Citation: Holzinger, A., Brugger, M. & Slany, W. (2011) Applying Aspect Oriented Programming (AOP) in Usability Engineering processes: On the example of Tracking Usage Information for Remote Usability Testing. In: Marca, D. A., Shishkov, B. & Sinderen, M. v. (Eds.) *Proceedings of the 8th International Conference on electronic Business and Telecommunications.* Sevilla, SciTePress INSTICC Setubal, 53-56.

APPLYING ASPECT ORIENTED PROGRAMMING IN USABILITY ENGINEERING PROCESSES

On the example of Tracking Usage Information for Remote Usability Testing

Andreas Holzinger, Martin Brugger
Research Unit Human–Computer Interaction, Institute for Medical Informatics, Medical University Graz
a.holzinger@hci4all.at, m.brugger@hci4all.at

Wolfgang Slany
Institute for Software Technology, Graz University of Technology, Austria
wolfgang.slany@tugraz.at

Keywords: Aspect oriented programming, usability engineering, remote usability testing

Abstract: Usability Engineering can be seen as a crosscutting concern within the software development process. Aspect Oriented Programming (AOP) on the other hand is a technology to support separation of concerns in software engineering. Therefore it stands to reason to support usability engineering by applying a technology designed to handle distinct concerns in one single application. Remote usability testing has been proven to deliver good results and AOP is *the* technology that can be used to streamline the process of testing various software products without mixing concerns by separating the generation of test data from program execution. In this paper we present a sample application, discuss our practical experiences with this approach, and provide recommendations for further development.

1 INTRODUCTION

The concept of Aspect Oriented Programming (AOP) describes a development methodology for separating crosscutting concerns during software development (Kiczales et al., 1997), (Furfaro, Nigro & Pupo, 2004) and has been applied in business applications (Di Francescomarino & Tonella, 2009).

In contrast to Object-oriented programming (OOP), where the common functionality is pushed up in the hierarchy tree, AOP separately defines such aspects and uses an AOP environment to manage the correct composition to a single executable program. (Elrad, Filman & Bader, 2001) distinguish aspects from high-level concerns such as security and quality of service down to low-level concerns similar to caching and buffering. The separation of concerns can lead to a cleaner system design and implementation.

Usage logging for remote asynchronous usability testing is a typical application for AOP allowing easy integration into existing software systems (Tarby et al., 2007).

Due to the transparent invocation of aspects no additional complexity has to be added during regular development of a software program. Depending on the software development language, AOP can even be applied to black-box systems.

Figure 1 shows a good example of shared concerns across independent OOP classes on the example of a medical application (shaded area).

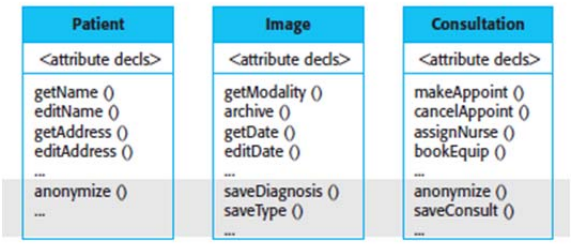

Figure 1 Example from healthcare: Three classes included in a patient record system along with some of the core methods for managing patient information: shaded areas show the methods required to implement the secondary statistics concern (Sommerville, 2010).

2 RELATED WORK

Originally, (Kiczales et al., 1997) found in their work many programming problems for which neither procedural nor object-oriented programming techniques were sufficient in capturing design decisions, resulting in "tangled" program code which is difficult to maintain. Kiczales et al. presented an analysis of why certain design decisions are so difficult to capture in actual code and called these properties "aspects". They showed that the reason they have been hard to capture is that they cross-cut the system's basic functionality. Consequently, they developed a new programming technique, called AOP, which makes it possible to clearly express programs involving such aspects, including appropriate isolation, composition, and reuse of the aspect code.

Object-oriented programming (OOP) proved to be a well-established and modern technology to model real domain problems. Some problems demand difficult design decisions as their nature do not fit well the OOP approach. The programming technique of AOP supports the problem nature of cross-cutting concerns as an extension to OOP.

(Tarby et al., 2007) report, concerning the types of application, that the AOP approach currently can trace any application written in Java and supporting AspectJ, where traced applications are today mainly interactive applications (WIMP – windows, icons, menu, pointing).

(Kojarski & Lorenz, 2007) present a practical composition framework, called AWESOME, for constructing a multi-extension weaver by plugging together independently developed aspect mechanisms. Their framework has a component-based and aspect-oriented architecture that facilitates the development and integration of aspect weavers. In a practical implementation they demonstrated the construction of a weaver for a multi-extension AOP language that combines COOL and AspectJ; however, the composition method is not exclusive to COOL and AspectJ – it can be applied to combine any comparable reactive aspect mechanisms.

