The Journal of Systems and Software 125 (2017) 47–67

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Automatic verification and validation wizard in web-centred end-user software engineering

David Lizcano[a,*], Javier Soriano[b], Genoveva López[b], Javier J. Gutiérrez[c]

[a] Universidad a Distancia de Madrid, UDIMA, Madrid, Spain
[b] Universidad Politécnica De Madrid, Madrid, Spain
[c] Universidad de Sevilla, Spain

ARTICLE INFO

Article history:
Received 7 October 2015
Revised 7 September 2016
Accepted 13 November 2016
Available online 15 November 2016

Keywords:
End-user software engineering
Web engineering
Reliability
End-user programming
visual programming
human-computer interaction

ABSTRACT

This paper addresses one of the major web end-user software engineering (WEUSE) challenges, namely, how to verify and validate software products built using a life cycle enacted by end-user programmers. Few end-user development support tools implement an engineering life cycle adapted to the needs of end users. End users do not have the programming knowledge, training or experience to perform development tasks requiring creativity. Elsewhere we published a life cycle adapted to this challenge. With the support of a wizard, end-user programmers follow this life cycle and develop rich internet applications (RIA) to meet specific end-user requirements. However, end-user programmers regard verification and validation activities as being secondary or unnecessary for opportunistic programming tasks. Hence, although the solutions that they develop may satisfy specific requirements, it is impossible to guarantee the quality or the reusability of this software either for this user or for other developments by future end-user programmers. The challenge, then, is to find means of adopting a verification and validation workflow and adding verification and validation activities to the existing WEUSE life cycle. This should not involve users having to make substantial changes to the type of work that they do or to their priorities. In this paper, we set out a verification and validation life cycle supported by a wizard that walks the user through test case-based component, integration and acceptance testing. This wizard is well-aligned with WEUSE's characteristic informality, ambiguity and opportunisticity. Users applying this verification and validation process manage to find bugs and errors that they would otherwise be unable to identify. They also receive instructions for error correction. This assures that their composite applications are of better quality and can be reliably reused. We also report a user study in which users develop web software with and without a wizard to drive verification and validation. The aim of this user study is to confirm the applicability and effectiveness of our wizard in the verification and validation of a RIA.

1. Introduction

The number of end-user programmers (people who program to achieve the result of a program primarily for personal rather than public use) (Ko et al., 2011) grows year by year and is much greater than the number of professional developers (Scaffidi et al., 2005). According to the US Bureau of Labor and Statistics, compared with three million professional programmers in the United States, there were more than 60 million end-user programmers (EUPs) using spreadsheets and databases at work in early 2014, many writing formulas and dashboards to support their job (Cao et al., 2014). This has spawned a lot of interest in research into all aspects of software development by end-user programmers and user-centred software engineering.

At the start of our research, we defined a model designed to enable EUPs to handle components tailored to their experience and problem domain knowledge (Lizcano et al., 2011a). The defined model is based on components of different levels of abstraction. Components and connectors, that is, elements that can be used to set up a data flow among components of the same level, become less detailed as we move up the hierarchy. We analysed the features required by a visual EUP-centred development environment empowering EUPs to effectively handle the defined components and connectors in (Lizcano et al., 2011b).

The next step was to define web end-user software engineering (WEUSE) and to specify the analysis, design and implementation stages of a RIA life cycle as enacted by an EUP (Lizcano et al., 2013). We proposed suitable mechanisms for supporting the activ-

* Corresponding author.
E-mail addresses: david.lizcano@udima.es (D. Lizcano), jsoriano@fi.upm.es (J. Soriano), glopez@fi.upm.es (G. López), javierj@us.es (J.J. Gutiérrez).

http://dx.doi.org/10.1016/j.jss.2016.11.025
0164-1212/© 2016 Elsevier Inc. All rights reserved.

ities of the RIA life-cycle phases. The aim was to empower EUPs to successfully compose the component graph and help them to rapidly build a RIA adapted to a specific problem that they need to solve. The reported WEUSE achieves this goal using a development wizard (DW) composed of well-defined and structured tasks. This wizard provides users with guidance to develop RIAs, which is outside the realms of their knowledge. The DW, implemented within an end user-centred visual development environment, called FAST, walks EUPs through these life-cycle stages. FAST is an EUP-centred integrated development environment developed as part of a 7th European framework programme project.

A RIA built by an EUP is an application composed of a set of components with inputs and outputs. The bottom-level components are able to invoke services and access web resources. They are designed and then published in repositories shared by the proprietors of the services/resources that are to be made visible. In order to promote the end-user development (EUD) philosophy, they are made accessible to EUPs or third parties. Users use EUD tools (like Yahoo! Pipes and Dapper, Kapow, JackBe, FAST, and so on) to access the above repositories and create personal compositions by connecting up some of these bottom-level components with others and/or building composite applications.

