2013 17th European Conference on Software Maintenance and Reengineering

PINCETTE – Validating Changes and Upgrades in Networked Software

Hana Chockler, *IBM, Haifa*
Giovanni Denaro, *University of Milano-Bicocca*
Meijia Ling, *University of Oxford*
Grigory Fedyukovich, *USI, Lugano*
Antti E. J. Hyvrinen, *USI, Lugano*
Leonardo Mariani, *University of Milano-Bicocca*
Ali Muhammad, *VTT*
Manuel Oriol, *ABB Zürich*
Ajitha Rajan, *University of Oxford*
Ondrej Sery, *USI, Lugano*
Natasha Sharygina, *USI, Lugano*
Michael Tautschnig, *University of Oxford*

***Abstract*—Networked control systems ensure today's dependability of essential infrastructure such as water, electricity and transportation. Over a life cycle of tens of years, maintenance upgrades of the software running these control systems must not introduce new software errors or otherwise break existing functionality. The EU FP7 project PINCETTE addresses this challenge by employing a combination of static and dynamic software verification tools. The changes between versions are used both as guidance for improving efficiency of software verification, and also to define new notions of correctness. We report on current progress and initial validation results by our industrial partners.**

***Keywords*-Software validation and verification; software testing; change impact analysis; run-time monitoring; model checking; incremental software development**

I. Introduction

In an increasingly technology-dependent world, the reliability of networked systems in complex infrastructures such as water, electricity, transport and communication is vital. These networked systems, which monitor and control complex processes, require upgrades for maintenance and software enhancement. However, upgrades can introduce software errors, lead to the loss of certain existing functionalities and create incompatibilities between the older and newer versions. Detecting errors using test suites is time consuming, expensive, and unreliable. Yet in safety-critical applications such as aerospace, nuclear reactors and medical devices etc., introduction of errors may be fatal.

Currently, the procedure for handling software changes and upgrades in industrial settings requires an expensive revalidation of the whole system. Consequently, the cost of this validation dominates the maintenance costs of the software – with an estimate of 40% to 70% of the life-cycle costs [1]. In some cases, this means that changes are not checked thoroughly because of the high cost of full revalidation, hence potentially introducing new errors into the system design. A typical answer of a software provider on the process of understanding the impact of changes is "Normally [we] understand the impact of changes by difference between code releases or simply based on the experience of the engineer managing the code." This situation arises because the state-of-the-art testing and validation tools are not optimized to validate system changes and upgrades, but instead focus on a single program version only. The risks and cost associated with upgrades leads to conservative attitudes towards technological advancements, resulting in systems performing below acceptable levels.

The goal of PINCETTE is to produce an automated system of localised verification via a combination of static and dynamic techniques. We aim at automatically detecting, localising and repairing program bugs, resulting in improvements in software reliability and reduction in maintenance costs. The demand of such a solution for validating upgrades is increasing as the scale and complexity of networked systems increase. Improvements in the reliability and cost-effectiveness of software upgrades would allow large-scale infrastructures such as the national grid and other mission-critical software [2] to confidently apply upgrades and explore the potential of innovations.

II. Methodology

PINCETTE uses a combination of methodologies—static, dynamic and hybrid—and associated technologies to identify the impact of changes.

There are many static analysis and formal verification based tools for verification of software (see [3] for a full survey on automated formal software verification). The most relevant technique to PINCETTE is Counterexample Guided Abstraction Refinement (CEGAR) [4]. In CEGAR, an initial coarse abstraction of the system is iteratively refined, based on invalid counterexamples seen in prior model checking runs. Eventually either an abstraction proves the correctness of the system or a valid counterexample is found. CEGAR is implemented in software model checkers such as SLAM [5], BLAST [6], ComFoRT [7], or SATABS [8]. It proved to be successful in detecting errors in industrial software.

Dynamic analysis is the analysis of the properties of a running program [9]. Dynamic analysis techniques have been used since the early seventies, initially mostly for performance analysis and debugging, and later for analyzing various properties for different purposes.

