Building a Bridge between User-Adaptive Systems Evaluation and Software Testing

Veronika Bogina
Information Systems Dept.
The University of Haifa,
Haifa 31905, Israel
sveron@gmail.com

Tsvi Kuflik
Information Systems Dept.
The University of Haifa
Haifa 31905, Israel
tsvikak@is.haifa.ac.il

ABSTRACT
User Adaptive Systems (UASs) are futile without software. Moreover, integrating user modeling component into software system may add bugs if not tested properly. However, the evaluation of UASs does not intersect with software evaluation as commonly defined in Software Engineering. We suggest adopting the common software engineering practices, changing the community's practice and methods by integrating software testing as an integral part of any study involving software development. That will allow win-win situation for both: the researcher and community, since the code will be bug-free and hence easily reproducible/reusable by other members of the community.

CCS Concepts
• Information systems → Recommender systems • Software and its engineering → Software testing and debugging

Keywords
Software Engineering; Testing; Evaluation; User-Adaptive Systems

1. THE BRIDGE FOUNDATION
User-Adaptive Systems (UASs) are interactive systems that adjust their functionality to individual users according to the user model that was built by learning user behavior, inference, or decision making [13]. Over the last decade, a wide variety of different UASs has been introduced. Indeed, they incentivize researchers to publish their results by writing papers that often have the same structure: They contain of few common essential parts, like introduction, related work, experiment/method, evaluation and conclusions. The evaluation part allows us, as researchers, to decide whether this specific system lives up to both: the scientific community and the end user expectations in terms of quality and performance [16]. However, no one, as far as we know, reported on testing these UAS for their correctness. This is, primarily, because performing a study without testing the software properly first is the reality in our community. As Valentino Rossi maintained once:" Once the races begin it's more difficult and there is never that much time for testing"[9].

Let us consider other researchers' expectations within the same scientific community when they read a relevant paper and are eager to replicate the same experiment again but with a different setup. There are two primary aspects in such scenario: reproducibility and correctness of the method/algorithm/system that introduced in the paper. Nowadays, the reproducibility is a hot topic [3][14]. Researchers' intent is obvious: they want to be able to compare their algorithm with the published one by reproducing it according to the paper that describes the study. Moreover, clear methodology eliminates a redundant mail correspondence, when researchers approach the authors of the published paper for more details, and by that reduces a frustration of people that often struggle with the guessing an author's intention. Albeit everyone agrees with the fact that it is essential to improve the quality of research algorithms by at least making their code publicly available and elaborating on the tools one had used during the research, nobody reports on any testing done to ensure that a system provides correct, consistent results. Once researchers get good results in terms of accuracy/coverage/novelty/serendipity or any other predefined evaluation metric, they hasten to share their findings with the community. However, without testing whether they have bug in their implementation or even inconsistency in the chain of actions according to the paper they have published, the results may be questionable.

Hence we inquire the reader:" How do you know whether your User Adaptive system's results are correct? Have you tried to verify your results by writing new code and reproducing the steps that were defined in your paper?" By following such strategy, you can kill two birds with one stone: first, test that your method/experiment is reproducible and secondly prove that it works correctly by testing results with other implementation.

The goal of this position paper is to examine existing evaluation methods in both UASs and Software Engineering and suggest testing approach that will strengthen this research field's outcome in the future.

Evaluation metrics are commonly used to determine both the quality and performance of UASs [2]. Most frequently used are statistical evaluation methods, personality tests, accuracy, RMSE, *A*/B Testing and Benchmarking*.* Albeit all these metrics appraise UASs on their performance, accuracy and statistical significance, none of them addresses software testing for code, that is used to implement these UAS.

2. A BRIDGE TO SOFTWARE TESTING
According to Myers' classic definition: "Testing is the process of executing a program with intention of finding errors." The intent of the testing is to discover as many errors as possible and by that bring the tested software to the accepted level of quality [5].

By taking different approaches software tests can be classified incongruously: according to the testing concept or to the requirements [5]. The former is related to the black box testing (functionality) and the while box testing (structural). The latter is defined by McCall's classic model for classification of software quality requirements [7] that is shown in the Table 9.1[6] .

The testing strategy choice depends on the software and its requirements: whether one develops desktop, web, android or mission critical system applications.

Let us peruse two kinds of applications to show the differences in the testing process. *Web application*, a client-server software

application, differs from other applications in few ways. Indeed, it can be accessed by a wide number of users, from different parts of the world. Since each one of them uses different hardware, OS, Web browser and etc., such application should be able to run on heterogeneous execution environments. Moreover, the ability to respond the user input in real time is essential [4]. Common examples for such applications are web mails, online retail stores, instant messaging chats, wikis and so on [12]. From the testing perspective an executing performance, an availability testing, web accessibility, different web browsers, operating systems and middleware testing, security, usability, hyperlinks testing are germane testing mechanisms that should be used here in order to verify functional and non-functional requirements [4] .

On the other hand, in *mission critical systems*, whose failure may cause the failure of some goal-directed activity, errors or failures cannot be tolerated. Moreover, in safety-critical applications failure can be catastrophic. Meaning that errors in such systems are unacceptable. Thus reliability, availability, clear documentation and instructions, proper design and reviews, security are essential parts of the genuine testing [15]. Common examples are online banking systems, railway and aircraft operating systems, electric power and other similar computer systems [10].

As can be seen above software testing is an essential part of software development. All applications require such approach, with no exception and depending on their functionality and the goal, various methods for testing are chosen.

3. DISCUSSION AND CONLCUSIONS

In this section, we claim that since user models and user modeling components should not be separated from the software, testing techniques that are applicable in software development should also suit User Modeling research. The question only is what testing techniques can be applied from Software Engineering field.

After scrutinizing various factors categories [6] and existing evaluation methods, it caught our eyes that there is a gap in UASs testing and only *Operation* factor is partially covered in UASs evaluation. Yet *Revision* and *Transition* are not considered. From Revision perspective there is a need to ensure code's testability and it needs to be tested at least for accuracy, as a starting point. Let us argue on how using *Transition* techniques can be advantageously.

Wouldn't it be useful to make our code *reusable* by publishing it on GitHub with good documentation and clear code and by this allowing other researchers to reuse modules from our code?

What about *portability*? Whether we should write software on Java (write once, run everywhere) to exclude compatibility issues, like we have with different python versions and packages support. Adaptability, installability and *interoperability* [11] can only strengthen our systems and code exchange between researchers.

We advocate that it is an essential process to use a software testing in User Adaptive Systems in the future, though it is up to researchers to decide what technique to use for their specific system. It would be beneficial to both: a researcher and the community.

As a further matter, in order to be able to rely on the results of the system, there is a need to ensure its testability and it needs to be tested at least for accuracy. Moreover, in order to enable replication of the experiments and software reuse, both portability and reusability need to be tested as well. We believe that approaching testing in UASs is going to benefit both the researcher and the community. As Burt Rutan maintained:" Testing leads to failure, and failure leads to understanding**".**

4. REFERENCES

[1] Avazpour, I., Pitakrat, T., Grunske, L., & Grundy, J. (2014). Dimensions and metrics for evaluating recommendation systems. In Recommendation Systems in Software Engineering, 245-273. Springer Berlin Heidelberg.

[2] Chin, D. N. (2001). Empirical evaluation of user models and user-adapted systems. User modeling and user-adapted interaction, 11(1-2), 181-194.

[3] Davidson, J., Liebald, B., Liu, J., Nandy, P., Van Vleet, T., Gargi, U., & Sampath, D. (2010, September). The YouTube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems* 293-296. ACM.

[4] Di Lucca, G. A., & Fasolino, A. R. (2006). Web application testing. In Web Engineering , 219-260. Springer Berlin Heidelberg.

[5] Galin, D. (2004). Software quality assurance: from theory to implementation. Pearson education.

[6] Galin, D. (2004). Software quality assurance: from theory to implementation. Pearson education. Table 9.1, 188.

[7] General Electric Company, McCall, J. A., Richards, P. K., & Walters, G. F. (1977). Factors in software quality: Final report. Information Systems Programs, General Electric Company.

[8] http://www.brainyquote.com/quotes/quotes/b/burtrutan394556.html

[9] http://www.brainyquote.com/quotes/quotes/v/valentinor301235.html

[10] https://en.wikipedia.org/wiki/Mission_critical

[11] https://en.wikipedia.org/wiki/Portability_testing

[12] https://en.wikipedia.org/wiki/Web_application

[13] Jameson, A. (2001, December). User-adaptive and other smart adaptive systems: Possible synergies. *In Proceedings of the first EUNITE Symposium*, Tenerife, Spain.

[14] Kohavi, R., Longbotham, R., Sommerfield, D., & Henne, R. M. (2009). Controlled experiments on the web: survey and practical guide. *Data mining and knowledge discovery*, 18(1), 140-181.

[15] Parnas, D. L., van Schouwen, A. J., & Kwan, S. P. (1990). Evaluation of safety-critical software. Communications of the ACM, 33(6), 636-648.

[16] Said, A., & Bellogín, A. (2015, September). Replicable Evaluation of Recommender Systems. In Proceedings of the 9th ACM Conference on Recommender Systems 363-364. ACM.