Focusing Education on Energy Efficiency Measurements During Software Testing

Csaba Szabó[0000−0001−5147−2452]

Department of Computers and Informatics
Faculty of Electrical Engineering and Informatics, Technical University of Košice,
Letná 9, 04200 Košice, Slovakia
csaba.szabo@tuke.sk

Abstract. The mission of software engineering teachers is to prepare future software engineers who can master every problem during the whole software life cycle. Besides skills related to comprehension of stakeholders' needs and to composition of corresponding working software, the ability to check correctness of the results plays also a significant role. In this paper, we focus on the latter set of skills. We focus on testing, where the level of automation is less important. We emphasize the measurement nature of testing, more precisely, we focus on software energy consumption measurement that can be provided during testing at different levels. All these aspects are presented from the point of view of a software engineering educator, with the aim to present how to set up a software engineering lab session that focuses on energy efficiency measurement during software testing and, with the authors' comments on this proposal.

Keywords: Energy consumption · Measurement · Software engineering education · Testing.

1 Problem definition

One of the challenges for battery manufacturers is how long the battery can operate without being continuously charged, and of course there are many other challenges such as the size that greatly affects the shape of the device, and the other factor which is an important feature of the battery is the weight of the identifier. The battery is considered to be somewhat lighter compared to the device that needs a battery to operate. The challenge here is how to make its size smaller and lighter weight and certainly a high efficiency in terms of operating time of the mobile device without being charged.

On the top of this hardware challenge, its software challenge brother exists, namely that the software itself should support energy savings. Doing that without limiting the user experience is considered nowadays as a silent but important goal of each software development that is targeting any kind of portable devices.

Remembering that the energy consumption of any mobile device is influenced starting with the running applications through the access level of basic services

up to the actual mood of the user, to develop software for such devices is already a challenge [1]. One could say, the software development challenge is always the same, but we have to point out "mobility" as key system property here. Battery power status also determines the system performance due to operating system level configuration – well known as "energy saving preferences".

It is even harder, if one (in our case the teacher) has to prepare students for such challenges. All the already known "best practices" and "energy saving tips" have to be presented in a context, which is easily comprehensible to the students. This can be done by positioning the concepts into a known environment such as software testing and test automation [2], in our case. This is what we aim with this paper, this is the content of the upcoming sections, starting with the proposal followed by an evaluation and closing with further tips on improvement.

2 Solution proposal

As it was stated above, we have to find the best suitable environment to introduce energy consumption measurement [3] and energy efficiency evaluation practices. It could be both initial software development and software evolution as well, as both development phases of the generic software life cycle offer opportunities to measure the product being developed/evolved [4].

With the choice of initial software development, the benefit is that all activities could get a focus on energy saving related issues, while choosing the software evolution alternative offers the possibility to evaluate the improvement in the product implementation. On the other hand-side, software evolution requires the existence of a working software at its beginning, while initial software development is the process that creates the product starting with the first requirement on the software.

Putting the two possible approaches into the teaching environment, the best option would be to use both. One semester for initial development of software and another one for the evolution of the same software.

Usually the teacher does not have two semesters in a row to present the course contents in the way presented in the previous paragraph. This is the reason why we have to decide which kind of development to use to introduce the selected practices. From the point of view of the development process architecture, and by the fact that initial development could be also evolutionary, we decide for software evolution. It includes many activities of initial development (except early requirement gathering and analysis) and emphasizes the importance of testing and evaluation.

This choice allows the teacher to:

- Let students look back in their development history (past projects) to be critical to themselves.
- Let them evaluate their results using code metrics and energy consumption measurement (or estimation).
- Let them integrate the above activities into the standard verification & validation processes of software evolution.

Programming language of the development is not important, therefore the student can select any of their previous project for the evolution – or all of them, if they are competing in the number of evolved projects or programming languages used. But, programming language usually determines or limits the development environment and tools used. The selection of these tools and their plug-ins also offers a good support for code metrics evaluation.

Energy consumption measurement and energy efficiency evaluation usually requires a different tool, as there are only few development environment that integrate energy consumption measurement or estimation by now.