1534-5351/13 $26.00 © 2013 IEEE
DOI 10.1109/CSMR.2013.72

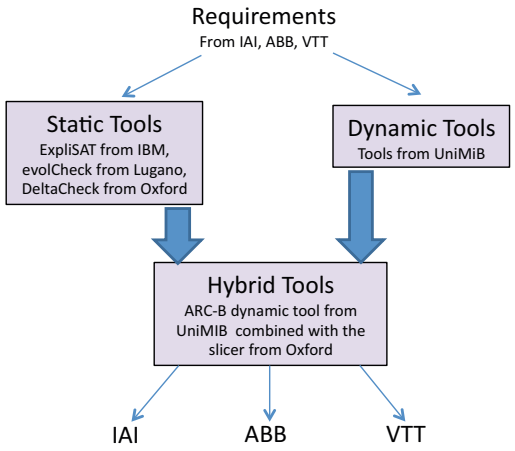

Figure 1. Overview of collaboration and tools

The combination of static and dynamic analysis techniques is an interesting field not fully explored yet. So far, static and dynamic analyses have developed as separate research domains: static analysis analyzes all possible program executions; dynamic analysis instruments the program and collects information about its executions as it runs. Traditionally, static analysis sacrifices precision to maintain soundness by overapproximating the set of real behaviours, whereas dynamic analysis is precise because it operates on real executions, however the results may not hold for all executions. Thus, static and dynamic analyses have complementary strengths and weaknesses [10].

Early work inspired by duality between dynamic and static analysis was dynamic invariant detection [11]. It led to a work that integrated dynamic detection and static verification [12]. The two papers demonstrated duality and synergy of static and dynamic analysis in program verification. In PINCETTE, we use this synergistic technique to construct automatic *regression testing* suites, where the previous version is used as a reference model. While there has been other work combining these two complementary approaches to system analysis, e.g., Direct Automated Random Testing (DART) [13], to the best of our knowledge, there has been no prior work which uses a combination of these techniques for verification of software upgrades.

Another unique aspect of the PINCETTE approach for upgrade validation is as follows: we do not revalidate the new version of the system in its entirety, instead we only verify the safety of the portion of the system affected by the upgrade and the impact of the upgrade on the new version. This is done by utilising the knowledge that the behaviour of the existing version of the system is safe. This will drastically reduce the time and effort required to revalidate the new version and result in cost savings.

III. Achievements and Outcomes

In the following we sketch the tools and methods developed by the partners of PINCETTE. They are linked together and evaluated as summarised in Figure 1.

A. Static Analysis of Software Upgrades

The static analysis technology from IBM, *ExpliSAT* [14], is adapted to focus on verification of changes while ignoring the rest of the system. The premise is that the rest of the system was known to work in production for a long time already, hence we only need to verify the behaviours of the system affected by the changes. This is done by performing an analysis of the control flow graph of the system and verifying only the paths that pass changed nodes. Experimental results on the validation cases show dramatic improvements over re-validation of the whole system.

In PINCETTE, static analysis based on symbolic model checking is addressed with *eVolCheck*, a function summary based upgrade checker [15], [16], [17]. The key idea in eVolCheck is to automatically summarize the behaviour of functions critical with respect to the correctness, by means of Craig interpolants [18]. These summaries are then used to localize the check of any new version and thus speed up the incremental analysis of upgrades.

The University of Oxford has developed *Deltacheck*, which implements a BDD-based dataflow analysis with a custom predicate abstraction engine. The key idea is that change is a dataflow predicate. DeltaCheck is designed for scalability and addresses large-scale validation cases. For instance, we successfully ran DeltaCheck on the Linux Kernel with 13.9 million LOC. Properties we can currently verify are checks for array bounds, pointer validity, division by zero, NaN on float and signed integer overflow.

B. Dynamic Analysis of Software Upgrades

The dynamic analysis technology developed in the project automatically generates models that represent the behaviour of a program version. These models are used to assess the impact and the correctness of upgrades [19]. Validation has taken place both on Linux and internal ABB code. Dynamic analysis includes and in turn exploits concolic testing to automatically generate test cases to sample the upgrade-relevant part of the state space of a program under test [20].

C. Development of Hybrid Analysis Technique

In PINCETTE we integrate dynamic and static analyses to build stronger approaches. In particular, slicing is used to restrict the scope of dynamic analysis and concolic testing to the code affected by the upgrade under consideration. Dynamic models are used as program properties that can be verified with static analysis tools. We believe this combination vastly extends the applicability of the PINCETTE approach especially when dynamic analysis is either difficult

or impossible, e.g., when the analyzed code cannot be executed outside an embedded target platform.

The static component in this combination is the slicer developed by University of Oxford. The slicer reduces the size of the program by identifying portions of the program relevant to changes. Relevant portions are identified through control and data dependency analysis. The reduced program, referred to as a slice, is extended to an executable program. The slice is then given to a dynamic analysis tool from UniMiB for monitoring and validating changes.

