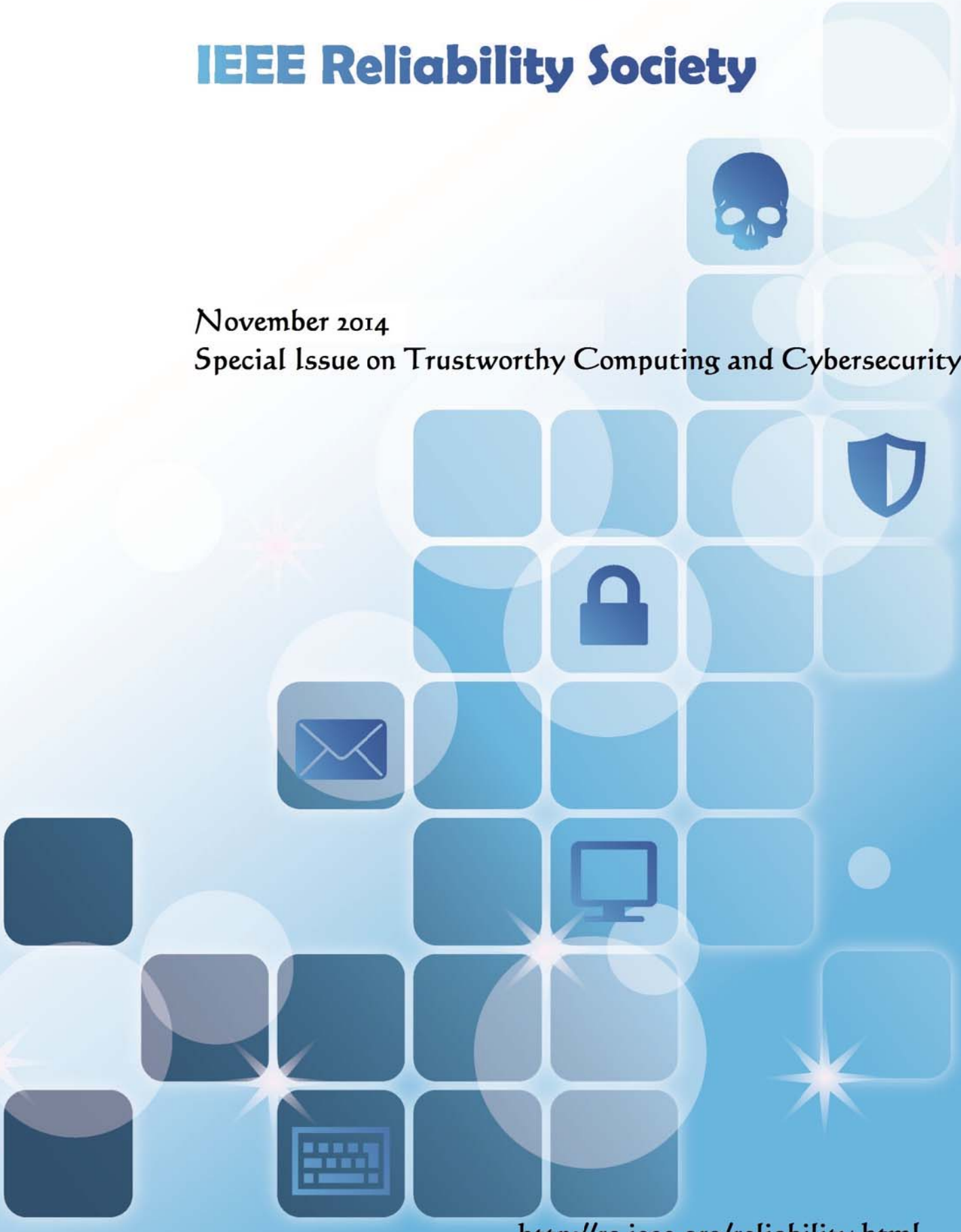
IEEE

Reliability

IEEE Reliability Society



November 2014
Special Issue on Trustworthy Computing and Cybersecurity

http://rs.ieee.org/reliability.html

Reliability Society Administrative Committee (AdCom) Member

OFFICERS(EXCOM)

President
Christian Hansen

Sr. Past President
Jeffrey Voas

Jr. Past President
Dennis Hoffman

Vice Presidents

Technical Activities
Shiuhpyng Winston Shieh

Publications
Phil Laplante

Meetings and Conferences
Alfred Stevens

Membership
Marsha Abramo

Secretary
W. Eric Wong

Treasurer
Bob Loomis

ELECTED MEMBERS-AT-LARGE (WITH VOTE)

TERM EXPIRES 2014 (DEC 31)

Scott B. Abrams
Pierre Dersin
Wei Dong
Carole Graas
Samuel J. Keene
Phil LaPlante

TERM EXPIRES 2015 (DEC 31)

Marsha Abramo
Loretta Arellano
Lon Chase
Shiuhpyng Winston Shieh
Alfred Stevens
Rex Sallade

TERM EXPIRES 2016 (DEC 31)

Lou Gullo
Christian Hansen
Pradeep Lall
Zhaojun (Steven) Li
Bob Loomis
Pradeep Ramuhalli

STANDING COMMITTEES AND ACTIVITIES/INITIATIVES

Web Master
Lon Chase

Standards
Lou Gullo

Chapters Coordinator
Loretta Arellano

Professional Development
Marsha Abramo

Constitution and Bylaws
Dennis Hoffman

Fellows
Sam Keene

Finance
Christian Hansen

Academic Education/Scholarship
Sam Keene

Meetings Organization
Alfred Stevens

Membership Development
Marsha Abramo

Transactions Editor
Way Kuo

Newsletter Editor
Joe Childs

Video Tutorials
Christian Hansen

Nominations and Awards
Jeffrey Voas

Life Science Initiative
Peter Ghavami

Transportation Electrification
Sam Keene
Pradeep Lall
Michael Austin

IEEE Press Liaison
Dev Raheja

IEEE GOVERNMENT RELATIONS COMMITTEES

Energy Policy
Jeff Voas

Transportation and Aerospace Policy
Scott Abrams

Medical Technology Policy
Jeff Voas

Critical Infrastructure Protection
Sam Keene

Career and Workforce Policy
Christian Hansen
(corresponding only)

Intellectual Property Committee
Carole Graas
(corresponding only)

Research and Development Committee
Pradeep Lall

Combined TAB Ad Hoc Committee on Attracting Industrial Members
Dennis Hoffman

2013 TAB Awards and Recognition Committee (TABARC)
Dennis Hoffman

Technical Committees

Technical Committee on Internet of Things (IoT)

Chair: Jeffrey M. Voas, National Institute of Standards and Technology
jeff.voas@nist.gov
Co–chair: Irena Bojanova, University of Maryland University College, USA
irena.bojanova@umuc.edu

Committee Member:

1. George F. Hurlburt: CEO of Change Index
2. Irena Bojanova: Program Director, Telecommunications Management, and Collegiate Professor, University of Maryland, University College

Technical Committee on System and Software Assurance

Chair: Eric Wong, University of Texas at Dallas
Email: ewong@utdallas.edu

Technical Committee on Prognostics and Health Management (PHM)

Chair: Rex Sallade, Sikorsky Aircraft Co.
Email: Rex.Sallade@SIKORSKY.COM
Co–chair: Pradeep Lall, Auburn University
Email: lall@eng.auburn.edu

Technical Committee on Trustworthy Computing and Cybersecurity

Chair: Wen-Guey Tzeng, National Chiao Tung University
Email: wgtzeng@cs.nctu.edu.tw

Committee Member:

1. Raul Santelices: Assistant Professor, Department of Computer Science and Engineering, University of Notre Dam
2. Brahim Hamid: Associate Professor, IRIT Research Laboratory, University of Toulouse, France

Technical Committee on Reliability Science for Advanced Materials & Devices

Chair: Carole Graas, Colorado School of Mine
& IBM Systems and Technology Group
Email: cdgraas@mines.edu

Technical Committee on Systems of Systems

Chair: Pierre Dersin, Alstom Transport
Email: pierre.dersin@transport.alstom.com

Standards Committee

Chair: Louis J Gullo, IEEE RS Standards

Email: Lou.Gullo@RAYTHEON.COM

Committee Member:

1. Ann Marie Neufelder: Softrel, LLC – Owner; IEEE P1633 Standard Working Group Chair
2. Lance Fiondella: Assistant Professor, Dept. of Electrical and Computer Engineering, University of Massachusetts Dartmouth; IEEE P1633 Standard Working Group Vice Chair
3. Steven Li: Assistant Professor, Industrial Engineering and Engineering Management, Western New England University; IEEE P61014 Standard Working Group Chair
4. Diganta Das: Research Staff at the Center for Advanced Life Cycle Engineering, University of Maryland
5. Sony Mathews: Engineer at Halliburton, IEEE P1856 Standard Working Group Chair
   Mike Pecht: Chair Professor and Director of Center for Advanced Life Cycle Engineering, University of Maryland
6. Arvind Sai Sarathi Vasan: Research Assistant at Center for Advanced Life Cycle Engineering , University of Maryland; IEEE P1856 Standard Working Group Vice-Chair
7. Joe Childs: Staff Reliability/Testability Engineer at Lockheed Martin

Working Group on Education

Chair: Zhaojun (Steven) Li, Western New England University

Email: zhaojun.li@wne.edu

Committee Member:

1. Emmanuel Gonzalez, Jardine Schindler Elevator Corporation

Editorial Board

President's Message

I am very excited to introduce to you Reliability Digest, a new technical feature of the Reliability Society Newsletter. I congratulate our VP of Technical Activities Dr. Shiuhpyng Shieh and his team for making this online magazine possible, and I thank the technical experts who contributed articles to this special issue on trustworthy computing. With Reliability Digest the Reliability Society is bridging the gap between the highly technical articles published in the society sponsored scholarly journals and the less technical announcements traditionally published in our newsletter.

This inaugural issue is offered to Reliability Society members as part of their membership benefits and, at least for a limited time, to the general public free of charge. Read the articles online, print them, or download them to your tablet or smart phone. No matter which way you choose to take the magazine with you, I hope you enjoy the technical content brought to you by your Reliability Society.

Editor-in-Chief's Message for the Inaugural Issue

Warm Welcome to the readers and authors of IEEE Reliability Digest!

It is my pleasure to introduce the inaugural issue of IEEE Reliability Digest. In this special issue on trustworthy computing and cybersecurity, Area Editor Wen-Guey Tzeng brings you the recent progress on this cutting edge technology. Dr. Wen-Guey Tzeng is an expert in the field of trustworthy computing and cryptography. Hope you all enjoy reading this very interesting issue.

IEEE Reliability Digest is a peer-reviewed online magazine of the IEEE Reliability Society, published quarterly in February, May, August, November, respectively. It addresses reliability, security and related assurance topics, including recent innovations and advancements of emerging hot topics as well as new techniques and methodologies, experiences and practices, case studies and benchmarks. Its scope covers but is not limited to cybersecurity, software assurance, intrusion detection, computer and network security, PHM (Prognostics and Health Management), reliability science for advanced materials and devices, big data, IoT (Internet of Things), mobile and cloud computing.

As the Editor-in-Chief, I would like to express my sincere gratitude to the area editors, authors and reviewers who devoted their time and effort in launching the first issue of Reliability Digest. The online magazine is intended to serve as a forum for scholars to exchange their opinions.

You are very welcome to submit your articles to Reliability Digest. Only with your support and assistance can Reliability Digest better serve our community. Your comments and feedback will be greatly appreciated.

Enjoy Reading.

Shiupyng Winston Shieh
Editor-in-Chief and VP Tech
ssp@cs.nctu.edu.tw

Special Issue Editor's Message

Preface

Trustworthy Computing is a broad concept and has been around since the introduction of computers to our society. Since computers become an essential part of our daily life, people would like them to be as trusted as utility supplies. A trustworthy system should meet people's expectation of achieving its goals without failing during its lifetime. Down to the engineering aspects, trustworthy computing consists of reliable hardware, robust and trusted software, security and privacy mechanisms, etc. Each aspect needs tremendous devotion of technological works to make it trustworthy.

In 2002, Microsoft created the Trustworthy Computing Initiative (TwC). It identifies four key areas: security, privacy, reliability and business integrity for its software development and services. We have seen strong momentum and much progress in pursuing these goals since then. In 2012, after 10 years of the Initiative, Microsoft retrospected its work on Trustworthy Computing and announced that it is more important than ever to provide trustworthy systems as computers are even more deeply embedded into our daily life than ever. Smart devices, cloud computing, Internet of things, big data, etc., all need to be trustworthy.

The Reliability Society recognizes the need of trustworthy computing and establishes Technical Committee on Trustworthy Computing and Cybersecurity. The first issue of Reliability Digest is devoted to this topic. The call-for-paper/article was announced on the website of the Reliability Society and sent to prospective scholars and researchers. We received very positive response of submission. Each submission is reviewed by at least three reviewers. We finally choose six papers for this issue.

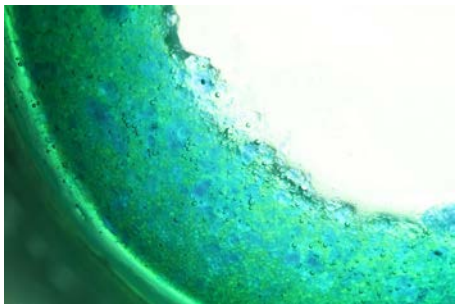
It is not possible to cover all aspects of trustworthy computing and provide complete treatment in an issue of Reliability Digest. We hope that the readers can piece together information from these papers and other sources to shape your own viewpoints of trustworthy computing and contribute in the future issues of Reliability Digest.

Wen-Guey Tzeng
Chair, Technical Committee on Trustworthy Computing and Cybersecurity
Professor, National Chiao Tung University

Reliability Digest
IEEE Reliability Society

Special Issue on Trustworthy Computing and Cybersecurity

Modeling of Secure and Dependable Applications Based on a Repository of Patterns: The SEMCO Approach



Brahim Hamid
IRIT- University of Toulouse
118 Route de Narbonne, 31062 Toulouse Cedex 9, France
hamid@irit.fr

***Abstract*—The requirement for higher quality and seamless development of systems is continuously increasing, even in domains traditionally not deeply involved in such issues. Security and Dependability (S&D) requirements are incorporated to an increasing number of systems. These newer restrictions make the development of those systems more complicated than conventional systems. In our work, we promote a new approach called SEMCO (System and software Engineering with Multi-COncerns) combining Model-Driven Engineering (MDE) with a model-based repository of S&D patterns to support the design and the analysis of pattern-based secure and dependable system and software architectures.**

The modeling framework to support the approach is based on a set of modeling languages, to specify security and dependability patterns, resources and a set of property models, and a set of model transformation rules to specify some of the analysis activities. As part of the assistance for the development of S&D applications, we have implemented a tool-chain based on the Eclipse platform to support the different activities around the repository, including the analysis activities. The proposed approach was evaluated through a case study from the railway domain.

Keywords*— *Security, Dependability, Resource, Pattern, Model-driven Engineering, Embedded Systems Engineering.

I. Introduction

During the last decades, the systems have grown with an increasing in terms of complexity and connectivity. In the past Security and Dependability (S&D) was not such a critical concern of system development teams, since it was possible to rely on the fact that a system could be easily, controlled due to its limited connectivity and, in most of the cases, its dedicated focus. However, nowadays, systems are growing in terms of complexity, functionality and connectivity not only in safety-critical areas (defense, nuclear power generation, etc.), but also in areas such as finance, transportation, medical information management and system using web applications.

Just consider Resource Constrained Embedded Systems (RCES) [1] and their added complexity and connectivity. The aforementioned challenges in modern system development push the Information and Communication Technologies (ICT) community to search for innovative methods and tools for serving these new needs and objectives. Regarding system security and dependability, in the cases of modern systems, the "walled-garden" paradigm is unsuitable and the traditional security and dependability concepts are ineffective, since it was based on the fact that it is possible to build a wall between the system and the outer world. In our opinion the foundation for comprehensive security and dependability engineering is a deep understanding of the modern systems, ongoing or previous security and dependability incidents and their implications on the underlying critical infrastructure.

The industrial context conducting our work is how to take into account several constraints, mainly those related to security and dependability, that are not satisfied by the well-known and the widely used technology for building applications for Resource-Constrained Embedded Systems. These requirements introduce conflicts on the three main factors that determine the cost of ownership of applications: (a) cost of the production, (b) cost of engineering and (c) cost of maintenance. In other words, systems with high dependability requirements for which the security level must be demonstrated and certified use almost exclusively technical solutions strongly oriented by the application domains. Applications based on these solutions are by definition dedicated, hardly portable between different execution platforms and require specific engineering processes. These specificities greatly increase the cost of the development in the different phases of their lifecycle.