Even though EUPs run some tests to check the goodness of the RIA that they have developed, they do not really perform a systematic verification and validation process in order to reliably check that the built RIA is error free.

In this paper, we describe a WEUSE verification and validation stage as a combination of test case generation and test case execution for testing a RIA at component, integration and acceptance level. This verification and validation process assures that the developed RIA conforms to the end-user specification and meets the needs for which it was built. During this stage, the users are assisted by a *What You See Is What You Test* testing wizard (TW) that walks untrained users through the implementation of a user-centred component, integration and acceptance testing plan.

Note that the aim of the testing process is to find and locate as many defects as possible, whereas the aim of the debugging process is to fix and remove the detected defects. In traditional software engineering, the testing team is responsible for testing, whereas the development team carries out debugging. In the case of WEUSE, however, the EUP performs both tasks. To do this, the EUP first executes the TW and then fixes the bugs that the wizard has detected and not automatically corrected. In the last analysis, the aim of the WEUSE verification and validation process described in this paper is to check that the software system meets the specifications and serves its intended purpose by applying the testing and debugging processes.

Effective verification and validation should prevent errors being compounded by the reuse of buggy software. Therefore, EUPs need to have access to an automatic system to validate their developments. Verification and validation should not, however, take too much time or effort because EUPs view it as being an unnecessary and unimportant process.

The WEUSE verification and validation stage includes three levels of testing: component, integration and acceptance testing. The unit components published in the catalogues have been built by specialised software providers and should, in principle, work correctly. However, EUPs may publish parameterisations and compositions based on these unit components, and these new ad-hoc components should be tested. Component and integration tests are run by the TW automatically to check that end-user software is error free. Inputs are the endpoints of the range of each of the expected data types. The TW runs acceptance tests based on data requested from users. It uses black-box testing to check how the data flow through the RIA. The TW displays the intermediate and final data in tabular format, indicating whether or not the processed internal and external data reach their destination. End users, who are problem domain experts, should then have no trouble with data checks. Based on the execution of the test cases that it generates, the TW will analyse components that generate any erroneous data items identified by users and ask the DW to suggest an alternative design (based on data flows among components or other equivalent components published in the catalogue). The RIA component, integration and acceptance tests are further divided into two stages: wizard-driven test case generation and test case execution, as described in Section 3.

The remainder of the article is divided as follows. Section 2 introduces research on the provision of support for the verification and validation of EUDs. There is no support for such tasks in the RIA field, but we have used earlier proposals in other EUD fields, such as spreadsheets, in our research. Section 3 reports the proposed verification and validation process for WEUSE, defining the types of test cases to be executed at each level and implementation details of how the TW supports EUPs. Section 4 documents our working hypotheses and the user study conducted to examine the evaluation criteria. This section includes detailed information on the user study to assure that it can be replicated in other fields. It also includes a statistical study of the results. Section 5 discusses threats to validity. Finally, Section 6 reports our conclusions, setting out the major contributions of our research to the state of the art and future lines of research that we intend to undertake.

2. Related work: end-user PROGRAMMER VERIFICATION and validation

Spreadsheets were the first real examples of developments by EUPs in the field of EUD (Rothermel et al., 2001; Chambers and Erwig, 2009). Over recent years errors made by EUPs have led to huge financial losses and defective quality at small and large companies alike (losses that were documented up to 20 years ago (Panko, 1995) and are still occurring). In response, research focused on *What You See Is What You Test* has been conducted in order to give users access to systematic testing procedures to validate their spreadsheets (Burnett et al., 2002).

Surprise-Explain-Reward is one What You See Is What You Test strategy (Wilson et al., 2003). This strategy employs surprise to draw the user's attention to software engineering tasks. Users are then encouraged, through explanations and rewards, to follow through with appropriate actions. This strategy has its roots in three areas of research—research about curiosity (psychology), Blackwell's model of attention investment (psychology/HCI), and minimalist learning (educational theory, HCI)—and has been used in work on model-driven methods (Burnett, 2009). We have used this strategy to provide users with visual guidance on the testing procedure in order to check that the right RIA has been built right. The procedure is explained in Section 3.

As mentioned in the introduction, we regard verification as a process of checking that the RIA conforms to the specifications given by the analyst and designer, roles played in this case by the end user. This process checks that the program implements all the sentences specified by the user to define the RIA (each sentence is a use case). On the other hand, validation, also performed by the user, checks that the result of executing the RIA satisfies user needs.

In the following, we summarise the end-user software engineering verification and validation activities identified in the state of the art, the problems that EUPs face within each activity and what solutions are now available for these problems.

- Generation of test cases for RIA use case verification and validation: