Jens Knoop, Uwe Zdun (Hrsg.): Software Engineering 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 69

An Empirical Study on Program Comprehension with Reactive Programming

Guido Salvaneschi, Sven Amann, Sebastian Proksch, and Mira Mezini[1]

Abstract:

Starting from the first investigations with strictly functional languages, reactive programming has been proposed as *the programming paradigm* for reactive applications. The advantages of designs based on this style over designs based on the Observer design pattern have been studied for a long time. Over the years, researchers have enriched reactive languages with more powerful abstractions, embedded these abstractions into mainstream languages – including object-oriented languages – and applied reactive programming to several domains, like GUIs, animations, Web applications, robotics, and sensor networks. However, an important assumption behind this line of research – that, beside other advantages, reactive programming makes a wide class of otherwise cumbersome applications more comprehensible – has never been evaluated. In this paper, we present the design and the results of the first empirical study that evaluates the effect of reactive programming on comprehensibility compared to the *traditional* object-oriented style with the Observer design pattern. Results confirm the conjecture that comprehensibility is enhanced by reactive programming. In the experiment, the reactive programming group significantly outperforms the other group.

Keywords: Reactive Programming, Controlled Experiment, Program Comprehension

Reactive applications are a wide class of software that needs to respond to internal or external stimuli with a proper action. Examples of such applications include user-interactive software, like GUIs and Web applications, graphical animations, data acquisition from sensors, and distributed event-based systems.

Over the last few years, reactive programming (RP) has gained the attention of researchers and practitioners for the potential to express otherwise complex reactive behavior in intuitive and declarative way. RP has been firstly introduced in Haskell. Influenced by these approaches, implementations of RP have been proposed in several widespread languages, including Java, Javascript and Scala. Recently, concepts inspired by RP have been applied to production frameworks like Microsoft Reactive Extensions (Rx), which received great attention after the Netflix success story. Finally, a lot of attention in the front-end developers community is revealed by the increasing number of libraries that implement RP principles, among others React.js, Bacon.js, Knockout, Meteor, and Reactive.coffee.

The relevance of RP comes from the well-known complexity of reactive applications, which are hard to develop and understand, because of the mixed combination of data and control flow. The Observer design pattern is widely used for such applications. It has the advantage of decoupling observers from observables. But, when it comes to program readability, it does not make things easier, because of dynamic registration, side effects in call-

[1] Technische Universität Darmstadt, Fachbereich Informatik, Fachgebiet Softwaretechnik, Hochschulstr. 10, 64289 Darmstadt, Deutschland, <lastname>@st.informatik.tu-darmstadt.de

backs, and inversion of control. In contrast, RP supports a design based on data flows and time-changing values: the programmer states which relations should be enforced among the variables that compose a reactive program and the RP runtime takes care of performing all the required updates. Dependencies are defined explicitly instead of being hidden in the control flow. Combination can be guided by types as opposed to callbacks that return void. Contrarily to the Observer pattern, control is not inverted and less boilerplate is required, since collecting dependencies and performing the updates is automatized by the framework. Based on these arguments, it has been argued that RP greatly improves over the traditional Observer pattern used in OO programming both from the *software design* perspective as well as from the perspective of facilitating the comprehensibility of the software.

Yet, little empirical evidence has been provided in favor of the claimed advantages of RP – especially enhancement of comprehensibility. Despite the intuition about its potential, the reactive style is not *obviously* more comprehensible than the Observer design pattern. For example, in the Flapjax paper [Me09] a Javascript application based on Observer is compared against a functionally equivalent RP version. The authors argument that the RP version is much easier to comprehend. However, the reader is warned that: *"Obviously, the Flapjax code may not appear any 'easier' to a first-time reader"*. Doubting, at this point, is legitimate: does RP really make reactive applications easier to read? Also, it is unclear how much expertise is required to find a RP program "easier" – if ever.

To fill the gap, this paper provides the first empirical evaluation of the impact of RP on program comprehension compared to the traditional technique based on the Observer design pattern. The experiment, based on the REScala language [SHM14], involves 38 subjects that where divided into an RP group and an OO group. They were shown a reactive application and their understanding of the reactive functionalities was measured. To the best of our knowledge, such a study has never been conducted before. Results show that (1) RP increases correctness of program comprehension, (2) comprehending programs in the RP style does not require more time than comprehending their OO equivalent, and (3) in contrast to OO where score results are correlated to programming skills, with RP (advanced) programming skills are not needed to understand reactive applications. The last result suggests that RP lowers the entrance barrier required to understand reactive applications.

References

[Me09] Meyerovich, Leo A.; Guha, Arjun; Baskin, Jacob; Cooper, Gregory H.; Greenberg, Michael; Bromfield, Aleks; Krishnamurthi, Shriram: Flapjax: A Programming Language for Ajax Applications. In: Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications. OOPSLA '09, ACM, New York, NY, USA, pp. 1–20, 2009.

[SHM14] Salvaneschi, Guido; Hintz, Gerold; Mezini, Mira: REScala: Bridging Between Object-oriented and Functional Style in Reactive Applications. In: Proceedings of the 13th International Conference on Modularity. MODULARITY '14, ACM, New York, NY, USA, pp. 25–36, 2014.