Embedded systems share a large number of common characteristics, including real-time and physical constraints (e.g. temperature), as well as energy efficiency requirements. Specifically, Resource Constrained Embedded Systems refer to systems which have memory and/or computational processing power constraints computing resources of RCES, e.g. memory, tasks, and buffers, are generally statically determined. The generation of RCES therefore involves specific software building processes. These processes are often error-prone because they are not fully automated, even if some level of automatic code generation or even model driven engineering support is applied. Furthermore, many RCES also have assurance requirements, ranging from very strong levels involving certification (e.g. DO178 and IEC-61508 for safety-relevant embedded systems development) to lighter levels based on industry practices. Consequently, the conception and design of RCES is an inherently complex endeavor. To cope with the growing complexity of embedded system design, several development approaches have been proposed. The

most popular are those using models as main artifacts to be constructed and maintained.

In embedded system design field, the integration of non-functional requirements from Security and Dependability (S&D) [2], [3] are exacerbating this complexity, mainly in the context of trade-offs. For instance, in the automotive domain, a car may need to have secure communication mechanisms for secure download or for secure data transfer. In the railway domain, a supervision system needs to have secure communication mechanisms to be able to activate the emergency brake when something goes wrong. The development of systems with security and dependability requires specialized expertise and skills. The cost of designing such features from scratch could easily exceed the cost of the rest of the system. For example (see Fig. 1), the development of a security component for a railway signaling system requires expertise in security that is seldom available in the railway industry and it requires expertise in the engineering process and validation practices of railway which are not always available in the S&D community. In fact, capturing and providing this expertise by the way of security and dependability patterns can support the integration of S&D features by design to foster reuse during the process of software system development. Patterns are specified and validated by security and dependability experts and stored in a repository to be reused as security and dependability building block function by software engineer in several domains.

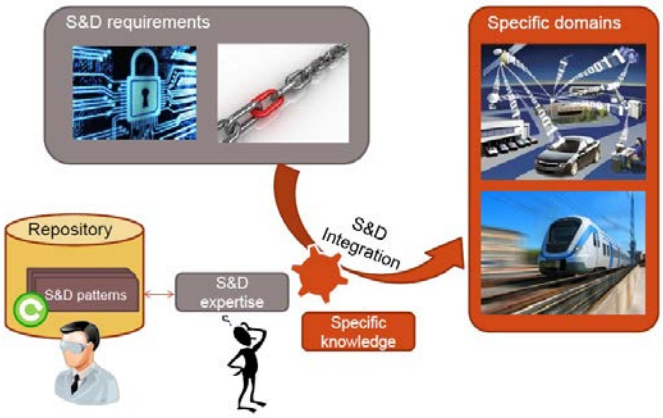


Fig. 1. Patterns for engineering systems with security and dependability requirements

Recent times have seen a paradigm shift in terms of design by combining multiple software engineering paradigms, namely, Model-Driven Engineering (MDE) [4] and Component Based Software Engineering (CBSE) [5]. Such a paradigm shift is changing the way systems are developed nowadays, reducing development time significantly.

In our work, we promote a new discipline for systems engineering around a model-based repository of modeling artifacts using a pattern as its first class citizen: *Pattern-based System Engineering (PBSE)*. The proposed approach called *SEMCO* for System and software Engineering with Multi-COncern [1] addresses two kinds of processes: the process of modeling artifacts development and system development with modeling artifacts. Therefore, we add a repository as a tier which acts as intermediate agent between these two processes. A repository should provide a modeling container to support modeling artifacts lifecycle associated with different methodologies. The patterns that are at the heart of our system engineering process reflect design solutions at domain independent and specific level, respectively.

II. BACKGROUND

A. Incorporating Security and Dependability in System Engineering

In system engineering, security and dependability may be compromised in several system layers. Usually, security is considered when design decisions are made leading to potential conflicting situations. The integration of security and dependability features requires the availability of system architecture expertise, application domain specific knowledge and security expertise at the same time to manage the potential consequences of design decisions on the security of a system and on the rest of the architecture. For instance, at architectural level, incorporating security means to have a mechanism (it may be a component or integrated into a component). Development processes for system and software construction are common knowledge and mainstream practice in most development organizations. Unfortunately, these processes offer little support in order to meet security and dependability requirements. Over the years, research efforts have been invested in methodologies and techniques for secure and dependable software engineering, yet dedicated processes have been proposed only recently, namely OWASP's CLAS[2], Microsoft's SDL[3] and McGraw's Touchpoints[4].

In *SEMCO*, our aim is (1) to identify the commonalities, discuss the specificity of each approach, and (2) to evaluate the integration of the *SEMCO* outcomes in these process models. The overall goal in *SEMCO* is to support any security and dependability engineering process.

B. Pattern-Based Development

Patterns are widely used today to provide architects and designers with reusable design knowledge. They refer to triples that describe solutions for commonly occur- ring problems in specific contexts. There are patterns for generic architecture problems [6], for security [7] and for other non-functional requirements.

Pattern-based development has gained more attention recently in software engineering by addressing new challenges that are not targeted in the past. In fact, they are applied in modern software architecture for distributed systems including middlewares [8], and real-time embedded systems [9], and recently in security and dependability engineering [7]. The related approaches promote the use of patterns in the form of reusable design artifacts.

The supporting research activities in PBSE examine three distinct challenges: (a) mining (discovering patterns from existing systems), (b) hatching (selection of the appropriate pattern); (c) application (effective use during the system development process). These three challenges often involve

[1] http://www.semcomdt.org/

[2] http://www.owasp.org

[3] http://msdn2.microsoft.com/en-us/library/ms995349.aspx

[4] http://www.swsec.com/resources/touchpoints/

widely diverse core expertise including formal logic, mathematics, stochastic modeling, graph theory, hardware design and software engineering.

In our work, we study only the two last challenges, targeting the (i) development of an extendible design language for modeling patterns for secure and dependable distributed embedded systems [10] and (ii) a methodology to improve existing development processes using patterns [11]. The language has to capture the core elements of the pattern to help its (a) precise specification, (b) appropriate selection and (c) seamless integration and usage. The first aspect is pattern-definition oriented while the second and the third aspects are more problem-definition oriented.

Usually, these design artifacts are provided as a library of models (sub-systems) and as a system of patterns (framework) in the more elaborated approaches. However, there are still lacks of modeling languages and/or formalisms dedicated to specify these design artifacts and the way how to reuse them in software development automation. More precisely, a gap between the development of systems using patterns and the pattern information still exists.

The *SEMCO* vision is to use S&D and architecture patterns and their interactions as parameters for the computation, analysis, selection and development of secure and dependable system and software architectures.

C. Model Driven Engineering (MDE)

MDE has the potential to greatly ease daily activities of S&D experts. In fact, MDE supports the designer to specify in a separate way S&D requirements issues at a greater abstraction level. MDE promotes models as first class elements. A model can be represented at different levels of abstraction and the MDE vision is based on (1) the metamodeling techniques to describe these models and (2) the mechanisms to specify the relations between them. Model exchange is within the heart of the MDE methodology as well as the transformation/refinement relation between two models. Domain Specific Modeling Languages (DSML) [12] in software engineering is used as a methodology using models as first class citizens to specify applications within a particular domain. There are several DSML environments, one of them being the open- source Eclipse Modeling Framework (EMF) [13]. EMF provides an implementation of EMOF (Essential MOF), a subset of the Meta Object Facility (MOF)[5], called Ecore. EMF offers a set of tools to specify metamodels in Ecore and to generate other representations of them. Query View Transformation (QVT)[6] is a standard to specify model transformations in a formal way, between metamodels conforming to MOF.

In the context of *SEMCO*, design decisions are one of the most important artefacts during architecting. Models of both security and dependability decisions and other architecture concerns decisions need to be complete, and need to be specified precisely and traced to other models. The *SEMCO* vision is that metamodeling and model transformation, within MDE (specification, design, analysis, implementation, test), allows reducing time/cost of understanding and analyzing system artefacts description due to the abstraction mechanisms and reducing the cost of development process thanks to the generation mechanisms.

III. THE SEMCO APPROACH

A. Objectives

SEMCO (System and software Engineering with Multi-COncern) aims at developing a model and pattern-based modeling framework for handling security and dependability for system and software architecture that semi-automatically supports the analysis and evaluation of secure and dependable architectures for verification and validation purposes, providing the subsequent re-design that optimizes both.

The framework provides several artifacts types representing different engineering concerns (Security, Dependability, Safety and Resources) and architectural information. These artifacts are stored in a model-based repository and provided in the form of modeling languages (*SEMCO-ML*), tools (*SEMCO-MDT*) and methods (*SEMCO-PM*). The nearest goal of *SEMCO* is going to contribute on “Understanding System and software Engineering with security and dependability features by design in resource constrained systems”.

B. Our Approach Through an Example: The Stakeholders

We propose a solution based on the reuse of software subsystems that have been pre-engineered to adapt to a specific domain. In order to understand our security and dependability engineering framework with patterns better, we provide a description of a one usage scenario.

In the example of Fig. 2, first a security expert develops a security subsystem called pattern. The security expert focuses mainly on security solution development in the form of patterns elements or mining patterns from existing systems. Next a software engineering expert adapts the pattern to engineering reuse. The main output of this activity is a specification of a pattern in suitable format for repository storage, to enforce reuse during system and software development processes. The activity of creating the blue artifacts is performed by the software engineer in collaboration with the security expert. The achieved role can be called a security and dependability pattern engineer. Then a domain process expert, for instance a railway domain expert adapts the security pattern into a version that is usable in the railway system development process, ensuring compliance of these artifacts with appropriate standards and that other engineering artifacts are available throughout each phase of a development process, creating the red artifacts.

Moreover, with the help of a software engineering expert, the pattern and its associated artifacts should be transformed into a version (green artifacts) that is adapted to the railway development environment. The activity of reusing the red artifacts is performed by dedicated tools that are customized for a given software engineering environment (development platform). Finally, a domain engineer, for instance a railway system and software developer reuses the resulting adapted and transformed pattern (green artifacts) to develop a railway system.

[5] http://www.omg.org/spec/MOF/
[6] http://www.omg.org/spec/QVT/

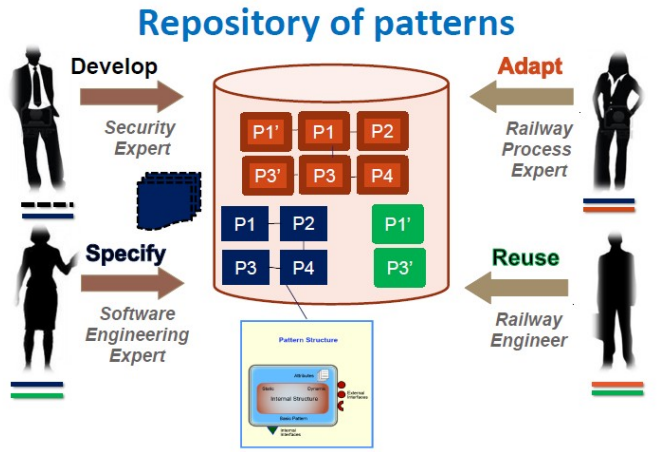


Fig.2. Our approach through an example: The Stakeholders

C. *Conceptual*

The *SEMCO* foundation is a model-based repository of modeling artifacts, including pattern, resource and property models and thereby supporting a pattern-based development methodology. The pattern is the first class citizen of these modeling artifacts to describe security and dependability solutions. The resource will capture the computing system platform and the property will allow to govern the use of patterns and to evaluate their security level for analysis for reuse. Specifically, we tend to overlook the three rules that govern pattern-based system development (1) the specification of these artifacts at different levels of abstraction, (2) the specification of relationships that govern their interactions and complementarity and the specification of the relationship between patterns and other artifacts manipulated during the development lifecycle and those related to the assessment of critical systems. It is a good application and promotion of model- driven engineering.

In *SEMCO*, a pattern is a subsystem dedicated to security and dependability aspects [14], to be specified by a security and dependability experts, and reused by domain engineers to improve systems/software engineering facing security and dependability requirements.

The core of *SEMCO* is a set of DSMLs, a repository, search and transformations engines. The DSMLs are devoted to specify patterns, a system of patterns and a set of models to govern their use, and thereby to organize, analyze, evaluate and finally validate the potential for reuse. In order to enforce reuse and to interconnect the process of the specification of these modeling artifacts and the system development with these artifacts, we developed a structured model-based repository to store these artifacts. Therefore, instead of defining new modeling artifacts, that usually are time and effort consuming as well as error prone, the system developer merely needs to select appropriate patterns from the repository and integrate them in the system under development. This is the role of search and transformation engines, where an artifact is identified/ selected from a repository and then the results are transformed towards specific domain development environments such as UML.

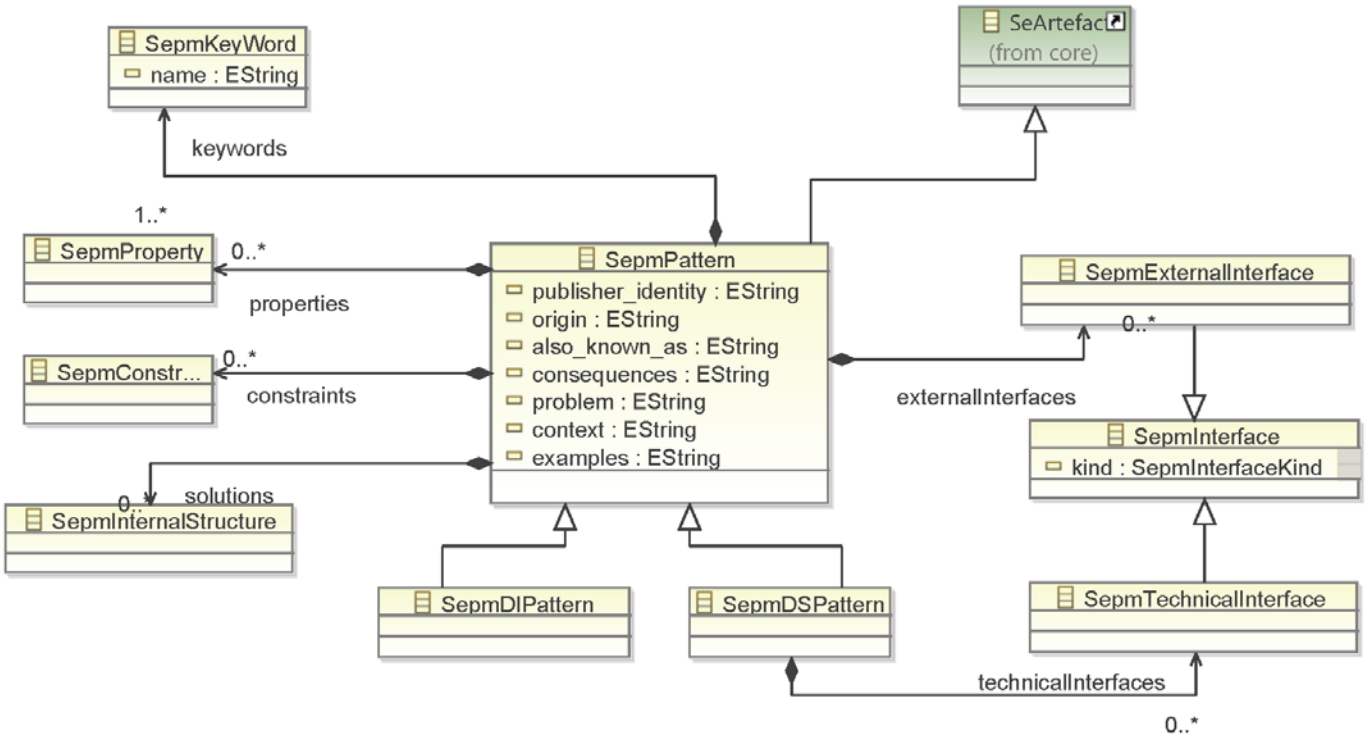


Fig. 3. The SEPM metamodel -Overview

IV. TECHNICAL FOUNDATION

The *SEMCO* vision is to create an integrated set of software tools to enable S&D RCES applications development by design, with the following objectives:

- The tools will improve the design, implementation, configuration and deployment of S&D RCES applications through:
  - Best-practice design methods: patterns and models
  - Innovative modeling and optimization techniques: Model Driven Engineering (MDE), Domain Specific Modeling Languages (DSML),
  - To foster reuse in multiple domains: repository.
- The tools will target multiple stakeholders in the RCES markets.
- The tools will provide and manage all interfaces with a common and evolving underlying core models and technologies.

As introduced in the previous section, security and dependability pattern engineer and the domain specific engineer use a number of tools. Those tools, based on model driven engineering techniques to create and then to reuse information that is stored in an engineering repository.

We now present an overview of our modeling framework building process as:

- *SEMCO-ML*: a set of DSMLs for the specification of the *SEMCO* modeling artifacts.
- *SEMCO-MDT*: a set of tools to support the *SEMCO* methods, the specification of the *SEMCO* modeling artifacts and the repository system.
- *SEMCO-PM*: a set of methodologies for the description of the PBSE methods.

