Profiling and Logging*.

Roles determine a base set of functionality to include in the interface, and, as the user works, additional functionality can be exposed or hidden by manipulating capabilities. This can be done both manually, through a user preference dialog that lists all available capabilities, or automatically, through trigger points in the interface. Trigger points offer a small amount of adaptive prompting in an otherwise adaptable (user-controlled) customization model: for example, when creating a new project the user can choose to "show all" types of possible projects; if the new project is associated with a disabled capability, the system will prompt the user to enable that capability.

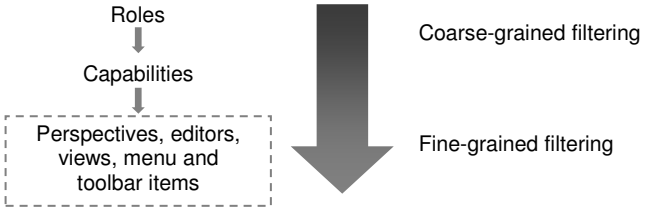

Figure 2. Customization mechanisms in RAD. Specific settings at each level are associated with a workspace.

INTERVIEW METHODOLOGY

Each interview was 1 hour long, with 32 semi-structured questions to understand use of roles and capabilities, and overall customization practice. At the end of the interview there was a debriefing and unstructured discussion period on managing user interface complexity. All interviews were conducted by the same researcher and were recorded, transcribed and coded for analysis. Since the interviews were exploratory, we did not set hypotheses beforehand. Instead, we used an open coding technique to develop categories of data [12]. This iterative process allowed us to identify emergent themes, and confirmed some of the focus areas of our investigation. We separated pure usability issues from what we consider to be the more generalizable benefits and challenges of reducing functionality. Almost all questions were open ended and participants were encouraged to speak freely, so the number of people who mentioned a point should be considered a minimum.

Through developer mailing lists and word of mouth, we recruited and interviewed 14 professional software developers (11 male, 3 female). They had between 2 and 30 years of software development experience (M = 11, SD = 9) and reported spending over 30 hours per week using an Eclipse-based development platform (SD = 13). Experience with RAD varied, ranging from less than a month for 3 participants to 12 months for another participant (M = 4.1, SD = 3.2). This was representative of the user base, since RAD had only been released 6 months before we conducted the study; the participant with 12 months of experience had initially used a pre-release version. Participants reported using RAD to develop a variety of applications, including: Web (7 participants), J2EE (4 participants), Java or plug-ins for Eclipse (6 participants), and database (1 participant). Three participants used Eclipse as their primary IDE, rather than RAD, and some questions were not asked of these users (noted when applicable in the next section).

FINDINGS

We first briefly discuss overall customization practice to provide context for the findings on roles and capabilities.

Overall Customization Practice

RAD provides 11 perspectives by default, though users can increase this by saving custom perspectives and installing additional plug-ins. On average, participants made use of 4 to 5 perspectives. Most participants (11) had multiple workspaces, with the median being 2 to 3 workspaces. All participants generally made at least minor customization changes to each workspace, including opening and closing different views on the code, changing the layouts of perspectives, and changing code formatting preferences, but none of the participants customized their menus and

toolbars individually. A reset feature is provided for perspectives, and 6 participants reported occasional use of this feature when they had changed their perspective significantly. Users can also create new perspectives by first customizing a perspective, then saving it under a new name. Only 1 participant used this feature.

Challenges in Reducing Functionality

As expected based on our participants' varied exposure to RAD, we found that people had different degrees of understanding about how roles and capabilities technically worked. While almost all participants (12) were aware of capabilities, only 8 of the 11 participants who did not use Eclipse as their main development platform were aware of roles, and only 6 of those knew how to change them. Interpretation of results should be made in this context.

The majority of participants (8) explicitly stated they liked roles or capabilities in principle, that is, their potential to reduce features in the interface. When asked if they would remove roles and/or capabilities from the interface, only 1 participant suggested removing both. While this positive response should motivate further work on roles and capabilities, several issues affected the user experience and these can be broadly grouped with respect to partitioning of functionality, presentation, and individual differences.

