POSTER

Automatic detection of software defects: an industrial experience

STEFANO GANDINI, Motorola Italy, Milan, Italy

DANILO RAVOTTO, Polytechnic of Turin, Turin, TO, Italy

WALTER RUZZARIN, Motorola Italy, Milan, Italy

ERNESTO ERNESTO SANCHEZ SÁNCHEZ, Polytechnic of Turin, Turin, TO, Italy

GIOVANNI SQUILLERO, Polytechnic of Turin, Turin, TO, Italy

ALBERTO PAOLO TONDA, Polytechnic of Turin, Turin, TO, Italy

Published: 08 July 2009

GECCO09: Genetic and Evolutionary Computation Conference
July 8 - 12, 2009
Québec, Montreal, Canada

Conference Sponsors:
SIGEVO

Automatic Detection of Software Defects: an Industrial Experience

S. Gandini*, D. Ravotto†, W. Ruzzarin*, E. Sanchez†, G. Squillero†, A. Tonda†
* Motorola Italia, † Politecnico di Torino, Dipartimento di Automatica e Informatica

ABSTRACT
Mobile phones are becoming more and more complex devices, both from the hardware and from the software point of view. Consequently, their various parts are often developed separately. Each sub-system or application may be worked out by a specialized team of engineers and programmers. Frequently, bugs in one component are triggered by the complex interaction between the different applications. Those errors sometimes lead to power dissipation and other misbehaviors that lower residual battery life, a catastrophic event from the user perspective. In this paper we propose a model-based automatic approach to uncover software bugs, which is intended to complement human expertise and complete a qualifying verification plan. The system has been applied on the prototype of a Motorola mobile phone during a partnership with Politecnico di Torino. We demonstrate that our approach is effective by detecting three distinct software misbehaviours that escape all traditional tests. The paper details the methodology, tests and results.

Categories and Subject Descriptors
D.2.5 [**Testing and Debugging**]: *Testing tools, Diagnostics.*

General Terms: Algorithms

Keywords: Evolutionary Algorithms, Mobile Phones, Software Testing, Power Consumption.

1. INTRODUCTION
Deep sleep is a state where a mobile phone runs only its very basic functions, waiting to be woken up by an external event. It is the state where the device is assumed to consume the least amount of energy. Misbehaviors in deep sleep mode can exhaust the battery of a mobile device without the user suspecting it.

We propose an automatic approach to software errors detection, designed to add content to an existing verification set, targeted at power related bugs. We use an Evolutionary Algorithm (EA) to generate stimuli for the mobile phone under test. Debug logs of running applications and physical measures from the device are feed back to the EA to create new stimuli.

2. PROPOSED APPROACH
There are many different system verification approaches that can be applied, including formal verification [4], constrained-random test generation [5], simulation-based and feedback-based techniques [2], and hand-written tests designed by skilled technicians that have a deep knowledge of the device. The technique used by industrial verification engineers when testing a mobile phone prototype is usually an emulation of user actions followed by measures to verify the correct behaviour of the phone.

The automatic approach proposed is model-based, since it exploits an internal model of the cell phone. However, the model is automatically extracted from a physical device, exploiting reverse-engineering techniques. Compared to other model-based techniques, the proposed one has reduced setup time, and avoids errors that could arise in the process of model building. Our approach exploits data gathered during the simulation, including both physical measures and debug logs of running applications, feeding it back to an EA that provides the necessary "intelligence" in the generation of stimuli.

The EA used is μGP^3 [1], which has been developed by the CAD Group of Politecnico di Torino and is available as a GPL tool. (http://sourceforge.net/projects/ugp3/)

Following the industrial verification engineers protocol, we map stimuli on sequences of key pressures. It is also possible to extend this approach by taking into account external elements, such as battery level and network conditions, but we choose not to, since we are specifically searching for errors that could arise due to user behaviour.

The presence of a bug cannot be expressed with a numeric value: a bug may be either detected or silent, with no intermediate possibilities. Since an EA, like real evolution, is able to work at its best if there is a slope toward local optima in the fitness landscape, we smooth the fitness landscape of our specific problem using heuristic measures. Then, we set the EA to maximize parameters that we think could lead to a quicker detection of the error, for example by rewarding with high fitness values individuals which activate more software applications.

We model the mobile phone software with a finite state machine (FSM). Each state represents a situation where a software application is idle, waiting for new commands; a transition is a command or a series of commands that connects a state to another (accessing to a different software application or using some functionalities of the same one). We dynamically create the FSM, obtaining the required information from software logs in debug mode. Not relying on a-priori knowledge helps avoiding errors that could arise in the process of modelling. During the experiments the FSM it is also used to evaluate the number of different transitions that each individual activates.[3] From the logs, we can also trace system messages regarding applications starting and closing, called "events" in the following.

GECCO'09, July 8–12, 2009, Montréal, Québec, Canada.
ACM 978-1-60558-325-9/09/07.

The fitness function takes into account three different factors. Such values are considered in hierarchical order, i.e., the *n*-th is checked only if the previous *n*-1 are equals.

1) The mean value of the current measured while the mobile phone is in deep sleep mode. To monitor power consumption we connect the phone to a special power supply.

2) The number of transitions, where a transition is defined as a passage from one state to another in the FSM.

3) The number of different events raised by the individual, with the aim to maximize messages from distinct applications, rewarding individuals which access a great number of separate programs.

Our aim is to reward series of actions that have a greater probability of triggering software errors, and detect the activation of a bug by measuring the current drain in deep sleep mode.

3. CASE STUDY

We applied the described framework to a mobile phone prototype implementing a P2K platform which had already been analyzed by verification engineers and passed all basic tests. The experiment required a phone, a radio tester, a power supply and a PC to control the instruments. An ad-hoc tool has been developed to import human-created test cases into μGP[3].

Individuals in our approach encode sequences of keys and pauses. The pressure of a key and a pause are defined as two macros, and both have only one parameter: the former the actual key pressed; the latter the length of the pause. We let μGP[3] freely manipulate and reassemble designer sequences, mixing and modifying them. Original user-defined sequence may be later used to create another test by adding, removing or changing single elements.

All the tests are performed with a population of 50 individuals, with approximately 30 new individuals created at each generation[1]. The clone scaling option of μGP[3] , which examines the genotype of each new individual in order to find out whether another identical individual already exists in the population, proved particularly useful during the experiments, because the time required to search the population for clones of an existing individual is much smaller than the time needed for a single evaluation.

Since our approach requires a great number of evaluations at each step, one of our main goals is to shorten the duration of a single test. With various improvements we reduce the test duration by 50 seconds, but the average evaluation time of an individual is still about 6 minutes and 24 seconds. The average time to complete a generation step is about 3 hours and 12 minutes.

3.1 Video recording bug

After about 16 hours of computation and 150 individuals evaluated, the majority of those showed a deep sleep current consumption between 2,5 and 3,2 mA, we obtained three individuals with a consumption of about 7,0 mA. The shortest individual consisted in about 100 lines of code, and after a deep analysis it showed that pressing specific buttons while the phone was in video recording mode caused the appearance of a warning dialog on the display and froze it completely, thus preventing the mobile phone from returning to deep sleep mode. The other two individuals shared the same sub-sequence of keys which caused the bug. After the discovery, μGP[3] constraints were modified to avoid the generation of individuals with the same pattern.

3.2 Voice call bug

The second bug was found after additional 120 hours of computation and about 1120 individuals evaluated. The best individual in the population brought the phone to an apparent deep sleep mode, but the power supply revealed a power consumption of 50 mA, which is more than 16 times the expected value in deep sleep mode. The phone did not display messages or manifested apparent abnormal behaviours. Eventually we discovered that the problem was caused by the video call button: if pressed along with a special series of keys during a normal voice call, it powered up the camera, letting it consume power as normal even when the phone switched back to deep sleep mode. After the discovery, μGP[3] constraints were modified to avoid the generation of individuals with the same pattern.

3.3 Incorrect behavior in a menu

A third problem, not related to power consumption, was discovered: by entering and exiting the menu without changing anything, the mobile phone reset some of its *settings* to their initial values, thus making the next tests fail.

4. CONCLUSIONS

An automatic, feedback-based, verification approach is proposed to attest the correct behavior of a mobile phone, exposing software errors that could eventually affect the device even after a preliminary verification performed by human beings. The framework is tested on a mobile phone prototype implementing the P2K platform, and manages to successfully locate bugs that are not detected by traditional verification. Two of them affect current consumption in deep sleep mode, thus being critical from a user's point of view.

5. AKNOWLEDGEMENTS

Our thanks to Simone Loiacono, Alessio Moscatello and Alessandro Salomone for their invaluable help.

6. REFERENCES

[1] Corno F. et al., *Evolving Assembly Programs: How Games Help Microprocessor Validation,* IEEE Transactions on Evolutionary Computation, Special Issue on Evolutionary Computation and Games, Dec. 2005, vol. 9, pp. 695-706

[2] Sanchez E. et al., *Efficient Techniques for Automatic Verification-Oriented Test Set Optimization*, IJPP, Vol. 34, Num. 1, March 2006, pp. 93 - 109, Ed. Springer Netherlands

[3] Ravotto D. et al., *An Evolutionary Methodology for Test Generation for DPeripheral Cores Via Dynamic FSM Extraction*, EvoHOT2008, March 26-28, 2008, Napoli, Italy, pp. 214-223

[4] A. Piziali, *Functional Verification Coverage Measurements and Analysis,* Kluwer Academic Publishers, 2004

[5] Jun Yuan et al., *A framework for constrained functional verification*, ICCAD-2003, pp. 142-145, 9-13 Nov. 2003

[1] The number of individuals created at each step can vary depending on the genetic operators applied by μGP[3]. Some operators can generate more than one child individual.