Additional and detailed information will be provided during the implementation of the related design environment. Then, we detail the description of the integrated process used for the development of the Safe4Rail application in Section V.

A. SEMCO-ML

To foster reuse of patterns in the development of critical systems with S&D requirements, we are building on a metamodel for representing S&D patterns in the form of subsystems providing appropriate interfaces and targeting S&D properties. Interfaces will be used to exhibit the pattern's functionality in order to manage its application. In addition, interfaces support interactions with security primitives and protocols, such as encryption, and specialization for specific underlying software and/or hardware platforms, mainly during the deployment activity.

The *System and software Pattern Metamodel (SEPM)* is a metamodel defining a new formalism for describing patterns, while the *Generic Property Metamodel (GPRM)* is used to specify property model libraries to define the S&D and resource properties of the patterns. The principal classes of the SEPM metamodel are described with the Ecore notation in Fig. 3. In the following, we detail the meaning of principal concepts used to edit a pattern.

- *SepmPattern*. This block represents a modular part of a system representing a solution of a recurrent problem. A *SepmPattern* is defined by its behavior and by its provided and required interfaces. A *SepmPattern* may be manifested by one or more artifacts, and in turn, that artifact may be deployed to its execution environment. The *SepmPattern* has attributes to describe the related recurring design problem that arises in specific design contexts.

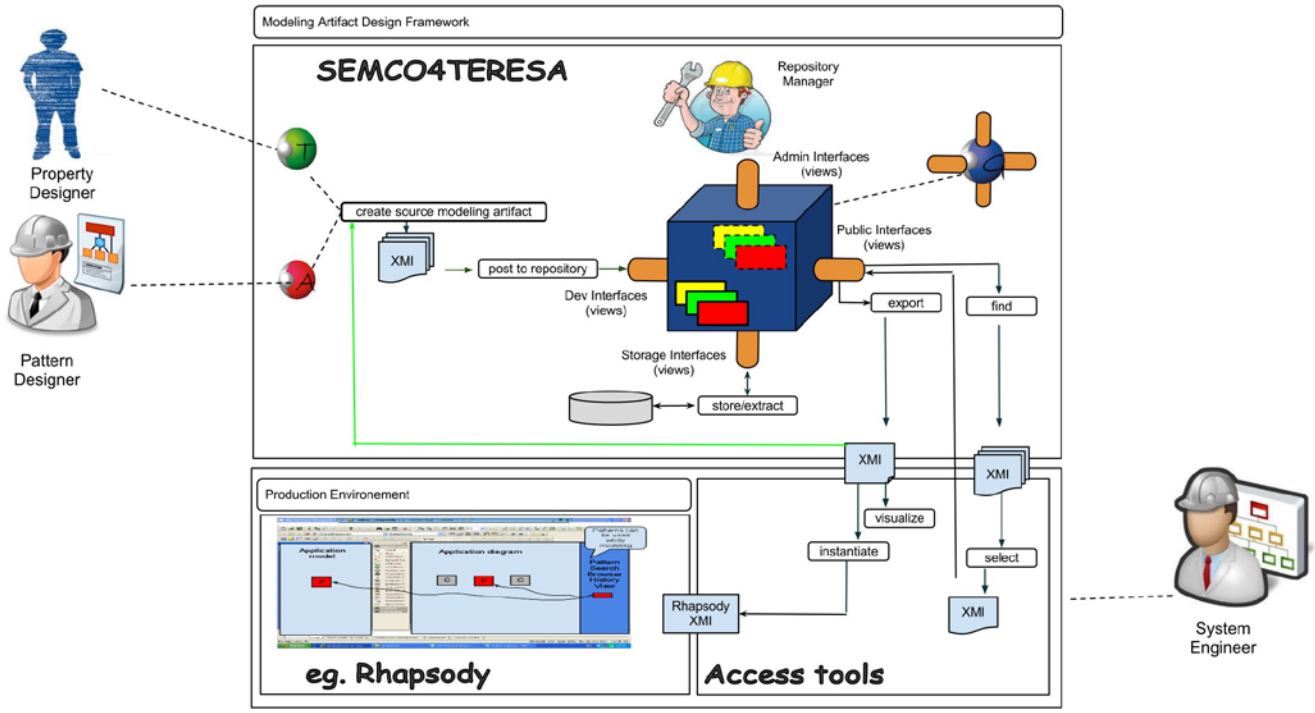


Fig. 4. The tool flow architecture

- *SepmInternalStructure.* Constitutes the implementation of the solution proposed by the pattern. Thus *the InternalStructure* can be considered as a white box which exposes the details of the pattern.
- *SepmInterface.* A pattern interacts with its environment with *Interfaces* which are composed of *Operations*. We consider two kinds of interface:
  (1) *SepmExternalInterface* for specifying interactions with regard to the integration of a pattern into an application model or to compose patterns, and
  (2) *SepmTechnicalInterface* for specifying interactions with the platform.
- *SepmProperty*. Is a particular characteristic of a pat- tern related to the concern dealing with and dedicated to capture its intent in a certain way. Each property of a pattern will be validated at the time of the pattern validation process and the assumptions used will be compiled as a set of constraints which will have to be satisfied by the domain application.

In addition to defining pattern artifacts, our pattern metamodel provides a way to formalize an *S&D Pattern System*, as a set of S&D patterns and their potential relationships, that enables an incremental support of our PBSE framework.

The *Generic Property (GPRM)* metamodel captures the common concepts of the two main concerns of trusted RCES applications: Security, Dependability and Resource on the one hand and Constraints on these properties on the other hand. The libraries of properties and constraints include units, types and categories. For instance, security and dependability attributes [5] such as authenticity, confidentiality and availability are categories of S&D properties. These categories require a set of measures types (degree, metrics…) and units (boolean, float…).

B. SEMCO-MDT

The tool-suite to support the *SEMCO* approach has to provide the following features:

- Repository life cycle: Allow the management of the repository, including deployment, set-up and organization.
- Modeling artefact life cycle: Provide the ability to editing S&D patterns and models, their validation, and their deposit in repository (DEP).
- System life cycle: Provide the ability to retrieve S&D patterns and models from repository (RET) by querying the repository, instantiate and integrate the results.

Using the proposed metamodels, the Eclipse Modeling Framework (EMF), and a CDO-based repository [7] ongoing experimental work is conducted with *semcomdt* (Semco Model Development Tools, IRIT's editor plugins) to produce an MDE Tool-chain, such as visualized in Fig. 4, supporting the approach. *semcomdt* provides a set of software tools, for instance for the design and for populating the repository and for retrieval and transform from the repository. For accessing the repository, *semcomdt* provides a set of facilities to help selecting appropriate patterns including *keyword* search, *lifecycle stage* search and property categories search. Currently the tool suite is provided as Eclipse plugins. For more details, the reader is referred to [15].

The following tools to perform the activities of management and populating the repository were developed:

- *Gaya:* a repository based on MDE technology was developed. This repository allows to store engineering and process knowledge associated with S&D patterns.
- *Arabion:* a tool for editing S&D patterns. These patterns must be stored in such a way that they can be reused later, enhanced and modified.
- *Tiqueo:* a tool for editing S&D properties and constraints. The focus is on the non-functional requirements that are associated with S&D patterns.

Moreover, a set of dedicated tools that are customized for a given software engineering environment (development platform) were developed: *Access tools* for Safe4Rail. The tool transforms the *Gaya* representation of S&D patterns into a representation that is consistent with the Safe4Rail set of tools (mostly Rhapsody UML-based) and the Safe4Rail process.

C. SEMCO-Methodology: From Pattern Repository to System Development

Along this description, we will give the main keys to understand why our process is based on a generic, incremental and a constructive approach. Once the repository[8] is available, it serves an underlying trust engineering process. In the process model visualized in Fig. 5 (the numbers in parentheses correspond to the numbers in Fig. 5), as activity diagram, the developer starts by system specification (A1) fulfilling the requirements. In a traditional approach (non pattern-based approach) the developer would continue with the architecture design, module design, implementation and test. In our vision, instead of following these phases and defining new modeling artifacts, that usually are time and effort consuming, as well as error prone, the system developer merely needs to select appropriate patterns from the repository and integrate them in the system under development.

For each phase, the system developer executes the search/select from the repository to *instantiate* patterns in its modeling environment (A4 and A9) and to *integrate* them in its models (A5 and A10) following an incremental process. The model specified in a certain activity is then used as an input work product in the following activities. Also, thanks to the system of patterns organization, the patterns identified in a certain stage will help during the selection activity of the following phases. Moreover, the system developer can develop their own solutions when the repository fails to deliver appropriate patterns at this stage. It is important to remark that the software designer does not necessarily need to use one of the artifacts stored in the repository previously included. He can define custom software architecture for some patterns (components), and avoid using the repository facilities (A6 and A11).

[7] http://www.eclipse.org/cdo/

[8] The repository system populated with S&D Patterns.

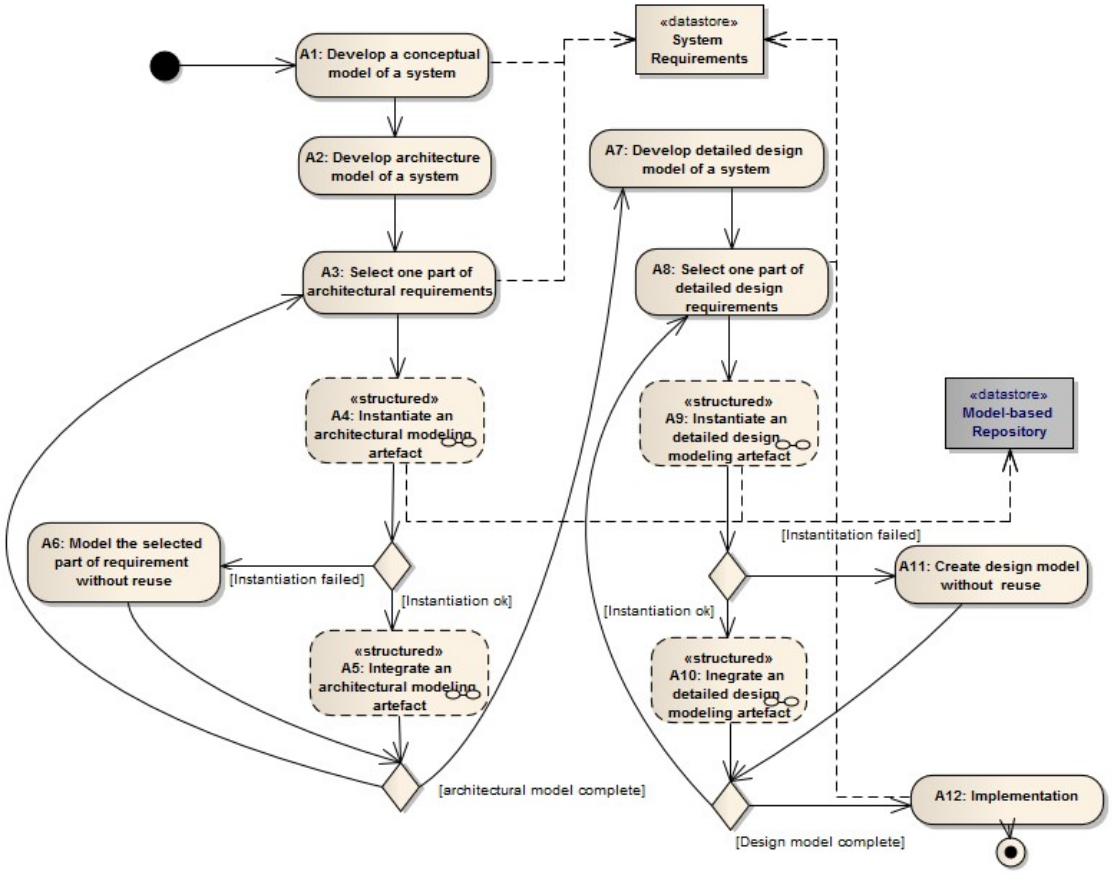


Fig. 5. The S&D pattern-based development process

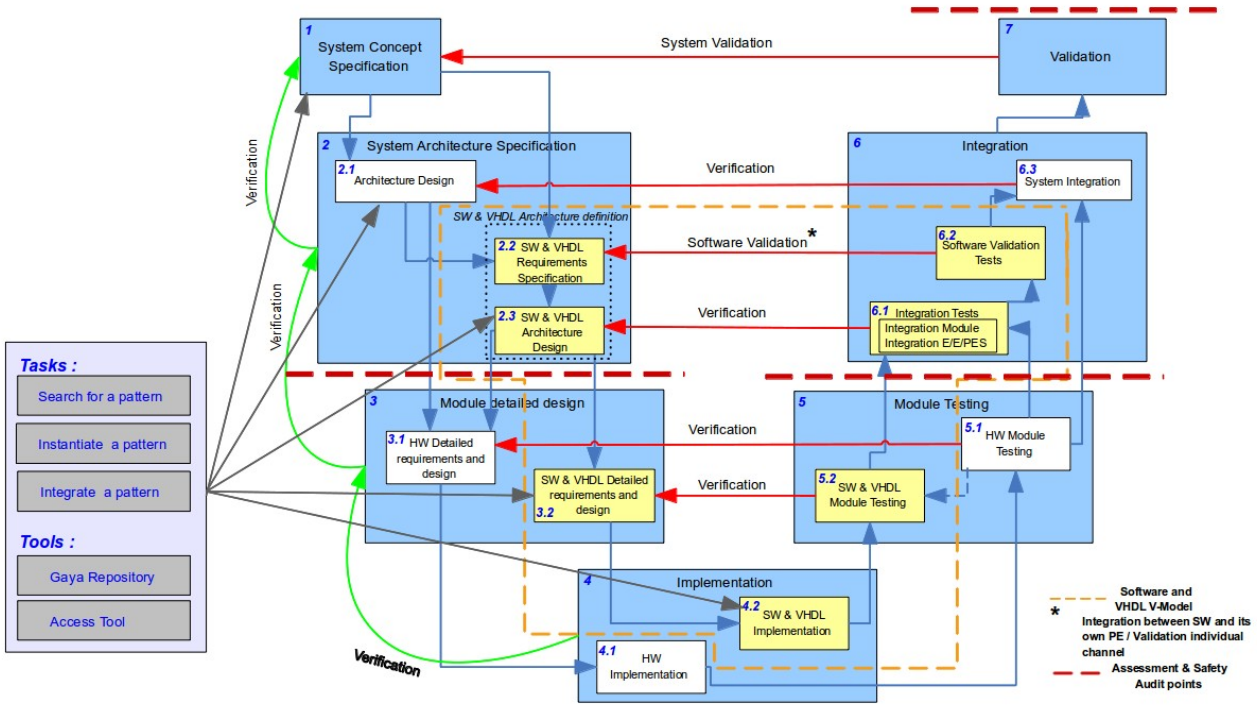


Fig. 6. An example of a railway engineering process

V. Application Domain Example

SEMCO is an effective approach, relying on an MDE tool-suite, to supporting secure and dependable system and software architecture engineering methodology and thus in our context supporting automated pattern-based building and access in industry[9]. We discuss the benefits, such as reduced modeling effort and improved readability, achieved when applying the methodology to an industrial case study. We have used the *SEMCO* modeling framework to model and to analyze secure and dependable pattern- based architectures for an application acting as one of the *TERESA* demonstrators for the railway domain called Safe4Rail. Safe4Rail is in charge of the emergency brake of a railway system. Its mission is to check whether the brake needs to be activated. Most important, the emergency brake must be activated when something goes wrong.

A. The TERESA Repository

An instance of the Gaya repository called *teresaRepository* was built in the context of the *TERESA* project to demonstrate reuse in a railway engineering environment and a metrology engineering environment. The railway and metrology domains analysis lead to identify a set of patterns to populate *teresaRepository*. We used the *Tiqueo* editor and *Arabion* editor to create the corresponding property libraries and the set of patterns, respectively. *Arabion* use the property libraries provided by *Tiqueo* to type the patterns property. Finally, we used the *Gaya* manager tool to set the relationships between the patterns.

The TERESA repository contains so far (on January 2014):

- *Compartments*. 21 compartments to store artifacts of the *TERESA* domains.
- *Users*. 10 users.
- *Property Libraries.* 69 property model libraries.
- *Pattern Libraries.* 59 patterns.

B. Application of the SEMCO Approach to a Railway System Case Study

Here, we examine the process flow for the example following the design process of Section IV-C and a very strict engineering process was followed, such as visualized in Fig.6, where specific activities were performed in order to achieve certification using the presented approach (the numbers in parentheses correspond to the numbers in Fig. 5). In this case, SIL4 level is targeted.