D. Generation and Prioritization of Test Cases

The University of Milan (UNiMiB) together with VTT developed a tool for the generation and prioritization of test cases that target controller applications implemented in LabVIEW, which is first of a kind. The current version of the tool can be used to validate stateless components and provides extensive control over the numeric inputs and outputs. The main directions of future work are extending the approach to stateful components, enriching the technique with more test case generation strategies and heuristics for test case selection and prioritization, and validating the approach with industrial case studies. The goal is to build a framework for regression testing of controller applications.

E. Validation of Results

The project results are currently undergoing validation w.r.t. industrial requirements in large-scale applications provided by our industrial partners ABB, VTT and IAI:

- Europe's power grid, the operational reliability of which is controlled by ABB's networked C/C++ code;
- VTT's real-time multi-process networked software for maintaining the thermonuclear reactor of ITER;
- real-time networked software developed by IAI for operating payloads on autonomous aircrafts for environmental monitoring, namely, the Multi-sensor Stabilized Electro-Optic System (MSEOS), widely used in Europe for detecting forest fires and locating missing people.

These wide-ranging applications demonstrate the broad applicability of PINCETTE.

IV. Project Details

PINCETTE[1] is three year STREP project under the EU 7$^{\text{th}}$ Framework Programme (FP7/2007-2003) ICT-2009.1.4 – Trustworthy ICT. It runs from July 2010 to June 2013 (36 months) with an EU contribution of €2.8 million. The consortium consists of a balanced combination of academic and industrial partners from various European countries:

- IBM Israel: Dr Hana Chockler
- University of Oxford: Prof Daniel Kroening
- Università della Svizzera Italiana: Prof Natasha Sharygina
- Università degli Studi di Milano-Bicocca: Dr Leonardo Mariani
- Technical Research Centre of Finland (VTT)
- Israeli Aerospace Industries (IAI)
- ABB, Switzerland

The academic partners and IBM act as technology providers, while ABB, IAI and VTT are validators.

[1] http://www.pincette-project.eu

V. Related Work

A. Static Analysis and Formal Verification

The application of software model checking tools to analyze software upgrades is a novel, and consequently, there is limited prior work. An earlier work was conducted by one of the project partners [21]. The results include a system upgrade checking technique to detect problems caused by a wide range of changes that could occur during system evolution, such as bug fixes, product upgrades (underlying platforms, third-party components, etc.), and changing requirements during the design process. The technique relies on learning regular sets [22] to automate this process, which is done in iterations and is based on the CEGAR framework. In parallel with PINCETTE, Microsoft Research developed *SymDiff* (Static semantic Difference) [23]. This infrastructure leverages and extends program verification to reason about program changes. SymDiff builds up on recent advances on program equivalence checking using SMT solvers and mostly focuses on the problem of inferring the conditions under which two programs are equivalent.

B. Dynamic Analysis

Several projects employed dynamic analysis techniques for checking compatibility of system upgrades [24], [25]. The most relevant technique for PINCETTE is described by Ernst et al. [25], who suggest a technique for checking compatibility of multi-component upgrades. They use dynamic monitoring to extract system models, which are subsequently evaluated using a compatibility test. Consequently, the models are restricted to input/output behaviours of the system and the actual internal component behaviours are ignored.

C. Combination of Static and Dynamic Analyses

Currently there are three industrial projects combining static and dynamic analyses, all led by Microsoft Research. These implement variants and extensions of a directed search: (1) SAGE (Scalable, Automated, Guided Execution) is a tool that uses instruction-level tracing for white-box fuzzing of Windows applications [26]. SAGE implements a directed search algorithm that maximizes the number of new inputs generated from each symbolic execution. (2) PEX (Program Exploration) is a tool that helps developers to write parametrised unit tests [27]. For each unit test, PEX uses dynamic test generation techniques to compute input values that exercise all statements and assertions in the program. (3) Yogi is a tool that combines testing and static analysis (as in [28] to check properties of Windows device drivers [29]).

D. Related EU Projects

SecureChange works on security of evolving systems. It focuses on potential security problems caused by software changes. However, it does not address problems of functional correctness and concurrency issues of networked systems and its techniques only apply to the abstract models of software, while PINCETTE supports reasoning also at the source level (C, C++) of networked software.

VI. Conclusion

The PINCETTE project provides significant advances in the field of verification and validation of evolving systems. The techniques developed in the course of this project first increase the confidence in software upgrades by employing automated formal verification techniques. Second, the cost of software maintenance is further reduced as these methods are built to scale to large industrial code bases by being guided by the actual changes.

References

[1] P. A. Grubb and A. A. Takang, *Software maintenance – concepts and practice (2. ed.)*. World Scientific, 2003.

[2] P. Farries and A. Rajan, "Pincette – validating changes and upgrades in networked software," *ERCIM News*, vol. 2012, no. 88, 2012.

[3] V. D'Silva, D. Kroening, and G. Weissenbacher, "A survey of automated techniques for formal software verification," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1165–1178, 2008.

[4] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *CAV*, 2000, pp. 154–169.

[5] T. Ball, R. Majumdar, T. D. Millstein, and S. K. Rajamani, "Automatic predicate abstraction of C programs," in *PLDI*, 2001, pp. 203–213.

[6] T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre, "Software verification with BLAST," in *SPIN*, 2003, pp. 235–239.

[7] S. Chaki, J. Ivers, N. Sharygina, and K. C. Wallnau, "The comfort reasoning framework," in *CAV*, 2005, pp. 164–169.

[8] E. M. Clarke, D. Kroening, N. Sharygina, and K. Yorav, "SATABS: Sat-based predicate abstraction for ANSI-C," in *TACAS*, 2005, pp. 570–574.

[9] T. Ball, "The concept of dynamic analysis," in *ESEC / SIGSOFT FSE*, 1999, pp. 216–234.

[10] M. D. Ernst, "Invited talk static and dynamic analysis: synergy and duality," in *PASTE*, 2004, p. 35.

[11] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin, "Dynamically discovering likely program invariants to support program evolution," *IEEE Trans. Software Eng.*, vol. 27, no. 2, pp. 99–123, 2001.

[12] J. W. Nimmer and M. D. Ernst, "Static verification of dynamically detected program invariants: Integrating Daikon and ESC/Java," *ENTCS*, vol. 55, no. 2, pp. 255–276, 2001.

[13] P. Godefroid, N. Klarlund, and K. Sen, "DART: directed automated random testing," in *PLDI*, 2005, pp. 213–223.

[14] H. Chockler and S. Ruah, "Verification of software changes with ExpliSAT," in *HotSWUp*, 2012, pp. 31–35.

[15] O. Sery, G. Fedyukovich, and N. Sharygina, "Interpolation-based function summaries in bounded model checking," in *HVC*, 2012, pp. 160–175.

[16] ——, "Incremental upgrade checking by means of interpolation-based function summaries," in *FMCAD*, 2012, to appear.

[17] G. Fedyukovich, O. Sery, and N. Sharygina, "Function summaries in software upgrade checking," in *HVC*, 2012, pp. 257–258.

[18] W. Craig, "Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory," *Journal of Symbolic Logic*, vol. 22, no. 3, pp. 269–285, 1957.

[19] F. Pastore, L. Mariani, A. Goffi, M. Oriol, and M. Wahler, "Dynamic analysis of upgrades in C/C++ software," in *ISSRE*, 2012, to appear.

[20] M. Baluda, P. Braione, G. Denaro, , and M. Pezzé, "Enhancing structural software coverage by incrementally computing branch executability," *Software Quality Journal*, vol. 19, no. 4, 2011.

[21] N. Sharygina, S. Chaki, E. M. Clarke, and N. Sinha, "Dynamic component substitutability analysis," in *FM*, 2005, pp. 512–528.

[22] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, 1987.

[23] S. K. Lahiri, C. Hawblitzel, M. Kawaguchi, and H. Rebêlo, "Symdiff: A language-agnostic semantic diff tool for imperative programs," in *CAV*, 2012, pp. 712–717.

[24] S. McCamant and M. D. Ernst, "Early identification of incompatibilities in multi-component upgrades," in *ECOOP*, 2004, pp. 440–464.

[25] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin, "Dynamically discovering likely program invariants to support program evolution," in *ICSE*, 1999, pp. 213–224.

[26] P. Godefroid, M. Y. Levin, and D. A. Molnar, "Active property checking," in *EMSOFT*, 2008, pp. 207–216.

[27] N. Tillmann and W. Schulte, "Parameterized unit tests," in *ESEC/SIGSOFT FSE*, 2005, pp. 253–262.

[28] N. E. Beckman, A. V. Nori, S. K. Rajamani, and R. J. Simmons, "Proofs from tests," in *ISSTA*, 2008, pp. 3–14.

[29] A. V. Nori, S. K. Rajamani, S. Tetali, and A. V. Thakur, "The Yogi project: Software property checking via static analysis and testing," in *TACAS*, 2009, pp. 178–181.