Partitioning functionality

Groups of features in a customization model should be relatively independent, cohesive, and meaningful to users. We identified several challenges related to this.

Fine-grained capabilities, were more popular than coarse-grained roles because they better matched perceived needs. While roles and capabilities both offer high-level feature grouping for customization, they do so at different levels of granularity. Participants generally chose to enable and disable the finer-grained capabilities rather than enabling roles. Part of the reason was that they felt the variation in tasks performed by users nominally in the same work role made it difficult to define roles. We asked all but the 3 participants who used Eclipse as their main IDE which roles they would categorize themselves under, and we compared this to the roles which were actually enabled in the workspace they had accessible during or after the interview. All but 2 people identified with several more roles than were enabled in their workspaces.

Trigger points and capabilities were useful because they allowed the user to enable features as needed rather than predicting needs in advance. Five of the 6 participants who knew how to change roles generally left the default roles when they created a new workspace even though 3 of them had changed their roles at some point in an earlier workspace. They found it easier to enable functionality automatically through trigger points or by manually enabling capabilities, and 3 of those participants considered roles to be irrelevant because instead, they could simply change their capabilities. For example, P8 said:

> *"I know for the GUI itself, it's not very intuitive, saying 'This is what I'm going to do' up front." (P8)*

Only 1 participant used roles as his primary method of enabling functionality. This was not necessarily because the role matched his work practice better than it did for other participants: he stated he had chosen this specific role (*Advanced J2EE*) because it appeared to be the most comprehensive. Thus, it made it easy to enable a large set of features with a single click.

Partitioning based on task was more effective than on expertise. Our analysis also suggests that the criteria by which roles are defined impacts the effectiveness of the customization model. All 11 of the roles in RAD group functionality in a task-oriented manner; for example, the *Java Developer* role is associated with functionality that is likely to be needed by that type of developer. However, 4 of the roles were also distinguished by expertise level: *Web Developer Typical* versus *Web Developer Advanced* and *Enterprise Java* versus *J2EE Developer*. The former role in each of these pairings represents only a subset of the functionality of the latter. Eight participants expressed concern over the difficulty of distinguishing between the expertise-oriented roles. For example, when asked to identify which roles he fits under, P7 said:

> *"The main ones would be Enterprise Java and Modeling, and I guess the Advanced J2EE. Although I have no idea why there's Enterprise Java and Advanced J2EE. I almost think it would be better to just have one." (P7)*

Although partitioning by expertise has been shown to be effective for novice users [1], our findings suggest that it may not be as effective for differentiating between the tasks of more experienced users (intermediate vs. expert users).

Presentation

Effective communication of a complex customization model to the user is non-trivial.

Capabilities more closely matched concrete tasks, so were easier to interpret. Many participants (8) found it difficult to map from a name or short description of a role or capability to actual features in the interface, thus making it difficult to know how to effectively customize their interface. For example, P1 expressed this frustration:

> *"If I need something but if I don't know which capability I need to [enable], how can I use that?" (P1)*

While some of this may be attributable to issues with partitioning functionality, it also highlights the challenge of effectively communicating the customization model to the user when the model is complex, such as RAD's, and contains multiple levels of granularity. It will be interesting to explore whether communicating the underlying mapping of roles to features more effectively increases their adoption relative to capabilities.

Designers need to promote the ability to discover unknown or unused features while still filtering what is presented to the user. More than half the participants (8) were concerned

about hiding functionality and not being able to find features when some roles or capabilities were disabled, a finding similar to previous work with word processor users [8]. Because of this concern, 4 participants mentioned that they generally enabled all functionality to ensure that they would be able to find what they needed. Although this may be due to individual differences (see below), it defeats the purpose of having roles and capabilities in the first place. The concern over hiding features stemmed from both: (1) the need to locate functions of which the user is already aware, and (2) the ease with which users can learn about and use new features in the user interface.