Once the requirements are properly captured and imported into the development environment, the process can be summarized with the following steps:

Activity A2: Develop architecture model of a system: (A3) The analysis of the requirements results in the needs of an architectural pattern for redundancy. Thus, activity (A4) is the instantiation of S&D patterns from the repository using the repository access tools. The running of the Retrieval tool using keywords *Redundancy* and *SIL4*, suggests to use a TMR pattern at architecture level. In addition, some diagnosis techniques imposed by the railway standard are suggested, thanks to the repository structure and the support of the system of patterns organization for the railway application domain. (A5) Finally, at architecture level, we will integrate the following patterns: (a) TMR (searched by the System Architect), (b) Diagnosis techniques (suggested by the tool) and (c) Sensor Diversity (searched by the System Architect).

Activity A7: Develop design model of a system: This activity involves the development of the design model of the system. (A8) The analysis of the requirements the architecture model and the identified architectural patterns will help during the instantiation activity of the design phase (A9). Based on the selected patterns, the repository may suggest related or complementary patterns. For instance, if the TMR has been integrated, the following patterns may be proposed for the design model iteration: (d) Data Agreement, (e) Voter, (f) Black Channel and (g) Clock Synchronization.

VI. Conclusion and Discussion

A Pattern Based System Engineering (PBSE) methodology based on a repository was specified. This engineering methodology fully takes into account the need for separation of roles by defining three distinct processes, the pattern modeling process, the repository specification process, and the pattern integration process. The implementation of a PBSE for S&D patterns is discussed in detail through a use case from railway domain. A set of languages were specified for the specification of S&D patterns, of S&D properties, of processes, and of the repository structure and content. By developing an effective model- and pattern- based engineering approach, *SEMCO* will contribute to the establishment of security and dependability as an engineering discipline in the area of embedded systems.

Our objective is to design frameworks to assist system and software developers in the domain of security and safety critical systems to capture, implement and document distributed system applications. We worked on the “theory of how”. We addressed four fundamental questions: (1) what is design solutions for the precise and valuable specifications of patterns and how can a pattern be specified hierarchically with all its facets? (2) what is a repository of patterns, and how can repository be built and used to instantiate its content? (3) what is PBSE and (4) how can pattern be integrated into a system under development? We also worked on a “practice of how” by providing the *SEMCO* tool-chain. We studied the first three questions, and we are now confronted with the fourth question.

We plan to extend this work in the following directions: We will investigate new design techniques to improve the pattern’s representation to ease their integration in existing software engineering processes targeting secure and dependable architectures. Furthermore, we will study the relation of our approach to the notion of pattern systems for security and safety critical systems, as a first step for MBSA (Model Based Safety Analysis). We wish to promote a

[9] The approach is evaluated in the context of the TERESA project (http://www.teresa-project.org/)

framework to define reference models and patterns (sub-systems) for modeling and analysis of systems with strong security and safety requirements. The results will be provided in a *SEMCO* repository.

We also aim to build an experimental study in part of a software development environment based on UML. This is to judge the relevance of the artifacts produced for the assessment process.

ACKNOWLEDGMENT

This work is initiated in the context of *SEMCO* framework. It is supported by the European FP7 *TERESA* project and by the French FUI 7 *SIRSEC* project. In addition, we would like to thank the *TERESA* consortium who gave us valuable feedback on this paper.

REFERENCES

[1] A. Ziani, B. Hamid, and S. Trujillo, Towards a Unified Metamodel for Resources-Constrained Embedded Systems, in 37th EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 485–492, IEEE, 2011.

[2] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, Security in embedded systems: Design challenges, ACM Trans. Embed. Comput. Syst., vol. 3, no. 3, pp. 461–491, 2004.

[3] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Transactions on Dependable and Secure Computing, vol. 1, pp. 11–33, 2004.

[4] D. Schmidt, Model-driven engineering, in IEEE computer, vol. 39, no. 2, pp. 41–47, 2006.

[5] I. Crnkovic, M. R. V. Chaudron, and S. Larsson, Component- based development process and component lifecycle, in Proceedings of the International Conference on Software Engineering Advances (ICSEA 2006), p. 44, IEEE Computer Society, 2006.

[6] F. Buschmann, K. Henney, and D. Schmidt, Pattern-Oriented Software Architecture, Volume 4: A Pattern Language for Distributed Computing. Wiley, 2007.

[7] M. Schumacher, Security Engineering with Patterns - Origins, Theoretical Models, and New Applications, vol. 2754 of Lecture Notes in Computer Science. Springer, 2003.

[8] D. C. Schmidt and F. Buschmann, Patterns, frameworks, and middleware: Their synergistic relationships, in ICSE, pp. 694– 704, IEEE Computer Society, 2003.

[9] B. P. Douglass, Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems. Boston, MA, USA: Addison- Wesley Longman Publishing Co., Inc., 2002.

[10] B. Hamid, S.Gurgens, C. Jouvray, and N. Desnos, Enforcing S&D Pattern Design in RCES with Modeling and Formal Approaches, in ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS) (J. Whittle, ed.), vol. 6981, pp. 319–333, Springer, octobre 2011.

[11] B. Hamid, J. Geisel, A. Ziani, J. Bruel, and J. Perez, Model- Driven Engineering for Trusted Embedded Systems Based on Security and Dependability Patterns, in SDL Forum, pp. 72– 90, 2013.

[12] J. Gray, J.-P. Tolvanen, S. Kelly, A. Gokhale, S. Neema, and J. Sprinkle, Domain-Specific Modeling. Chapman & Hall/CRC, 2007.

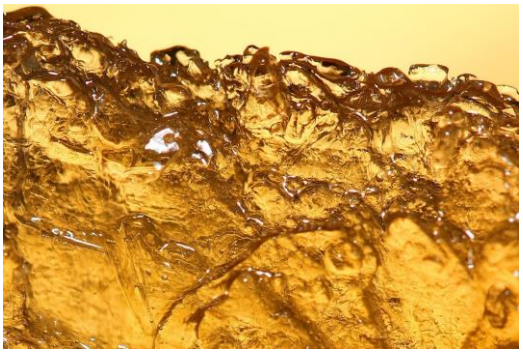
[13] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, EMF: Eclipse Modeling Framework 2.0. Addison-Wesley Professional, 2nd ed., 2009.

[14] B. Hamid and C. Percebois, A modeling and formal approach for the precise specification of security patterns, in Engineering Secure Software and Systems - 6th International Symposium, ESSoS 2014, vol. 8364 of Lecture Notes in Computer Science, pp. 95–112, Springer, 2014.

[15] B. Hamid, A. Ziani, and J. Geisel, Towards Tool Support for Pattern-Based Secure and Dependable Systems Development, in ACadeMics Tooling with Eclipse (ACME), Montpellier, France, pp. 1–6, ACM DL, 2013.

Dr Brahim Hamid: *B. Hamid received his PhD degree from the University of Bordeaux (France), his PhD thesis was on the study of dependability in distributed computing systems. Then he worked at the CEA-Saclay List (French laboratory specialized on real-time embedded system modelling) where he worked on distributed component-based applications for dependable embedded and real-time. Currently, he is an associate professor at the University of Toulouse (France) and member of the IRIT-MACAO team. His main research topic is software languages engineering, at both the foundations and application level, for the development of secure and dependable distributed systems using model-driven engineering and design patterns. He participated in several research projects. In particular, he has led successfully the IRIT effort on the TERESA European project, and several national projects.*

Advanced Dependence Analysis for Software Testing, Debugging, and Evolution



Raul Santelices, Haipeng Cai, Siyuan Jiang, and Yiji Zhang
Department of Computer Science & Engineering
University of Notre Dame
Notre Dame, Indiana, United States
email: {rsanteli|hcai|sjiang1|yzhang20}@nd.edu

***Abstract*— Software is more than just a collection of individual entities (e.g., components, statements). The behavior of software is the result of the interactions among those entities. To analyze and improve the functional behavior of software for software reliability and engineering tasks such as testing, debugging, and evolution, we need dependence analysis to identify which entities affect which other entities, and in which ways.**

In this article, we briefly review classical concepts and applications of dependence analysis and we present new directions and results on dependence quantification and abstraction, which extend the utility of classical dependence analysis for modern software.

Keywords—software testing; debugging; change-impact analysis; program dependence; dependence quantification

I. INTRODUCTION

Software defects have enormous economical and human costs for our modern society, which increasingly depends on software. Defects can affect critical qualities of software and information systems such as correctness, reliability, security, and safety. Developers, however, struggle daily with market pressures, resource constraints, and increasing complexity, all of which limits their ability to detect and fix serious software defects. Moreover, modern software evolves (i.e., it changes constantly), which adds a new dimension of complexity to software assurance tasks. For example, changing the type of a collection of items from a set to a list might lead to undesired duplicate items. For another example, adding minimum-age constraints to a banking system might prevent children from monitoring their accounts even if only parents can withdraw funds.

To ensure the reliability of a version or series of versions of the software, developers create test suites to automate its testing. When a test case fails (e.g., the system crashes or allows unauthorized access), developers perform debugging to identify and fix the underlying defect. This test-and- debug process continues for each new version that is produced as the software evolves. Each such version is, in fact, the result of a change process. Because software entities interact and affect each other, before making any change, developers must understand the impacts and risks of their proposed changes and take appropriate actions (e.g., adapt impacted entities, update the test suite).

These testing, debugging, and evolution tasks in real-world software are far from trivial. To complete these tasks effectively and timely, automated support is crucial. Program-analysis techniques and, in particular, dependence analysis [7], have arisen to automatically identify relation- ships among software entities (e.g., components, methods, statements) and to reason about those relationships. Dependence analysis underlies most white-box testing techniques [9] as well as (semi-)automated debugging [14] and change-impact analysis [2]. Thus, dependence analysis is crucial for a large class of software-engineering and assurance activities.

For example, it is possible to identify the defect that causes a sensor to output an erroneous value v because that value is computed by (i.e., is data dependent on) a statement s which, while correct, executes when decided by (i.e., is control dependent on) a condition c that compares two temperature values which are in different units, making c the source of the defect. For this example, testing would exercise these dependencies [9] by making c lead to the execution of s, which affects the output v whose value is not the one expected. Then, debugging traverses those dependencies back- wards [14] from the output v until identifying c as the cause (unit mismatch). When preparing to fix this defect and make other changes, impact analysis [2] would identify which code, such as other computations also dependent on c, might be affected. Impacted code would have to be inspected and also changed if necessary. Finally, the interactions among changes should be tested [10].

Data and control dependencies, however, do not capture all aspects of the interactions among software entities. Remarkably, classical analyses do not indicate under which program states a bug or change propagates its effects—they only indicate which dependencies they propagate through [11]. Although other types of dependencies, such as structural (e.g., call graphs) and socio-technical (e.g., developers and components) [13], are used to model relationships in software projects at higher levels of abstraction to capture distinct information, the essential information they convey is still whether entities depend on other entities, but not how. Moreover, at abstract levels, commonly-used structures such as call graphs only represent control flow rather than behavioral dependencies: a method that calls another method might or might not affect its behavior.

To overcome these limitations of classical dependence analysis, two directions have emerged recently which complement and enrich the information given by those analyses: information carried by dependencies describing or

quantifying the states in which they propagate effects [9, 12, 15] and higher-level behavioral definitions of dependencies that supersede simpler control-flow relationships [4, 5]. For software evolution, the first direction identifies the exact conditions on the program state under which a change propagates its effects through sequences of dependencies [9]. Because executing sequences of dependencies is not enough to guarantee that the effects of changes will be observed, these conditions are needed. In this same direction, as a more scalable alternative to computing and monitoring complex state conditions, a newer trend is to quantify dependencies with the probabilities that the effects of bugs or changes propagate through them [12, 15].

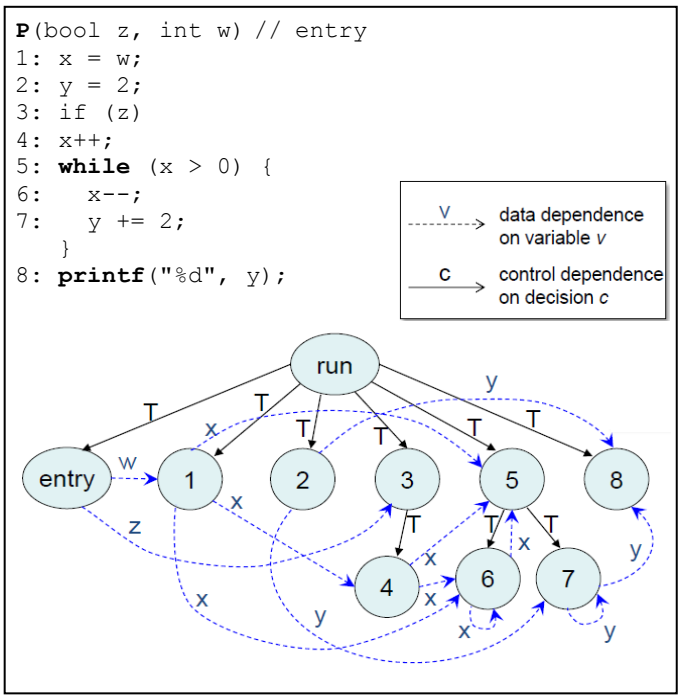


Fig. 1. Example program P and its program dependence graph (PDG).

The second recent direction addresses the imprecision of method-level structural dependence analyses [1] which are based on execution orders and scales better than statement-level analyses. Studies for the dynamic (i.e., execution-based) form of this analysis have identified a large degree of imprecision in these traditional techniques [5]. Consequently, a new technique was developed which uses behavioral definitions of method-level dependencies based on data- and control-flow analysis. Based on initial results and further ongoing research, the new technique appears to triple the precision of classical results without significant increases in cost [4], to make method-level dependence analysis a truly cost-effective alternative to statement-level analysis.

II. Classical Dependence Analysis

This section reviews core concepts of classical dependence analysis and illustrates these concepts using the example program of Figure 1. In this example, program P takes a boolean `z` and an integer `w` as inputs, initializes local variables `x` and `y`, conditionally increments x based on `z`, updates `y` in a loop controlled by `x`, and finally prints the value of `y`. The graph on the right shows the dependence structure of this program, as explained next.

A. Syntactic Dependencies

Syntactic program dependencies [7] are derived directly from the program's syntax. These dependencies are classified as control or data dependencies. A statement s1 is control dependent on a statement s2 if a branching decision at s2 determines whether s1 necessarily executes. In Figure 1, for example, statement 4 is control dependent on statement 3 because the decision taken at 3 determines whether statement 4 executes or not.

A statement *s* is data dependent on a statement *t* if a variable `v` defined (written) at *t* might be used (read) at *s* and there is a definition-clear path from *t* to *s* in the program for `v` (i.e., a path that does not re-define `v`). For example, in Figure 1, statement 8 is data dependent on statement 7 because 7 defines `y`, 8 uses `y`, and there is a path (7,5,8) that does not re-define `y` after 7 in the last iteration of the loop. Statement 8 is also data dependent on 2 because an execution might not enter the loop (the loop re-defines `y` at 7).

The parameters `z` and `w` at the entry line of P are inputs and thus are not data dependent on any statement. However, these parameters are treated as definitions of `z` and `w`. Thus, the statements 1 and 3, which use those variables, are data dependent on the entry of P.

On the right, Figure 1 shows the program dependence graph (PDG) of P. PDGs and their inter-procedural (all-functions) extensions depict the dependence structure of entire programs. In the figure, the nodes are the statements of the program (1 to 8) and two special nodes: *run* which represents the decision of executing the program and *entry* for the entry of P where the parameters are defined. Solid edges represent control dependencies labeled with the corresponding control decision (e.g., T or F). The solid edges T from run represent the consequences of deciding to run P. Dashed edges represent data dependencies and are labeled with the corresponding variables.

B. Semantic Dependencies

Semantic dependencies represent the actual behaviors that the program can exhibit. Syntactic dependencies, in contrast, can only (over-)approximate those behaviors because a sequence of data and control dependencies between two statements is a necessary but not sufficient condition for those statements to be semantically dependent. Informally, a statement *s* is semantically dependent on a statement *t* if there is any change that can be made to *t* that affects the behavior of *s*.

For example, in Figure 1, the statements in the loop (5, 6, and 7) are semantically dependent on statements 1, 2, 3, and 4 because the latter could be changed so that the execution of the loop changes (e.g., by iterating more or fewer times) or the state of the variables computed in the loop (`x` and `y`) changes. In this case, the semantic dependencies of the loop statements coincide with their direct and transitive syntactic dependencies—as represented by the PDG.

However, for this same example, if w is guaranteed to be always negative (e.g., P is a function in a larger program whose only caller passes -1 for `w`), the loop head (statement 5) will always behave in the same way by evaluating to false and the loop body (statements 6 and 7) will never execute. In that case,

despite being syntactically dependent on statements 1–4, the loop statements 5–7 are not semantically dependent on 1–4.