Regarding testing as measurement basis, we have to note that static code analysis is used to be a part of testing of software. Besides that, selected parts of the application code base are being executed during white box testing (mainly unit testing), which by a small extension of energy consumption measurement can present the energy consumption of the test case – an indirect look at energy consumption of the tested code. During black box testing, the complete application is being tested using test scenarios. These test scenarios are mainly comparable to intended all-day use cases of the software, others represent the borderline scenarios – including the ones when the user is in bad mood. Measuring energy consumption of the execution of these black box tests then gives an approximate (but direct) look at energy consumption of the product.

Here is the main benefit of evolution (when compared to initial software development). The teacher can prepare the starting version for the evolution, including code that can be improved, list of known bugs and the test base! The existence of test for retesting and regression testing is very important here as the productivity of evolution can be increased by this feature.

Assuming all above steps were made, the integration of energy efficiency measurements into the process of testing look from the student's perspective of activities as follows:

1. Select the product that could be your past project or from a repository.
2. Evaluate it using static code analysis, testing, energy measurement, usability survey, etc.
3. Improve it (different kinds of evolution such as add/change functionality, repair or adapt)
4. Retest to be sure you eliminated a defect or fault
5. Regression test (including re-evaluation)
6. Conclude results (make a final verdict across all available data, including energy efficiency)

3 Discussion

As energy consumption measurement is relatively new compared to other techniques such as static code analysis, black/white box testing and debugging, it might be the point of failure. But if it gets combined with these elder principles building a composite score for each student, the critical property is much lower.

Looking at the possibilities of grading, we can find various "levels of freedom", which offer themselves to be used separately or as part of a grade composition:

1. number of different projects,
2. number of programming languages applied/used,
3. code quality of the final product,
4. energy efficiency of the final product,
5. code quality improvement during software evolution,
6. energy efficiency improvement during software evolution.

Considering all "levels of freedom" of task completing, many competitions could be defined for the competitive students, while the achievers will collect the "small victories" in number of projects, the perfectionists' goal will be to optimize all code metrics and minimize energy consumption. For the average student, improvement of energy efficiency and code comprehensibility could be the achievable goal.

Student competition can be even more supported by not letting them return to their past projects, but allowing them to choose from a selected repository. Like with computer games, all game levels are then equally available to everyone. To support equity as well, a common public forum of students and teachers could be also created.

Our future work in this area will focus on configuration of a portable integrated development, testing and energy consumption estimation environment. This environment will be used in the frame of software evolution or initial development subjects to support education on energy efficiency measurement during software testing. It might limit students' creativity by offering a semi-closed sandbox, as hardware plays a very important role by the current architecture of energy consumption estimation. Some research aims to break this limitation – we are looking forward to those results to get them also integrated.

Acknowledgement

This paper is part of the Intellectual Output O2 of the Erasmus+ Key Action 2 (Strategic partnership for higher education) project No. 2017-1-SK01-KA203-035402: "Focusing Education on Composability, Comprehensibility and Correctness of Working Software". The information and views set out in this paper are those of the author(s) and do not necessarily reflect the official opinion of the European Union. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein.

References

1. J. Saraiva, M. Couto, Cs. Szabó, D. Novák: Towards Energy-Aware Coding Practices for Android, Acta Electrotechnica et Informatica, Vol. 18, No. 1, 2018, pp. 19–25. https://doi.org/10.15546/aeei-2018-0003

2. D. Li, Y. Jin, C. Sahin, J. Clause, W. G. J. Halfond: Integrated energy-directed test suite optimization, in Proc. of the 2014 International Symposium on Software Testing and Analysis, ISSTA 2014, ACM, 2014, pp. 339–350.
3. M. Santos, J. Saraiva, Z. Porkoláb, D. Krupp: Energy Consumption Measurement of C/C++ Programs Using Clang Tooling, in Proceedings of the SQAMIA 2017: 6th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Z. Budimac, ed., Belgrade, Serbia, 11-13.9.2017, Paper No. 15, 8 pages, also published online by CEUR Workshop Proceedings No. 1938 ISSN 1613-0073.
4. Cs. Szabó, J. Saraiva: Focusing software engineering education on green application development, in Conference of Information Technology and Development of Education – ITRO 2017, Novi Sad, Serbia, pp. 165–169, ISBN 978-86-7672-302-7.