Changing requirements concerns users. Our participants identified three situations in which they would be concerned about only having a filtered set of the features in the interface: when their role evolved, such as from a developer to a manager; when they temporarily needed a set of features associated with another role; and when they wanted to engage in exploratory behaviour of the interface for a short period of time.

Individual differences

Finally, we found that different participants had different reactions to reducing functionality in the user interface. Some felt overwhelmed by having many features while others were not bothered by extra functionality and preferred not to filter any features. As such, we need to cater to both feature-keen and feature-shy users [8], and to increase system trust, especially for those users who may be reluctant to customize even when a reduced-functionality interface could be more efficient. Four participants immediately enabled all functionality when creating a new workspace. To illustrate this, when asked which of the roles she would want enabled, P5's response was: "Every single one of them!" This behaviour supports the inclusion of a toggle mechanism, such as that provided in the multiple interfaces approach [7], to provide quick access to the full functionality set for this type of user.

Summary of Design Implications

Participants preferred to use finer-grained capabilities to roles, for several reasons that can inform future designs: (1) capabilities more closely matched the tasks a user performed, while roles were broader, not necessarily matching an individual user's tasks; (2) capabilities were more concrete, so it was easier to interpret the mapping from capabilities to individual features; and (3) capabilities could be easily enabled on an as-needed basis. Grouping of features based on advanced expertise levels was also less effective than grouping by task. As well, although most users wanted to filter features in their interface, it is important to consider how easily unknown or unused features can be discovered. Finally, for those users who do not want to filter any features, an easy toggle mechanism enabling the full functionality set should be provided.

CONCLUSION

An interview study has allowed us to identify several open issues in designing coarse-grained customization mechanisms. Our findings suggest that finer-grained, task-oriented groupings of features (i.e., capabilities) may be more effective than role-based groupings. The design implications are especially applicable for role-based and layered interfaces. The challenges we have identified with respect to partitioning of functionality, presentation, and individual differences highlight potentially critical design choices, and should guide further research in the area.

ACKNOWLEDGMENTS

We thank Jen Hawkins, Jin Li, Lawrence Mandel, and Arthur Ryman of the IBM Toronto Lab for their help in running this study. We also thank IBM Centers for Advanced Studies and NSERC for funding.

REFERENCES

1. Carroll, J. M., and Carrithers, C. Training wheels in a user interface. *CACM*, 27(1984):8, 800–806.
2. Christiernin, G.L., Lindahl, F., and Torgersson, O. Designing a multi-layered image viewer. *Proc. NordiCHI '04*, (2004), 181–184.
3. Findlater, L., and McGrenere, J. Evaluating reduced-functionality interfaces according to feature findability and awareness. Proc. IFIP Interact 2007, (2007), 592-605.
4. Greenberg, S. Personalizable groupware: Accommodating individual roles and group differences. *Proc. ECSCW*, (1991), 17-31.
5. Linton, F., Joy, D., Schaefer, H.-P., and Charron, A. Owl: A recommender system for organization-wide learning. *Educational Technology & Society*, 3(2000):1, 62-76.
6. Mackay, W. E. Triggers and barriers to customizing software. *Proc. CHI '91*, (1991), 153-160.
7. McGrenere, J., Baecker, R., and Booth, K. An evaluation of a multiple interface design solution for bloated software. *Proc. CHI 2002*, (2002), 163-170.
8. McGrenere, J., and Moore, G. Are we all in the same "bloat"? *Proc. Graphics Interface*, (2000), 187-196.
9. Plaisant, C., Kang, H., and Shneiderman, B. Helping users get started with visual interfaces: Multi-layered interfaces, integrated initial guidance and video demonstrations. *Proc. HCI International*, (2003), 790-794.
10. Shneiderman, B. Promoting universal usability with multi-layer interface design. *Proc. CUU 2003*, (2003), 1-8.
11. Smith, R. B., Hixon, R., Horan, B. Supporting flexible roles in a shared space. *Proc. CSCW*, (1998), 197-206.
12. Strauss, A., and Corbin, J. Basics of qualitative research: Grounded theory procedures and techniques. Sage, Newbury Park, CA, USA, 1990.