This case illustrates the caveats of classical dependence analysis, which in is most precise form is based on syntactic (control and data) dependencies. Unfortunately, computing semantic dependencies is an undecidable problem [7]. In between these two types of dependencies, however, there is room for incorporating state information not captured by classical data- and control-dependence analysis, as discussed in the next sections.

III. Dependence Quantification

Recently, to address the limitations of classical dependence analysis, we introduced the concept of *dependence quantification* [12]. This approach enriches dependencies by assigning to them *quantities* that represent properties such as likelihood or strength. Such quantities denote the relative relevance of dependencies in programs, which help users and tools focus their attention first on the dependencies that—likely—matter the most. Quantification has been found to effectively direct users first to the areas most affected by a change [3] or most likely to contain bugs [15].

A. Example

Consider the code fragment in Figure 2 for finding tangents between circles. The forward static slice (transitive syntactic dependencies) from `c` at line 2 contains lines 2–11, which suggests that they might be affected by `c`. However, `c` at line 2 strongly affects `c*c` at line 3 but less strongly affects the branching decision in that line—variations in `c` may or may not flip the branch taken. Therefore, because `c` may or may not affect this decision, the remaining lines are "less affected" than lines 2 and 3. Lines 6–11, however, also use the value of `c`, which makes them "more affected" than lines 4 and 5 (but still less affected than lines 2 and 3).

Quantification identifies these differences among lines in the slice. This kinds of approach can give, for example, a score of 1.0 to lines 2 and 3, 0.5 to lines 4 and 5, and an intermediate value 0.75 to the rest. The actual scores for these lines will depend on the specific technique used to quantify the slice. We describe two such techniques next.

B. Static Analysis

Static quantification of program slices (i.e., transitive syntactic dependencies) can be achieved by analyzing the control- and data-flow structure of programs. The advantages of this approach to quantification are that no execution data is required and the results represent all behaviors of the program. The latter advantage is quite attractive for tasks like testing because, no matter how many test cases have been created, this approach points users to behaviors not tested yet.

```
 1: for (sign1 = +1; sign1 >= -1; sign1 -= 2) {
 2:   c = (r1 - sign1 * r2) / d;
 3:   if (c*c <= 1.0) {
 4:     h = sqrt(d*d-pow(r1-sign1*r2),2))/d;
 5:     for (sign2=+1; sign2>=-1; sign2-=2) {
 6:       nx = vx * c - sign2 * h * vy;
 7:       ny = vy * c + sign2 * h * vx;
 8:       print x1 + r1 * nx;
 9:       print y1 + r1 * ny;
10:       print x2 + sign1 * r2 * nx;
11:       print y2 + sign1 * r2 * ny; }}}
```

Fig. 2. Excerpt from a program that computes the tangents among circles.

For this static approach to dependence quantification, we used two key insights:

1. Some data dependencies are less likely to occur than others because the conditions to reach the target from the source of the dependence vary.
2. Data dependencies are more likely to propagate information than control dependencies, yet control dependencies should not be ignored either as in classical analyses.

Using the first insight, a reachability and alias analysis of the control flow of the program have been created. This analysis estimates the probability that the target of a dependence is reached from its source and that both points access the same memory location. Using the second insight, the approach performs another reachability analysis, this time on the dependence graph, which gives a lower but non-zero score to control dependencies.

We estimate the probability that a statement *a* affects a statement *b* by computing two components: (1) the probability that a sequence of dependencies from *a* to *b* occurs when the program executes and (2) the probability that information flows through that sequence. We present more details of this static approach and initial positive results in [11].

C. Dynamic Analysis for Quantification

Given a representative test suite, we can quantify slices via differential execution analysis [10] and sensitivity analysis [8].

1) Differential Execution Analysis: Differential execution analysis (DEA) is designed specifically for forward slicing from changes to identify the runtime semantic dependencies [7] of statements on changes. Semantic dependencies tell which statements are truly affected by which other statements or changes. Although finding semantic dependencies is an undecidable problem, DEA detects such dependencies on changes when they occur at runtime to under-approximate the set of semantic dependencies in the program. Therefore, DEA does not guarantee 100% recall of semantic dependencies but it achieves 100% precision. This is much better than what dynamic slicing normally achieves [6, 10].

DEA works by executing a program before and after the change, collecting the augmented execution history [10] of each execution, and then comparing both histories. The execution history of a program for an input is the sequence of statements executed for that input. The augmented execution history is the execution history annotated with the values read and written by each statement occurrence. The differences between two such histories reveal which statements had their

occurrences or values altered by a change—the conditions for semantic dependence. A formal definition of DEA is given in [10].

2) Sensitivity Analysis: DEA can be used to quantify static forward slices when the change is known—for post-change impact analysis. To do this, a DEA-based quantification technique can execute the program repeat- edly with and without the change for many inputs and find, for each statement, the frequency with which it is impacted by the change. If the inputs are sufficiently representative of the program's behavior, we can use these frequencies as the quantities for the statements in a slice.

More generally, however, the specifics of a change might not be known when a user asks for the impacts of modifying a statement or when the slicing task does not involve a change (e.g., debugging, information-flow analysis). For such situations, we created SENSA, a new sensitivity- analysis technique and tool for slice quantification and other applications [3]. Sensitivity analysis is used in many fields to determine how modifications to some aspect of a system (e.g., an input) affect other aspects of that system (e.g., the outputs) [8].

We designed SENSA as a generic modifier of program states at given locations, such as changes or failing points. SENSA inputs a program P, a test suite T, and a statement *c*. For each test case *t* in T, SENSA executes *t* repeatedly, replaces each time the value(s) computed by *c* with a different value, and uses DEA to find which statements were affected by these modifications. With this information for all test cases in T, SENSA computes the sensitivity of each statement *s* in P to the behavior of *c* by measuring the frequency with which *s* is affected by *c*. These frequencies are the degree of dependence on statement c of all statements s in P, given T.

For a forward static slice from statement *c* in program P, SENSA uses T to quantify the dependence on *c* of the statements in that slice. For a backward static slice from *s*, SENSA can be used in a similar fashion to quantify the dependence of *s* on selected statements *c* from that slice.

SENSA is highly configurable. In addition to parameters such as the number of times to re-run each test case with a different modification (the default is 20), SENSA lets users choose among built-in modification strategies for picking new values for c at runtime. Furthermore, users can add their own strategies. SENSA ensures that each new value picked for *c* is unique, to maximize diversity while minimizing bias. Whenever a strategy runs out of possible values for a test case, SENSA stops and moves on to the next test case. SENSA offers two modification strategies from which the user can choose:

1. Random: A random value is picked within a specified range. By default, the range covers all elements of the value's type except for char, for which only readable characters are picked. For some reference types such as String, objects with random states are picked. For all other reference types, the strategy currently picks null.
2. Incremental: A value is picked that diverges from the original value by increments of *i* (the default is 1.0). For example, for a value *v*, the strategy first picks $v + i$ and then picks $v - i$, $v + 2i$, $v - 2i$, etc. For common non-numeric types, the same idea is used. For example, for string *foo*, the strategy picks *fooo*, *fo*, *foof*, *oo*, etc.

3) Empirical Results: To assess the effectiveness of SENSA, we implemented it to analyze Java-bytecode programs and applied it to the task of predicting change impacts at various program locations. The scenario is early change-impact analysis, in which users query for the potential impacts of changing a location without necessarily knowing the details of the change yet.

For this study, we chose four Java subjects for which many test cases and changes (bug fixes) are provided for research studies. Table 1 in columns 1–4 lists these subjects along with their sizes, test suites and changes used. We compared the results of SENSA for this task with those of the two main techniques from classical dependence analysis: static slicing (code-based) and dynamic (execution-based) slicing.

We first applied SENSA and static forward slicing to the locations of the changes to reproduce the scenario in which users query for the consequences that changing those locations would have. Then, for each result of SENSA, we ranked the statements from greatest to lowest score. For comparison, we also ranked the static and dynamic slices using Weiser's approach [14] by visiting statements breadth-first from the change location and sorting them by increasing visit depth. For tied statements in a ranking, we used as their rank the average of their positions in that ranking.

To compare the predictive power of the rankings given by SENSA and slicing, we applied the changes, one at a time, to the corresponding location in its subject. Then, for each change, we used DEA on the unchanged and changed subject to find the statements actually impacted when running all test cases for that subject. Using these actual impacts, we calculated how closely each ranking predicted those impacts. For each ranking and each impacted statement found by DEA, we determined the percentage of the static slice that would have to be traversed, in the ranking's order, to reach that statement. We call this percentage the cost of finding an actually-impacted statement using that ranking. Then, we computed the average cost of finding all actual impacts for each ranking—the lower this cost is, the better the technique that produced that ranking is at predicting impacts. Also, to assess how close to the best possible result each ranking was, we created the ideal ranking by placing all actually-impacted statements at the top of that ideal ranking.

The last five columns of Table 1 present the average cost, for each subject and the seven changes in that subject, of the following rankings: the ideal ranking, the static and dynamic slicing rankings (using Weiser's traversal), and the rankings for SENSA and its strategies Rand (Random) and Inc (Incremental).

The ideal (best possible) costs in Table 1 reveal that, on average, most of the statements in Schedule1 were actually impacted, in contrast with the other three subjects, for which low numbers of statements were impacted. Overall, for static and dynamic slicing, the prediction costs ranged between 23–50% with average 36% and 41%, respectively, whereas the two strategies of SENSA were much closer to the ideal predictions at 13–48% and averages of 28% and 29%, respectively.

In all, SENSA seems much better overall than classical slicing (Weiser's traversal) at predicting actual impacts. A non-

parametric (i.e., no data distribution assumptions) Wilcoxon signed-rank test gives p-values below 0.05, as shown at the bottom of Table 1, which indicate that the superiority of SENSA over slicing for predicting impacts is statistically significant.

$M0_e\ M1_e\ M2_e\ M5_e\ M2_i\ M1_i\ M3_e\ M1_i\ M0_i\ M4_e\ M4_i\ M0_i\ x$

(Subscript *e* for entry event and *i* for returned-into event; entry method M0; exit event *x*)

Fig. 3. Example execution trace of a program with six methods M0–M5.

IV. Method-level Dependencies

A. Problem

For scalability of dependence analysis, it is often desirable to switch the granularity of an analysis from the statement level to the method level. Although important information can be missed by doing so, the analysis can provide a higher-level picture of larger software systems.

Unfortunately, at the method and coarser levels, classical analyses use simple structural dependencies such as method calls. For example, a method m called by a method ml, directly or transitively, is considered as dependent on ml. This can be quite imprecise. In fact, we found in re- cent studies that classical method-level dynamic impact analyses such as PATHIMPACT [1] can be too imprecise [5] with large numbers of false positives. The reason is that, at its core, PATHIMPACT simply marks as impacted by a method m all methods called or returned into after m executes.

Figure 3 shows an example execution trace where the *e* and *i* subscripts for a method represent the entry and return-into events for that method. When queried for the impact set of M2, in addition to M2 itself, PATHIMPACT first finds M5, M3, and M4 as impacted because they were entered after M2 was entered and then finds M0 and M1 because they returned after M2 was entered (i.e., parts of them executed after parts or all of M2). Thus, the resulting dynamic impact set of M2 is {M0, M1, M2, M3, M4, M5} for this trace. For multiple traces, the results are unioned.

B. Solution

Reaching a method *m'* after a method m at runtime is a necessary condition for *m* to impact *m'*, but not all such methods *m'* necessarily depend on *m*. To fix this problem, we recently proposed a technique called DIVER [4] which builds first a method-level dependence graph of the program— based on statement-level dependencies—which is then used to prune execution traces to find which methods really depend on *m*. For a reasonable cost, these method-level dependencies can be much more precise for impact analysis than simple call information.

DIVER works in three phases: static analysis, runtime, and post-processing. The static-analysis phase computes method-level data and control dependencies for the entire program. This is done only one time, regardless of the executions to be considered for the runtime phase and the queries for impact sets performed at post-processing. At the method level, a method *m* is dependent on a method *m'* if there is a statement *s* in *m* that is dependent on a statement *s'* in *m*' via a sequence of one or more statement-level data and control dependencies. The runtime phase is identical to PATHIMPACT by simply collecting the method traces. The post-processing phase, however, uses the static dependencies to determine whether a method *m* that executed after *m'* did so along a dependence path from *m'* to *m*. If not, unlike PATHIMPACT, DIVER does not report m as impacted.

To illustrate, consider again the trace in Figure 3. It is easy to imagine a program with these methods in which not all methods depend on each other—see [4] for an example. For such a program and query M2, for example, DIVER traverses the trace to find which dependencies were exercised in it after M2 and, via those dependencies, which methods depended directly or transitively on any occurrence of M2. When DIVER finds M2, the impact set starts as {M2}. Then, there might be only one outgoing dependence from M2 in the graph—to method M5 [4]—in which case the impact set is {M2, M5}, in contrast with PATHIMPACT which reports all six methods.

C. Empircal Results

Table 2 presents comparative precision results for DIVER and PATHIMPACT for the same four subjects used in Section III.C.3, with two statistics per subject and overall for all queries (last row) for the corresponding data points: the mean and the standard deviation (*stdev*) of the impact set sizes and ratios.

The results in the table show that, on average, the DIVER impact sets were much smaller than for PATHIMPACT, especially for the two largest subjects. Large numbers of false positives for PATHIMPACT were identified as such and pruned by DIVER. For example, PATHIMPACT identified 160 methods on average in its impact sets for Ant, whereas DIVER reported only 18 for a mean ratio of 25.7%. (These values are means of ratios—not ratios of means.) Also, the large standard deviations indicate that the impact-set sizes fluctuate greatly across queries for every subject except Schedule1. The results suggest that DIVER is even stronger with respect to PATHIMPACT for larger subjects, which are more representative of modern software. For the smaller subjects Schedule1 and NanoXML, DIVER provides smaller gains possibly due to the proximity and interdependence of the few methods they contain.

We applied the Wilcoxon signed-rank one-tailed test for all queries in each subject and also for the set of all queries in all subjects. This is a non-parametric test that makes no assumptions on the distribution of the data. The last column in Table 2 shows the resulting p-values. For $\alpha = .05$, the null hypothesis is that DIVER is not more precise than PATHIMPACT. The p-values show strongly that the null hypothesis is rejected and, thus, the superiority of DIVER is statistically significant for these subjects and test suites.

In all, DIVER can safely prune 70% of the impact sets computed by PATHIMPACT, which amounts to a precision increase by a factor of 3.33 (i.e., by 200%). Thus, method-level data and control dependencies should replace method calls as the common practice for high-level analysis. For more technical and empirical details on DIVER, we refer the reader to [4].

V. Conclusion

Dependence analysis for software testing, debugging, evolution, and many other tasks has a long history [1, 7, 14]. However, there are still under-explored dimensions of

dependence analysis which can significantly increase the cost-effectiveness of its applications to these tasks [3, 4, 12]. This ongoing work shows that fundamental advances are still possible and necessary.

In this article, we revisited the classical concepts of data and control dependence analysis at the statement level and structural (call) dependencies at the method and higher levels. Then, we showed some recent advances and results on two particular dimensions—quantification and abstraction— which considerably improve upon the results of classical analyses. Through this exposition, we expect the reader to have gained new insights into the fundamentals and some recent advances of this field, which underlies the description of software behavior for virtually any engineering task.

ACKNOWLEDGMENT

This work was supported by ONR Award N000141410037 to the University of Notre Dame.

REFERENCES

[1] T. Apiwattanapong, A. Orso, and M. J. Harrold. Efficient and Precise Dynamic Impact Analysis Using Execute-after Sequences. In Proceedings of IEEE/ACM International Conference on Software Engineering, pages 432–441, May 2005.

[2] S. A. Bohner and R. S. Arnold. An Introduction to Software Change Impact Analysis. In Software Change Impact Analysis, Bohner & Arnold, Eds. IEEE Computer Society Press, pages 1–26, June 1996.

[3] H. Cai, S. Jiang, R. Santelices, Y. jie Zhang, and Y. Zhang. SENSA: Sensitivity Analysis for Quantitative Change-impact Prediction. In Proc. of IEEE Int'l Working Conf. on Source Code Analysis and Manipulation, pp. 165-174, Sept. 2014.

[4] H. Cai and R. Santelices. DIVER: Precise Dynamic Impact Analysis Using Dependence-based Trace Pruning. In Proc. of IEEE/ACM Int'l Conf. on Automated Software Engineering, New Ideas track, pp. 343-348. Sept. 2014..

[5] H. Cai, R. Santelices, and T. Xu. Estimating the Accuracy of Dynamic Change-Impact Analysis using Sensitivity Analysis. In Proc. of Int'l Conf. on Software Security and Reliability (IEEE Reliability Society), pages 48–57, June 2014.

[6] W. Masri and A. Podgurski. Measuring the Strength of Information Flows in Programs. ACM Trans- actions on Software Engineering and Methodology, 19(2):1–33, 2009.