3 REMOTE ASYNCHRONOUS USABILITY EVALUATION

Good usability is meanwhile accepted as a key factor for success of every application whether it is a typical e-Business application (Calisir et al., 2010) or a safety critical application in e-Health where it is clearly demonstrated that bad usability can cause serious problems (Rolleke, 2009), (Nielsen, 2005).

Usability is most often defined as the ease of use and acceptability of a system for a particular class of users carrying out specific tasks in a specific environment; whereas ease of use affects the end users' performance and satisfaction, acceptability affects whether the product is used or not. One of the basic lessons we have learned in human-computer interaction (HCI) is that usability must be considered before prototyping takes place. The earlier critical design flaws are detected, the more likely they can be corrected. Thus, user interface design should more properly be called user interface development, analogous to software development, since design usually focuses on the synthesis stages, and user interface components include metaphors, mental models, navigation, interaction, appearance, and usability (Holzinger, 2005).

Still the process of usability engineering and its iterative approach is often costly and time consuming. Using automation to decrease time and costs of usability evaluations therefore directly impacts the economic success of products. The process of usability evaluation can described by the activities following activities (Ivory & Hearst, 2001):

- Capture: collecting usability data
- Analysis: interpreting usability data
- Critique: suggesting solutions

Using AOP the "capture" phase is directly influenced as the effort to collect information can be drastically reduced while the quality of usability data can be increased and easily customized. Therefore also the other activities benefit of AOP indirectly.

Collecting Usage Information

In Usability Engineering the task of automatic capturing of usability data can be greatly enhanced by the usage of AOP. Within the scope of this work the application of AOP does not cover a whole usability engineering process. The technology can be applied across several usability evaluation methods during the "Capture" automation type mentioned in (Ivory & Hearst, 2001). Also the distinction of WIMP (windows, icons, pointer, and mouse) and Web interfaces applies to this contribution as we focus on current client technologies. The current progress in web technologies mitigates this distinction as modern Javascript powered web applications also allow powerful client-side applications where AOP could be applied to collect usage information.

4 SAMPLE IMPLEMENTATION

Using AOP to collect usage information is the main focus of this work. Therefore a sample application has been developed taking advantage of AOP. Given the current popularity of Objective-C (OBJC) because of the iOS platform and the lack of a maintained and publicly available AOP implementation, the sample application was implemented using the MacOS/iOS platform and Objective-C 2.0. The OBJC runtime allows easy integration of AOP with minimal changes to the original program, which is very important to allow an easy integration into existing projects. Nevertheless this concept can be transferred to other platforms without difficulties.

4.1 Sample Application

The sample application was developed to evaluate different input methods in an application for script breakdown. Pre-selection heuristics (PSH) were used to enhance the tedious task of text selection. Another mechanism to speed up script breakdown is Auto-advancing short-key (AAS) tagging which was the main focus to be tested. AAS automatically advances the selection to the next text passage after classification of a text component identified by the PSH. Within the test the acceptance of these technologies should be validated.

As the software is targeted at a very specific audience a standard laboratory usability test would not have been able to be performed within a reasonable time frame. For this specific test a self contained software package has been developed to be distributed among the target group. The invitations to the test were distributed as personal invitation via email, the results were automatically transmitted by the test software. Several domain experts across Europe were invited to participate in the test. The self contained test setup allowed the test to be performed in the natural working environment of the target group. Each test consisted of a background information form for gathering statistical data, a tutorial document presenting the different input methods, the actual test and a SUS feedback questionnaire.

Aspects have been implemented to track interface events of keyboard input (AAS), context menus, and drag and drop actions. A centralized logging facility was used to persistently store log output. A custom plain text format was used for better readability of the log output. For automated processing an XML format could easily be created by exchanging the Logger class. Within the logged data the quality of AAS could be measured by identifying user corrections to the proposed selections by the AAS mechanism.

4.2 Definition and Implementation of Aspects

Adding aspects to the given object hierarchy was achieved using a technology called “method swizzling” offered by the OBJC runtime. Aspect functionality was achieved by implementing categories to existing classes. Categories allow the extension of existing classes without inheritance and knowledge of the original class source-code providing access to the internal state of an object. Therefore this technology suits the AOP approach very well, separating the original functionality from aspect behavior. For weaving aspects to the existing OOP model, the OBJC runtime exchanges all calls to methods defined in a configuration, with aspect code altering the actual execution of the program. The configuration is defined in the property list format using dictionaries and arrays to define aspects for selected classes.