[7] A. Podgurski and L. A. Clarke. A Formal Model of Program Dependences and Its Implications for Software Testing, Debugging, and Maintenance. IEEE Transactions on Software Engineering, 16(9):965–979, 1990.

[8] A. Saltelli, K. Chan, and E. M. Scott. Sensitivity Analysis. John Wiley & Sons, Mar. 2009.

[9] R. Santelices and M. J. Harrold. Demand-driven propagation-based strategies for testing changes. Software Testing, Verification and Reliability, 23(6):499–528, 2013.

[10] R. Santelices, M. J. Harrold, and A. Orso. Precisely detecting runtime change interactions for evolving software. In Proceedings of IEEE International Conference on Software Testing, Verification and Validation, pages 429–438, Apr. 2010.

[11] R. Santelices, Y. Zhang, H. Cai, and S. Jiang. Change-Effects Analysis for Evolving Software. Advances in Computers, 93:227–285, Mar. 2014.

[12] R. Santelices, Y. Zhang, S. Jiang, H. Cai, , and Y. jie Zhang. Quantitative Program Slicing: Sepa- rating Statements by Relevance. In Proceedings of IEEE/ACM International Conference on Software Engineering – New Ideas and Emerging Results track, pages 1269–1272, May 2013.

[13] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb. Tesseract: Interactive visual exploration of socio-technical relationships in software development. In Proc. of IEEE/ACM Int'l Conf. on Softw. Eng., pages 23–33, May 2009.

[14] M. Weiser. Program slicing. IEEE Transactions on Software Engineering, 10(4):352–357, 1984.

[15] Y. Zhang and R. Santelices. Prioritized Static Slicing for Effective Fault Localization in the Absence of Runtime Information. Technical Report TR 2013-06, CSE, U. of Notre Dame, Nov. 2013. 12pp.

TABLE I. AVERAGE INSPECTION EFFORTS USING PREDICTIONS OF SLICING AND SENSA

Subject name	Lines of code	Test cases	Changes studied	Ideal (best) case	Average effort			
					Static slicing	Dynamic slicing	SENSA Rand	SENSA Inc
Schedule1	301	2650	7	47.90	50.14	48.34	48.01	48.01
NanoXML	3521	214	7	8.84	22.71	27.09	20.27	22.37
XML-security	22361	92	7	5.00	31.94	45.37	13.15	21.49
Ant	44862	205	7	3.21	39.16	41.55	29.84	23.76
			average:	**16.24**	**35.99**	**40.59**	**27.82**	**28.91**
			standard deviation:	19.36	13.22	13.08	19.20	20.59
				p-value w.r.t. static slicing:			**0.0098**	**0.0237**

TABLE II. RELATIVE PRECISION AS RATIOS OF IMPACT SET SIZES OF DIVER TO PATHIMPACT (PI).

Subject name	PI IS Size		DIVER IS Size		IS Size Ratio		Wilcoxon
	mean	stdev	mean	stdev	mean	stdev	p-value
Schedule1	18.0	1.6	12.8	4.7	71.3%	24.5%	6.65E-05
NanoXML	82.6	48.1	37.1	28.9	51.7%	33.1%	2.40E-30
XML-security	199.8	168.4	45.1	68.1	28.8%	30.3%	4.79E-102
Ant	159.5	173.4	17.9	34.3	25.7%	33.6%	2.94E-100
average:	166.2	164.9	32.2	53.1	**30.8**%	**33.3**%	**9.29E-07**

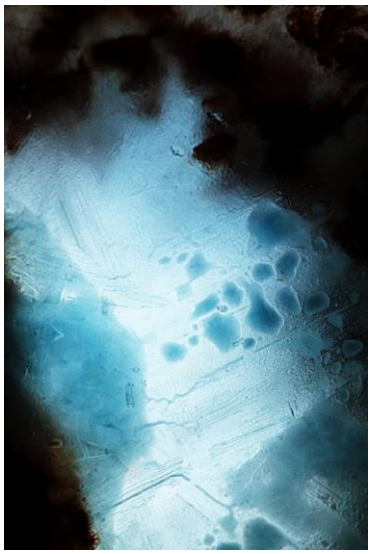
Raul Santelices *is an Assistant Professor in the Computer Science and Engineering Department at the University of Notre Dame in the US. In the past, he was as a software architect in educational and video games ventures. His academic interests are software engineering in general and program analysis in particular for software testing, debugging, and evolution. He received the Best Paper and ACM SIGSOFT Distinguished Paper awards at the IEEE/ACM International Conference on Automated Software Engineering in 2008. His work is partially funded by the Office of Naval Research in the US.*

Siyuan Jiang *is a Ph.D. student in Computer Science at the University of Notre Dame in the US. Her research interest is in software engineering, specifically on dynamic slicing, change-impact quantification, and dynamic symbolic execution. She obtained her M.S. and B.E. degrees in Software Engineering at East China Normal University and Tongji University in China, respectively.*

Haipeng Cai *is a Ph.D. student in Computer Science at the University of Notre Dame in the US. He worked on computer graphics and visualizations during his previous graduate studies and was a software developer in internet search services and embedded systems. His current research interests are in software engineering with emphasis on program analysis. He obtained his M.S. degree in Computer Science from The University of Southern Mississippi.*

Yiji Zhang *is a Ph.D. student in Computer Science at the University of Notre Dame in the US. Her research interest is in software engineering, specifically on static probabilistic models of program behavior for program slicing and fault localization. She obtained a B.E. in Software Engineering at Zhejiang University in China.*

Application of Multi-Model Combinations to Evaluate the Program Size Variation of Open Source Software



Shih-Min Huang
Fab. 12 Equipment Automation
Taiwan Semiconductor Manufacturing Company Limited
Hsinchu, Taiwan
n0140105@hotmail.com

Chin-Yu Huang
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
cyhuang@cs.nthu.edu.tw

***Abstract*—As one of the most important internal attributes of software systems, the estimation of software size is crucial to project management and schedule creation/tracking. Program size can be typically described by the length, functionality, or complexity of the software, but in practice most people customarily use the lines of code (LOC) as a major measure of program size. In actuality, LOC is still widely used, and can be easily measured upon project completion. In this study, we used the methods of multi-model combinations to predict and analyze the size distribution and size-change rate of Open-Source Software (OSS). Experiments based on real OSS data will be performed, and the evaluation results have shown that the proposed method had a better prediction capability of size distribution and size-change rate of OSS. Our goal is not to add one more model to the already-existing large number of models, but to emphasize a new approach for the evaluation of program size variation and reveal different issues of software sizing.**

Keywords*—Size estimation, Code length, Combinational Model, Laplace distribution, Normal distribution.

I. INTRODUCTION

During the software development life cycle (SDLC), size is one of the metrics related to the properties and specifications of the software. For the purpose of software measurement, acquiring objective and quantitative results can be important for applications regarding schedule and budget planning, cost estimation, quality assurance testing, etc [1]. Some research has shown that the statistical distribution of source-code file size is a lognormal distribution [2]. But Concas et al. [3] reported that the statistical distribution of source code file sizes follows a double Pareto distribution. For years to come, we may see numerous research that concentrates on file size rather than on the software size-change rate. As a matter of fact, if the focus is directed to the software size-change rate, this metric may give particular insight into software metrics.

Presently, the commercial off-the-shelf (COTS) systems can be roughly divided into two kinds: Closed-Source Software (CSS, e.g., Microsoft Windows, Adobe Photoshop, etc.) and Open-Source Software (OSS, such as Ubuntu Linux, Samba, Apache HTTP Server, etc.) The source code of CSS is typically not available, but OSS generally allows users to access the source code. Many successful OSS projects like Linux, Apache and others are growing steadily. But the main challenge facing CSS and/or OSS companies at present is how to develop the software and complete all the required tests on time. In the past, Amit et al. [4] reported that the number of OSS projects is growing at an exponential rate. But it can be found that the software size-change has a similar tendency with the interest rate change in Economics. Samuel et al. [5][6] once used the asymmetric Laplace distribution (ALD) model to analyze the interest rate and currency exchange rate in finance.

In this study, we have proposed a method of multi-model combinations to analyze the program size variation in OSS [7]. We utilized two weighted combination methods, equal and dynamic, for reliability estimation of real OSS data. Three probability distribution models, namely the ALD model, the normal distribution (ND) model, and the Laplace distribution (LD) model, were selected as the candidate models for model combinations. Actually, these models represent different features: the ND and LD models are usually used in the research fields of nature and science for real-valued random variables, and the ALD model was used in observing the asymmetrical phenomena such as the exchange rate and interest rate change.

Specifically, we first studied the performance comparisons between the single model and the proposed model combinations regarding the distribution of the software size-changing rate. Additionally, we further adopted a modified dynamic weight decision approach for the proposed model combination. In addition, we compared the single model to the equally-weighted combined model using different weight decision approaches. The proposed

weight decision approach may be considered as another way to determine the optimal single software size-change rate distribution model. We collected the number of LOC edited per month for OSS systems, and transformed the datasets into software size-change rate per month for OSS systems.

The rest of the paper is organized as follows. Section 2 gives a brief review of the existing literature, which mentions the related works about the software size-change rate in OSS systems, and introduces the concepts of model combination. Section 3 illustrates the dynamic weight decision approach and the proposed modified Bayesian inference dynamic weight decision approach (MBIWDA). Section 4 shows the experimental results of the OSS data collected from Ohloh.net using the proposed approach. We discussed different weight decision approaches in weighted combinational models and compared the prediction performance between the various combined models. Finally, section 5 summarizes the experimental results and discusses potential research directions in the future.

II. Related Works

2.1 Software Size Estimation

Software quality control (SQC) and software quality assurance (SQA) typically rely heavily on fault removal and forecasting. If developers and/or managers could identify the most error-prone components that are difficult to maintain early on, they would be able to optimize testing-resource allocation, enhance defect removal efficiency, and increase fault detection effectiveness, etc. Thus the customer-detected faults after delivery could be greatly reduced. Software project data could be systematically collected and analyzed during testing and operational phases, and they are assumed to provide additional information for the developer's reference.

OSS has become an essential and primary way to develop software. In the software market, some software products reuse OSS components (e.g., web servers, internet browsers, client email, etc.). We have to take advantage of one probability distribution model to observe its current state and estimate the future development of OSS if we want to understand how OSS changes and grows in the future.

The distribution of file size has been discussed intensely in recent years. Some research reported that the size is lognormally distributed. Additionally, several researchers have also shown that in some cases power law distribution offers a better explanation [8]. For instance, the Pareto distribution model and related distribution models are widely used in social, scientific, and many other types of observable phenomena. Therefore, the "80-20 rule" may be true, which argues that 80% of the effects results from 20% of the causes. We can attribute 80% of the faults to 20% of the modules in software engineering [9][10][11].

However, it is noted that previous research may not have come to an explicit conclusion about the distribution of file sizes. Another important issue is the growth rate of the software size. In the past, some research reported that the number of OSS projects was growing at an exponential rate. Nevertheless, less research focused on the single OSS size-change rate fitted by which probability distribution model.

Let us consider a motivating example to explain this part. Here we assumed that the software size-change rate can be presented by the natural logarithm of the line of code (LOC) ratio for two consecutive months defined as follows:

$$Y_m = \log(i_m / i_{m-1}) \tag{1}$$

where i_m is the LOC of the month m, and the resulting values of the loragithmic Y_m is the software size-change rate in month m. The Mozilla Firebug LOC whole version size-change rate data per month from May 2008 to July 2012 were collected and presented in Fig. 1(a). As seen from Fig. 1(a), we found that there are no negative values in this data set. Fig. 1(b) also depicts the Mozilla Firefox whole version LOC size-change rate data per month from April 2002 to July 2012. As shown in Fig. 1(b), some negative values existed in the collected data. From Figs. 1(a) and 1(b), it can be seen that the traditional log-normal distribution model or the Pareto distribution model may not be appropriate for handling software size-change rate data which could have both positive and negative values in some cases.

But we can find that from Figs. 1(a) and 1(b), the software size-change rate demonstrates a similar trend to the interest rate change in economics. Figs. 2(a)-2(c) further plots the number of size-change rate versus size-change rate for Apache HTTP Server, Eclipse Platform, and Ubuntu. It is obvious that these distributions are quite skewed or fat-tailed and peaky. In the past, the asymmetric Laplace distribution (ALD) model was commonly used to analyze stock yield, interest rate, and currency exchange rate, among other factors in economics. Samuel et al. [5][6] once used the ALD model to analyze the interest rate and currency exchange rate in finance. Klein [12] studied the yield rates of 30-year Treasury bonds from 1977 to 1993, finding that the empirical distribution was too "peaky" and "fat-tailed". Kozubowski and Podgórski [13] proposed an ALD model for analyzing interest rates, proving that this relatively simple model is able to capture the peakedness, fat-tailedness, skewness, and high kurtosis observed in the data. The *Probability Density Function* (PDF) of the ALD model is given as follows:

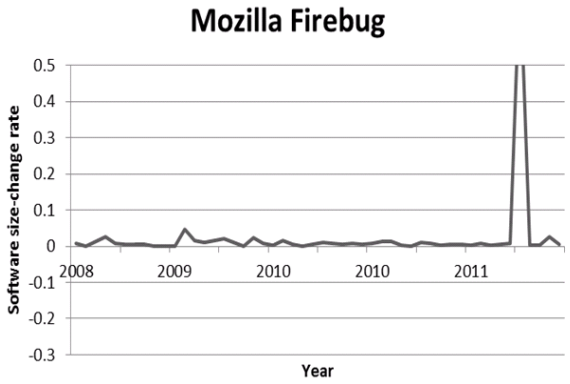
$$f_{ALD}(x) = \frac{1}{\sigma}\frac{\kappa}{1+\kappa^2}\begin{cases}\exp(-\frac{\kappa}{\sigma}x), & \text{for } x \geq 0\\ \exp(\frac{1}{\sigma\kappa}x), & \text{for } x < 0\end{cases} \tag{2}$$

where κ is a skewness parameter and σ is a scale parameter. In addition, the Laplace distribution (LD) and the normal distribution (ND) were also widely used in reliability engineering, economics and in the finance fields [14][15], and their PDFs are defined by

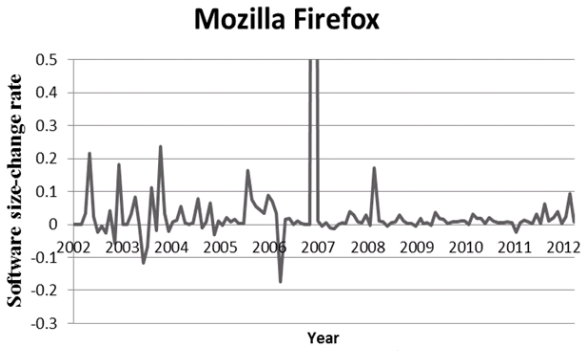
$$f_{ND}(x,\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{3}$$

$$f_{LD}(x \mid \mu, b) = \frac{1}{2b}\exp(-\frac{|x-\mu|}{b}), \tag{4}$$

where μ is the mean, σ is the standard deviation, and b is the scale parameter. But the practical problem is that sometimes the selected models disagree in their desired predictions, while no single model can be trusted to provide consistently accurate results across various applications [16]. In the following, the ALD model, the LD model, and the ND model will be selected as the candidate models and we will use the methods of multi-model combinations to predict and analyze the size distribution and size-change rate of OSS.

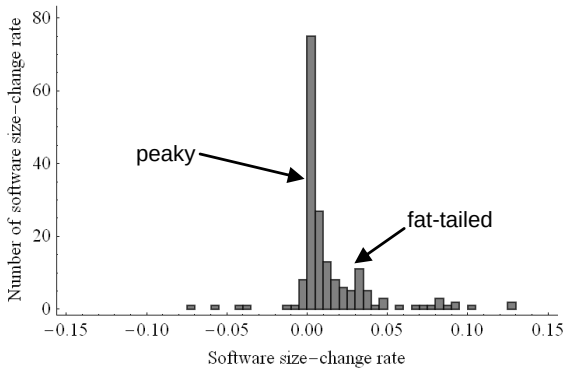


(a) Mozilla Firebug.

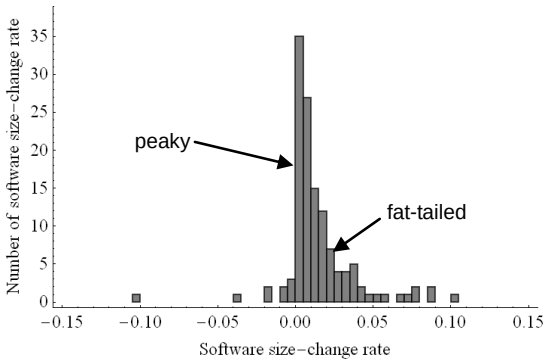


(b) Mozilla Firefox.