Key	Type	Value
▼ Root	Dictionary	(1 item)
▼ AspectClasses	Array	(3 items)
▶ Item 0	Dictionary	(2 items)
▶ Item 1	Dictionary	(2 items)
▼ Item 2	Dictionary	(2 items)
OriginalClass	String	NSApplication
▼ Selectors	Dictionary	(1 item)
sendAction:	String	InteractionLogger_sendAction:

Figure 2: Sample configuration for adding aspects to existing classes.

4.3 Application Design

OBJC emerges to support the implementation to AOP very well. Only one additional class is necessary for reading the aspect configuration and managing the aspect weaving. Using the provided configuration aspect methods are exchanged with the original methods at runtime. The aspect is also responsible for calling the original method. This can easily break the original application behaviour, still allowing alteration of the program execution. Implementing an alternate approach only allowing pre- and post-method invocations should also be taken into consideration. To the original Application the addition of AOP is completely transparent. The second class used as AOP infrastructure is the logger itself. It is not directly AOP related but an essential addition to the original application. Three aspects are necessary to identify all necessary user interface

events. In a blackbox system aspects can be attached to known base classes to log information on a low level. If additional information about the application design exists, customized aspects can be developed. OBJC has several tools for analyzing binary applications at runtime. F-Script is a well-known tool revealing enough information about almost any application to allow easy integration of AOP into existing applications.

5 CONCLUSIONS

As in the definition of AOP, the term cross cutting concern can be perfectly applied to usability engineering interests. The concept of AOP therefore is perfectly suitable for adding data collection facilities to existing applications for usability testing. AOP is also easily portable to other programming languages and the application within this paper just a sample for how easy it is to reduce the effort for usability testing. A best case scenario reduces code to a single implementation of a shared concern across all relevant classes without increasing code complexity. Still the evaluation of the gathered data is a challenging task. Automatic processing of collected data is essential as large volumes can easily be gathered. Several usage patterns like repetitive operations on a single element were identified hinting at usability issues. Also the detection of heavily used features can be used to further increase the user satisfaction by optimizing these workflows and every change can be evaluated afterwards for its effectiveness. A downside of the easy application of AOP is related to privacy concerns. Collected data should always be anonymized and the least amount of sensitive information should be recorded. A user notification and opt-in mechanism is mandatory, in order to avoid negative user experiences.

ACKNOWLEDGEMENTS

We thank Karl Heinz Struggl for the technical support in implementation of the sample application.

REFERENCES

Calisir, F., Bayraktaroglu, A. E., Gumussoy, C. A., Topcu, Y. I. & Mutlu, T. (2010) The relative importance of usability and functionality factors for online auction and shopping web sites. *Online Information Review*, 34, 3, 420-439.

Di Francescomarino, C. & Tonella, P. (2009) *Cooperative Aspect Oriented Programming for Executable Business Processes*. New York, IEEE.

Elrad, T., Filman, R. E. & Bader, A. (2001) Aspect-oriented programming - Introduction. *Communications of the ACM*, 44, 10, 28-32.

Furfaro, A., Nigro, L. & Pupo, F. (2004) Multimedia synchronization based on aspect oriented programming. *Microprocessors and Microsystems*, 28, 2, 47-56.

Holzinger, A. (2005) Usability engineering methods for software developers. *Communications of the ACM*, 48, 1, 71-74.

Ivory, M. Y. & Hearst, M. A. (2001) The state of the art in automating usability evaluation of user interfaces. *ACM Computing Surveys (CSUR)*, 33, 4, 470-516.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J. M. & Irwin, J. (1997) Aspect-oriented programming. In: Aksit, M. & Matsuoka, S. (Eds.) *Ecoop'97: Object-Oriented Programming*. Berlin 33, Springer-Verlag Berlin, 220-242.

Kojarski, S. & Lorenz, D. H. (2007) AWESOME: An aspect co-weaving system for composing multiple aspect-oriented extensions. *ACM Sigplan Notices*, 42, 10, 515-534.

Nielsen, J. (2005), Medical Usability: How to Kill Patients Through Bad Design In: Jakob Nielsen's Alertbox, April 11. Online available: http://www.useit.com/alertbox/20050411.html, last access: 2011-01-13

Rolleke, T. (2009) Incident reports to BfArM support the importance of usability for patient safety. In: Dossel, O. & Schlegel, W. C. (Eds.) *World Congress on Medical Physics and Biomedical Engineering, Vol 25, Pt 12*. New York, Springer, 298-300.

Sommerville, I. (2010) *Software Engineering 9*. New York, Addison-Wesley.

Tarby, J. C., Ezzedine, H., Rouillard, J., Tran, C. D., Laporte, P. & Kolski, C. (2007) Traces using aspect oriented programming and interactive agent-based architecture for early usability evaluation: Basic principles and comparison. In: Jacko, J. A. (Ed.) *Human-Computer Interaction, Part 1, Interaction Design and Usability*. Berlin, Springer-Verlag Berlin, 632-641.