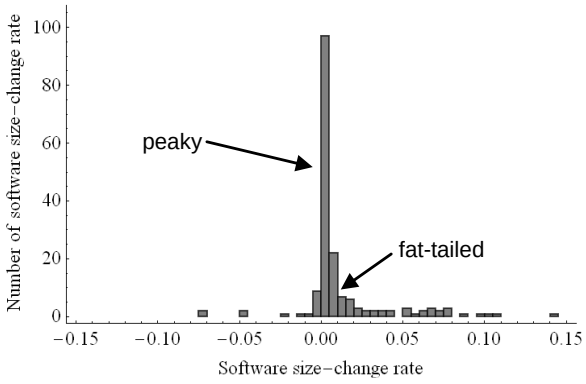
Fig. 1. OSS program size-change rate.



(a) Apache HTTP Server.



(b) Eclipse Platform.



(c) Ubuntu.

Fig. 2. Number of OSS size-change rate vs. size-change rate.

2.2 Weighted Combinational Models

Theoretically, different models represent a range of underlying assumptions that are only applicable to certain validated data or testing conditions. This could pose a challenge for managers in selecting the most suitable model and in assessing the quality measures of software. Rather than selecting the best model, some researchers have also suggested that using more than one model can reduce the risk of only trusting a single model [16], [17]. In principle, applying combinational models could pose an annoying problem in terms of determining proper weights for each model. Some research has suggested that the equally-weighted linear combination methods are suitable for generating a new combined model [18]. It is a simple, low-risk, and high-performance method to solve the above problem. For example, Lyu and Nikora [16][17] used the arithmetic average of each candidate model's prediction as a linearly equally-weighted (LEW) combination prediction. Later, Su and Huang [19] adopted the neural-network-based approach with dynamic weight decision methods for creating the combinational models. Additionally, Hsu and Huang [7] also reported that the relationship between models is the nonlinear geometric weight (NLGW).

Another relationship between models is the nonlinear harmonic weight (NLHW). In some circumstances, the LEW, NLGW, and NLHW decision approach may not provide more accurate results. Consequently, Hsu and Huang [7] proposed the Genetic Algorithm (GA) to determine the weight assignment. Moreover, some studies also proposed using machine learning techniques, and implemented the adaboosting algorithm to combine the model. These

researches used the adaboosting-based self-combination model (ASCM) and adaboosting-based multi-combination model to improve the reliability estimation of software systems [20]. Other research also suggested that the Bayesian inference dynamic weight decision approach (BIWDA) be used to determine the dynamic weight of the candidate model [21].

It is noted that the BIWDA originated from Bayes' theorem, which depicts the relation between two conditional probabilities that are the reverse of each other [22]. Here we will adopt the Bayes' theorem to calculate the weight for prediction system *r* at prediction stage *j* [21]:

$$w_j^{\ r} = PL_{1,j-1}^{r} / (\sum_{k=1}^{m} PL_{1,j-1}^{k}), \tag{2}$$

where $PL_{1,j-1}^{r}$ is the prequential likelihood from time 1 to *j*-1 of the prediction system *r*, and w_j^r has to comply with the rules of Eq. (3):

$$\sum_{r=1}^{m} w_j^r = 1 \tag{3}$$

We can use both Eq. (2) and Eq. (3) to predict T_j, which is the inter-failure time at stage *j* described as follows.

$$P(T_j < t \mid t_{j-1,\ldots,t_1}) = \sum_{r=1}^{m} w_j^r \hat{F}_j(t) \tag{4}$$

But it also has to be noted that above combined model could have great performance with one specified dataset but exhibit unstable performance for unknown datasets. Therefore, it is hard for us to determine the appropriate weight decision approach for each model. In the following, a modified BIWDA model is proposed.

III. Weighted Combinational Models

3.1 Equal Weight Combinations

First, for the ND, LD, and ALD models, we selected the equal weight decision approach to form a combinational model. The equal weight combination model is the easiest combinatorial method. For each candidate model, the weight is constant and equal. Furthermore, Lyu and Nikora [16][17] suggested that the equally-weighted linear combinatorial method can be used to reduce the risks of depending on a specified model. That is,

$$\hat{E}(t) = \sum_{i=1}^{n} w_i \times E_i(t) \text{ and } w_i = \frac{1}{n}, \tag{5}$$

where *n* is the number of models, $E_i(t)$ is the estimated result (i.e., cumulative number of number of faults) of the *i*th model by time *t*, and w_i is the fixed weight of the *i*th model. Additionally, Hsu and Huang [7] asserted that the relationship between models is nonlinear, and NLGW can be described as follows:

$$\hat{E}(t) = \sqrt[n]{\prod_{i=1}^{n} E_i(t)^{w_i}} \text{ and } \sum_{i=1}^{n} w_i = n \tag{6}$$

The definition of NLGW is based on the nth root of the geometric product of estimated results with exponential weights. Here we will assign a static equal weight for NLGW, known as the nonlinear geometric equal weight (NLGEW), where each candidate model has the same and static weight in the combinatorial model to correspond to the software size-change rate. Finally, the harmonic mean is also suitable for calculating the average rates and is defined as the reciprocal of the arithmetic mean or the reciprocal of a set of positive real numbers [23]. Another relationship that can be found between models is the nonlinear harmonic equal weight (NLHEW), which can be expressed by

$$\hat{E}(t) = \frac{\sum_{i=1}^{n} w_i}{\sum_{i=1}^{n} \frac{w_i}{E_i(t)}} \text{ and } \sum_{i=1}^{n} w_i = n. \tag{7}$$

3.2 Modified BIWDA Linear Combination Model

In practice, there are some drawbacks in the traditional weight decision approaches and these drawbacks influence the estimated results if the accurate weight decision approach is not adopted. For instance, the equally-weighted approach may exhibit better performance in specified applications of software reliability engineering [17]. Moreover, the GA for the weight assignment can avoid the above problem if the equally-weighted decision approach is used, but the drawback is that it cannot be converged in the limited time. Hence, the following section describes the approach we proposed that appropriately reflects the relative importance of the candidate model.

Additionally, we could not directly select the best probability distribution model to satisfy all OSS datasets. The problem was addressed by seeking the combination model that mostly corresponded to the true distribution of the software size-change rate. As mentioned in the preceding section, some widely-used probability distribution models in economics, such as the ND model, the LD model, and the ALD model will be considered as the candidate models for our proposed weighted decision approach.

Here we have assumed that the weights with MBIWDA at time t_j are determined by the past predictive accuracy (ie.t_1,..., t_{j-1}). Moreover, we have considered that each weight of the candidate model was a learning process of earlier predictions. Therefore, the weight of the candidate model was influenced by the previous prediction results. In short, we applied and modified the theory of BIWDA for calculating the weight of each candidate model in this section. In the MBIWDA, the weight for candidate model *i*, at time *t* is similar to Eq. (2):

$$w_t^i = P(M_i \mid t_1,\ldots,t_{t-1}) = \frac{P(t_1,\ldots,t_{t-1} \mid M_i)}{\sum_{k=1}^{m} P(t_1,\ldots,t_{t-1} \mid M_k)} = \frac{p(t_1,\ldots,t_{t-1} \mid M_i)}{\sum_{k=1}^{m} p(t_1,\ldots,t_{t-1} \mid M_k)} = \frac{\prod_{l=1}^{j-1} f^i(x_l)}{\sum_{k=1}^{m} \prod_{l=1}^{j-1} f^k(x_l)} \tag{8}$$

where M_i is a candidate system numbered i. That is,

$$w_t^{\,i} = \frac{PL_{1,t-1}^{i}}{\sum_{k=1}^{n} PL_{1,t-1}^{k}}, \quad (9)$$

where $PL_{1,t-1}^{i}$ is the prequential likelihood regarded as the criterion for judging the predictive performance of each candidate model. Also, this criterion can be viewed as a global comparison of goodness of prediction for one candidate model with another from time 1 to t-1 of the candidate model i, and $PL_{1,t-1}^{i}$ is expressed as:

$$3.3 \quad PL_{1,t-1}^{i} = \prod_{j=1}^{t-1} f^{i}(x_j), \quad (10)$$

where $f^{i}(x_j)$ is the probability density function of the candidate model i, and x_j is the software size-change rate at time j.

We utilized both Eq. (9) and (10) to find the closest degree of each candidate model to identify the true distribution at specified time t. Therefore, we obtained the weight of the specified model by time t. Furthermore, we could accumulate the weight of the specified model for time t and calculate the average accumulative weight of the specified model expressed as follows.

$$w_i = \frac{1}{t}\sum_{j=1}^{t} w_t^{j} \text{ and } \sum_{i=1}^{n} w_i = 1 \quad (11)$$

where w_i is the average weight given to the ith model. Finally, we assumed that the average weight w_i can represent the overall and stationary weight of the model i in the combination model. Additionally, we substituted Eq. (11) into Eq. (5) and calculated the estimated results of the dynamic weight combination model, as defined in Eq. (12).

$$\hat{E}(x) = \sum_{i=1}^{n} w_i \times E_i(x) \text{ and } \sum_{i=1}^{n} w_i = 1 \quad (12)$$

where n is the number of models, $E_i(x)$ is the estimated result of the ith model at stage x, and w_i is the modified dynamic weight of the ith model.

IV. Experiment and Analysis

4.1 Selected Data Sets

In this study, the Apache HTTP Server data collected from Ohloh.net are used for the comparison of model performance [24]. The experiments are performed by the following steps:

Step 1. First, we recorded the direct measures of software from Ohloh.net on a monthly basis. In this step, we excluded comments and blankness in codes, and thus fetched the accurate LOC.

Step 2. Using the normalized proportion of the monthly cumulative OSS software size-change rate, the software size-change rate arranged in an increasing order.

Step 3. The parameters of these candidate models were estimated by applying the MLE and LSE.

Step 4. MBIWDA was applied to a linear combination of the two models (of ND, LD, and ALD).

4.2 Evaluation Criteria

For the purpose of comparing the performance of the different combined models, several evaluation criteria were selected and listed as follows.

(a) The *Mean Squared Error* (MSE) is typically defined as[25],[26] ,[27] ,[28], [29]:

$$MSE = \frac{1}{n-\theta}\sum_{i=1}^{n}(X_i - O_i)^2, \quad (13)$$

where θ is the number of the estimated model's parameters. The mean squared error (*MSE*) can be regarded as the average of the squares of errors. The *MSE* value should be as small as possible.

(b) The *Coefficient of Determination* (R^2) is defined as [30]

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(X_i - O_i)^2}{\sum_{i=1}^{n}(O_i - \overline{O})^2}, \text{ where } \overline{O} = \frac{\sum_{i=1}^{n} O_i}{n} \quad (14)$$

The R^2 measures the degree of fitting data by the model. Eq. (14) shows that the R^2 is between 0 and 1. It is noted that the higher R^2 value is considered with a better fitness of the model.

(c) The *Akaike Information Criterion* (AIC) is defined as [31]:

$$AIC = 2k - 2\log[L], \quad (15)$$

where k is the number of parameters in the model and L is the maximized value of the likelihood function for the model. AIC evaluates the performance of MLE. A smaller *AIC* is better.

(d) The *Kolmogorov-Smirnov Test* (K-S test) [32], [33], [34] The K-S test is a nonparametric test for measuring the equality of a random sample with a reference probability distribution. The K-S test calculates the distance between the CDF of the actual data and that of the theoretical probability distribution model. The K-S statistic for a given $F(x)$ is

$$D_n = \sup_x \left| F(x_i) - F_n(x_i) \right|, \quad (16)$$

where sup_x is the supremum of the set of distances. If the empirical distribution function is accommodated by the theoretical distribution function, D_n may approach 0. Moreover, we applied the null hypothesis to test the relationship between the empirical distribution function and theoretical distribution function. The null and the alternative hypothesis are H_0: *The data follow a specified distribution.* H_1: *The data do not follow the specified distribution.* The

hypothesis regarding the distributional form was rejected at a significance level α=0.05 if the test static, *D* calculated by Eq. (16) is greater than the critical value (CV) shown in Eq. (17):

$$CV_{(1-\alpha,n)} = 1.358/\sqrt{n},\ \alpha = 0.05,\ n > 35. \quad (17)$$

In this experiment, we will use arrows (↑,↓) to indicate whether the specified model accommodated the data. In addition, models of better performance should exhibit a smaller K-S statistic.

4.3 Experiments and Discussions

In order to validate the performance of our proposed method, we selected equally-weighted combinatorial model as the basis of comparison. To compare the models, we assigned each model's rank for each criterion and summed up these ranks. The superior models showed lower summed values whereas the worse model had higher summed values. Generally, there are two widely-used estimation methods for model parameters: the Least Square Estimation (LSE) and the Maximum Likelihood Estimation (MLE). The method of MLE estimates parameters by solving a set of simultaneous equations and is better in deriving confidence intervals. However, the equation sets are typically complex and usually have to be solved numerically. On the other hand, the method of LSE minimizes the sum of squares of the deviations between what we actually observe and what we expect. Here we employ the methods of MLE and LSE to estimate the parameters of selected models. Table 1 shows the estimated values of all selected models when the methods of MLE and LSE are used.

When using LSE, the MBIWDA's weight assignment for the ND and LD were $\boldsymbol{W_{ND}=0.003,\ W_{LD}=0.997}$, for the ND and ALD were $\boldsymbol{W_{ND}=0.002,\ W_{ALD}=0.998,}$ and for the LD and ALD were $\boldsymbol{W_{LD}=0.100,\ W_{ALD}=0.900}$, respectively. The estimated results shown in Table 2 suggest that the MBIWDA can achieve excellent performance for reflecting the actual software size-change rate. As we can see in Table 2, the MBIWDA can get the relatively well rank expressed in parentheses in most criteria. That is, the combined models with MBIWDA demonstrated higher performance compared to the combined model with other weight decision approaches. Here we also devoted our attention to the comparisons of model performance when the method of MLE is used. Similarly, we separately calculated the weight using the MBIWDA. The ND and LD were $\boldsymbol{W_{ND}=0.839,\ W_{LD}=0.161}$, for the ND and ALD were $\boldsymbol{W_{ND}=0.106,\ W_{ALD}=0.894,}$ and for the LD and ALD were $\boldsymbol{W_{LD}=0.020,\ W_{ALD}=0.980}$, respectively. As shown in Table 3, the combined model with MBIWDA exhibited better performance in almost every criterion as did the LSE method in this dataset. Overall, the combined model with MBIWDA produced relatively high fit values.

Table 1. **Estimated values of model parameter**

ND (LSE)	μ=0.0163, σ=0.0119
LD (LSE)	μ=0.0172, b=0.011
ALD (LSE)	σ=0.0061, κ=0.3504
ND (MLE)	μ=0.0223, σ=0.0651
LD (MLE)	μ=0.0055, b=0.1532
ALD (MLE)	σ=0.0140, κ=0.482

Table 2. **Comparison results (LSE)**

ND+LD				
Criteria	MSE	R^2	K-S Test	Rank
LEW	0.001(3)	0.997(2)	0.082↓(4)	3
NLGEW	0.001(1)	0.997(3)	0.082↓(2)	2
NLHEW	0.001(3)	0.997(4)	0.082↓(2)	4
MBIWDA	0.001(2)	0.998(1)	0.008↓(1)	1
ND+ALD				
Criteria	MSE	R^2	K-S Test	Rank
LEW	0.0004(4)	0.998(4)	0.069↓(4)	4
NLGEW	0.0004(2)	0.998(3)	0.069↓(3)	2
NLHEW	0.0004(3)	0.998(2)	0.069↓(2)	3
MBIWDA	0.0003(1)	0.999(1)	0.059↓(1)	1
LD+ALD				
Criteria	MSE	R^2	K-S Test	Rank
LEW	0.0004(3)	0.998(2)	0.068↓(4)	2
NLGEW	0.0003(2)	0.998(3)	0.068↓(3)	3
NLHEW	0.0004(3)	0.998(4)	0.068↓(2)	3
MBIWDA	0.0003(1)	0.999(1)	0.061↓(1)	1

Table 3. Comparison results (MLE)

ND+LD					
Criteria	MSE	R^2	AIC	K-S Test	Rank
LEW	0.016(4)	0.924(4)	1089(2)	0.345↑(4)	4
NLGEW	0.013(3)	0.934(3)	26268(3)	0.341↑(3)	3
NLHEW	0.012(2)	0.938(2)	51718(4)	0.337↑(2)	2
MBIWDA	0.010(1)	0.953(1)	1034(1)	0.305↑(1)	1
ND+ALD					
Criteria	MSE	R^2	AIC	K-S Test	Rank
LEW	0.003(4)	0.985(4)	2462(1)	0.198↓(4)	4
NLGEW	0.003(2)	0.987(3)	26948(3)	0.189↓(2)	2
NLHEW	0.003(3)	0.987(2)	51435(4)	0.193↓(3)	3
MBIWDA	0.001(1)	0.995(1)	2730(2)	0.129↓(1)	1
LD+ALD					
Criteria	MSE	R^2	AIC	K-S Test	Rank
LEW	0.009(4)	0.959(4)	718(2)	0.256↑(4)	4
NLGEW	0.004(2)	0.978(3)	1704(4)	0.222↑(3)	3
NLHEW	0.004(2)	0.978(2)	-392(1)	0.191↓(2)	2
MBIWDA	0.001(1)	0.996(1)	792(3)	0.116↓(1)	1

V. CONCLUSIONS

To conclude, the present study is an application of multi-model combinations on the software size-change rate. Here we adopted different combinational models to describe the distribution of OSS size-change rates and our findings confirmed that OSS size-change rates can be fitted by the methods of multi-model combinations. We also provided a MBIWDA that determined the weight of the each candidate model. Furthermore, the proposed method was used to conduct experiments from real OSS project. Experimental results show that the proposed method could acquire better performance than other equally-weighted methods for fitting OSS size-change rate data. It has to be noted that our proposed combinations would not be restricted to a particular kind of candidate models or data sources. Additionally, it is also worth noting that the parameter estimations could become more complicated and tedious

with the use of an increasing number of combined models. However, the additional calculations can be fully automated.

ACKNOWLEDGMENT

The work described in this paper was supported by the Ministry of Science and Technology, Taiwan, under Grants NSC 101-2221-E-007-034-MY2, NSC 101-2220-E-007-005, and MOST 103-2220-E-007-022.

REFERENCES

[1] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 7th Edition, 2010.

[2] I. Herraiz, D. M. German and A. E. Hassan, "On the Distribution of Source Code File Sizes," *Proceedings of the 6th International Conference on Software and Data Technologies* (ICSOFT 2011) pp.18-21, Seville, Spain, July 2011.

[3] G. Concas, M. Marchesi, S. Pinna, and N. Serra, "Power-laws in a Large Oriented Software System," *IEEE Trans. on Software Engineering*, Vol. 33, Issue 10, pp. 687-708, Oct. 2007.

[4] D. Amit, and D. Riehle, "The Total Growth of Open Source," *Open Source Development, Communities and Quality*, pp. 197-209, Springer US, 2008.

[5] K. Samuel, T. J. Kozubowski, and K. Podgorski, "The Laplace Distribution and Generalizations: A Revisit with Applications to Communications," *Economics, Engineering, and Finance*, No. 183, Springer, 2001.

[6] S. Gennady, and M. S. Taqqu, "Stable Non-Gaussian Random Processes," *Econometric Theory*, Vol. 13, pp. 133-142, 1997.

[7] C. J. Hsu and C. Y. Huang, "Reliability Analysis Using Weighted Combinational Models for Web-based Software," *Proceedings of the 18th International World Wide Web Conference (WWW 2009)*, pp. 1131-1132, Madrid, Spain, April 2009.

[8] M. Mitzenmacher, "A Brief History of Generative Models for Power Law and Lognormal Distributions," *Internet Mathematics* Vol.1, No. 2, pp. 226-251, 2004.

[9] R. Cooper, and T. J. Weekes, *Data, Models, and Statistical Analysis*, Rowman & Littlefield, 1983.

[10] S. L. Pfleeger, F. Wu, and R. Lewis, *Software Cost Estimation and Sizing Methods: Issues and Guidelines*, Vol. 269. Rand Corporation, 2005.

[11] B.W. Boehm, *Software Engineering Economics*, Pearson Education, 1981.

[12] G. E. Klein, "The Sensitivity of Cash-flow Analysis to the Choice of Statistical Model for Interest Rate Changes," *Insurance: Mathematics and Economics*, Vol. 16, No. 2, pp.187-187, 1995.

[13] T. J. Kozubowski, and K. Podgorski, "A Class of Asymmetric Distributions," *Actuarial Research Clearing House*, Vol. 1, pp. 113-134, 1999.

[14] H. Okamura, T. Dohi, and S. Osaki, "Software Reliability Growth Model with Normal Distribution and its Parameter Estimation," *Proceedings of the IEEE International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering (ICQR2MSE 2011)*, pp. 411-416, Xi'an, China, June 2011.

[15] G. Bottazzi, and A. Secchi, "Why are Distributions of Firm Growth Rates Tent-shaped?" *Economics Letters*, Vol. 80, Issue 3, pp. 415-420, Sep. 2003.

[16] M. R. Lyu, *Handbook of Software Reliability Engineering*, McGraw Hill, 1996.

[17] M. R. Lyu, and A. Nikora, "A Heuristic Approach for Software Reliability Prediction: the Equally-weighted Linear Combination Model," *Proceedings of IEEE International Symposium on Software Reliability Engineering (ISSRE 1991)*, pp. 172 -181, Austin, TX, USA, May 1991.

[18] M. R. Lyu, and A. Nikora, "Applying Reliability Models More Effectively (software)," *IEEE Software*, Vol. 9, Issue 4, pp. 43-52, July 1992.

[19] Y. S Su, and C. Y. Huang, "Neural-network-based Approaches for Software Reliability Estimation Using Dynamic Weighted Combinational Models," *Journal of Systems and Software*, Vol. 80, Issue 4, pp. 606-615, Apr. 2007.

[20] H. Li, M. Zeng and M Lu, "Adaboosting - based Dynamic Weighted Combination of Software Reliability Growth Models," *Quality and Reliability Engineering International*,Vol. 28, Issue. 1, pp.67-84, 2012.

[21] M. Lu, S. Brocklehurst, and Bev Littlewood, "Combination of Predictions Obtained From Different Software Reliability Growth Models," *Predictably Dependable Computing Systems*, ESPRIT Basic Research Series, pp. 421-439, 1995.

[22] G. E. P. Box, and G. C. Tiao, *Bayesian Inference in Statistical Analysis*, Vol. 40. John Wiley & Sons, 2011.

[23] P. S. Bullen, *Handbook of Means and their Inequalities*, Springer, 2003.

[24] http //www.ohloh.net/, accessed 21 March 2013

[25] S. D. Conte, H. E. Dunsmore, V. Y. Shen, *Software Engineering Metrics and Models*, 1986

[26] P. Rook, *Software Reliability Handbook*, Elsevier Science Inc., New York, NY, USA, 1990.

[27] N. F. Schneidewind, "Finding the Optimal Parameters for a Software Reliability Model," *Innovations in Systems and Software Engineering*, Vol. 3, Issue 4, pp. 319-332, Dec 2007.

[28] K. Shibata, K. Rinsaka, and T. Dohi, "PISRAT: Proportional Intensity-based Software Reliability Assessment Tool," *Proceedings of the 13th IEEE Pacific Rim International Symposium on Dependable Computing* (PRDC 2007), pp. 43-52, Melbourne, Victoria, Australia, Dec. 2007.

[29] M. Xie, Q. P. Hu, Y. P. Wu, and S. H. Ng, "A Study of the Modeling and Analysis of Software Fault - detection and Fault - correction Processes," *Quality and Reliability Engineering International* Vol.23, Issue 4, pp. 459-470, Dec. 2007.

[30] G. Keller and B. Warrack, *Statistics for Management and Economics*, Duxbury, 1999.

[31] H. Akaike, "A New Look at the Statistical Model Identification," *IEEE Trans. on Automatic Control*, Vol.19, Issue 6, pp.716-723, Dec. 1974.

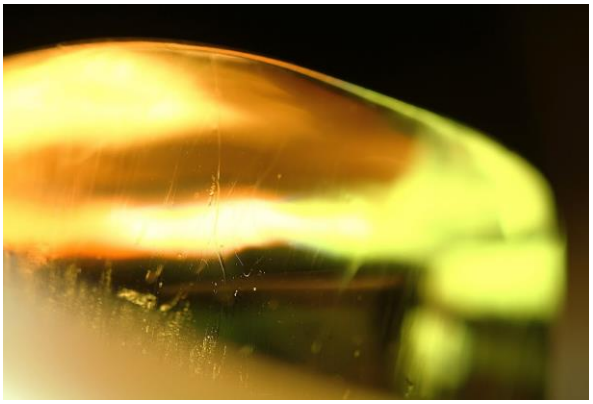
[32] A. L. Goel, "Software Reliability Modeling and Estimation Technique," *Technical Report, RADC-TR-82-263*, Rome Air Development Center, Oct. 1982.

[33] W. J. Conover, *Practical Nonparametric Statistics*, John Wiley and Sons,1980.

[34] G. Triantafyllos, and S. Vassiliadis, "Software Reliability Models for Computer Implementations— An Empirical Study," *Software: Practice and Experience* Vol.26, Issue 2, pp.135-164, Feb. 1996.

Shih-Min Huang *received the B.E. degree (2011) in Department of Information Management from National Central University, Taoyuan Taiwan and the M.E. degree (2013) in Institute of Information Systems and Applications from National Tsing Hua University, Hsinchu, Taiwan. He is currently an IT engineer in the Taiwan Semiconductor Manufacturing Company, Ltd. His research interests include software quality and software measurement.*

Chin-Yu Huang *(M'05) is a Professor in the Department of Computer Science at National Tsing Hua University (NTHU), Hsinchu, Taiwan. He received the M.S. (1994), and the Ph.D. (2000) in Electrical Engineering from National Taiwan University, Taipei, Taiwan. He was with the Bank of Taiwan from 1994 to 1999, and was a senior software engineer at Taiwan Semiconductor Manufacturing Company from 1999 to 2000. Before joining NTHU in 2003, he was a division chief of the Central Bank of China, Taipei. His research interests are software reliability engineering, software testing, software metrics, fault tree analysis, and system safety assessment. He has published over 100 papers in these areas. He received the Ta-You Wu Memorial Award from the National Science Council of Taiwan in 2008. He also received an Honorable Mention Best Paper Award in the 2010 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM 2010). In 2010, he was ranked at the 7th place of top scholars in Systems and Software Engineering worldwide between 2004 and 2008 by the Journal of Systems and Software based on his research on software reliability, software testing, and software metrics. He is a member of IEEE.*



Memory Demand Projection for Guest Virtual Machines on IaaS Cloud Infrastructure

Yi-Yung Chen, Yu-Sung Wu, Meng-Ru Tsai
Dept. of Computer Science
National Chiao Tung University
Hsinchu City, Taiwan
chen.yi-yung@livemail.tw, hankwu@g2.nctu.edu,tw, vtxyer@gmail.com

***Abstract*—Virtualization technology has been widely adopted in IaaS cloud computing environment. Through virtualization, the processor, network, and storage resources can be transparently shared at fine granularity, but the memory still requires explicit coarse-grained provisioning in most cases. Yet it is not always clear how much memory should be provisioned for a virtual machine (VM). It depends on the application workload and characteristics of the underlying platform. We present NIMBLE, a novel system to project the memory demand of virtual machines in IaaS cloud environment. NIMBLE monitors the page swapping activities of a VM at runtime and project its memory demand by indicating the expected execution time of the application workload for each targeted guest physical memory size. This allows more intuitive and cost-effective memory resource provisioning for VMs. The experiment results indicate that NIMBLE can effectively project memory demand for selected benchmark workloads on both Linux and Windows guest VMs. The results also indicate that NIMBLE incurs negligible performance overhead.**

Keywords—virtual machine, memory demand, memory management, introspection, performance, cloud computing

I. INTRODUCTION

Platform virtualization enables consolidation of computation and storage resources in a datacenter environment. Resources are provisioned in the unit of virtual machines (VMs). Several VMs can be consolidated onto a host machine to improve the utilization of hardware resources and reduce cost. Still, a VM should have sufficient resource for its application workload to meet performance requirements. However, accurate provisioning of resources is difficult because in many cases the datacenter operator has few clues about the resource demand of the application workload running inside a guest VM. VM tenants, on the other hand, is not necessarily aware of the characteristics of the underlying platform either.

We develop NIMBLE, which stands for Non-Invasive Memory BottLeneck Estimator, to project the memory demand of guest VMs in IaaS cloud environment. A VM with insufficient memory resource may experience application performance degradation due to thrashing. On the other hand, over-provisioning of guest physical memory will limit the number of VMs that can be accommodated by a host machine. It is thus of great interests for both IaaS cloud service providers and VM tenants to know how much physical memory is actually required by a guest for its given application workload to attain cost-effective performance. Previous work[1-4] focus on estimating the page miss rates and dynamic memory rebalancing of VMs. However, not all application workload are as sensitive to page miss rates. It is not clear if the dynamic memory balancing will improve application performance cost-effectively. Jones et al.[5] proposed using the running time of workload as the metric for guiding the selection of proper physical memory size. However, their work was not targeted for the virtualization environment and it requires modification of the operating system kernel.

Distinct from previous work, NIMBLE can project the memory demand of a VM by indicating the expected application execution time for a targeted guest physical memory size. NIMBLE is designed to be non-invasive in that it does not require additional modifications to a guest VM other than preinstalled standard balloon drivers[6]. NIMBLE is OS agnostic and incurs minimal performance overhead. In the following, we will first have a brief overview of existing virtual machine memory management techniques in Sec. II. We will then introduce the issue of memory demand projection and the design of NIMBLE in Sec III. The evaluation and the conclusion are given in Sec. IV and Sec. V respectively.

II. VIRTUAL MACHINE MEMORY MANAGEMENT

The configuration of a system's physical memory (RAM) is traditionally considered as given and barely changes throughout the system's lifetime. Consequently, most existing memory management schemes are designed to make the best use of a given physical memory configuration. On an IaaS cloud, a hypervisor is used to provision the physical memory of a host machine to the VMs running on the host. The guest operating system of a VM will manage the provisioned memory, and the applications running on the guest systems may further employ additional layers of memory managements such as the garbage collection mechanism in JVM-based applications. Following is a summary of existing memory management schemes that have been proposed for virtual machines running in a hypervisor-based IaaS cloud environment.

A. Guest-Level Memory Managment

Guest operating systems commonly employ virtual memory management[7]. Memory pages that do not fit in the guest

physical memory are swapped to backing stores such as a disk. Virtual memory management will swap the memory pages back to the physical memory before access to the pages can be made. In general, when a guest is under memory pressure, an increase of page swapping activities can be expected. On the other hand, the execution of the application workload on the guest is very likely to be bound by the I/O due to the excessive page swapping activities. As a precondition, we may also expect high utilization of the guest physical memory. Otherwise, the virtual memory manager of the guest would have unnecessarily swapped out memory pages that could still be retained in the guest physical memory.

Most of the virtual memory managers used by the guest operating systems follow LRU-like page replacement algorithms[7]. Roughly speaking, they work by swapping less frequently used pages to the disk so that the physical memory space can be utilized efficiently for more performance critical workload such as the so-called working sets and the memory pages that are frequently accessed. To determine the memory usage of a guest, one may use VM introspection[8] to inspect the memory usage statistics as maintained by the guest virtual memory manager. For example, in the Linux kernel, the `totalram_pages` field tracks the total number of memory frames that can be used by the kernel. The `vm_stat` structure maintains records about the numbers of free memory pages and dirty memory pages. The `swap_info` structure maintains records of free swap size and total swap size. All these records are kept in the guest kernel memory space. Their memory offsets can be looked up through the kernel symbol `System.map`. However, it is not straightforward to determine the amount of memory as required by the guest by reading these usage statistics. It is neither clear how a change of the provisioned memory size would affect the performance of the guest. This is the memory demand projection problem we will introduce soon in Sec. III.

B. *Host-Level Memory Management*

The guest physical memory of a VM is provisioned by the hypervisor through software shadow page table or hardware-based nested paging such as AMD NPT[9] or Intel EPT[10]. At the most basic form, the provisioning simply maps a portion of the host physical memory to a guest VM, and the mapping will remain static throughout the lifetime of the VM. However, as it is not always possible to estimate the memory demand of a VM accurately beforehand, dynamic mapping mechanisms that can reclaim unused guest physical memory pages (and give them back to the guest on demand) have been proposed. *Memory ballooning* relies on a balloon driver[11] installed in the guest that steals unused memory from the guest when host physical memory is low. The process is referred to as the inflation of balloon. On the other hand, when host physical memory is abundant, space may be given back to the guest through deflation of the balloon, which is essentially a deallocation of the guest memory by the balloon driver. Another approach of reclaiming guest physical memory is *hypervisor swapping*[12]. In *Hypervisor swapping*, the hypervisor can swap out guest physical memory pages to a swap file. Should access to a swapped-out page is attempted by the guest, a page fault at either the shadow page table or the hardware NPT/EPT will be triggered. The hypervisor will then swap in the page back to the physical memory. Finally, there are also technologies such as *memory compression* and *transparent page sharing*[12] that aim at reducing redundant data in the physical memory.

III. Memory Demand Projection

Traditional memory management can be viewed as a reactive approach to virtual machine memory management in the sense that the software stack running the VM (and indirectly the VM tenant) focuses on the goal of making the best use of the provisioned guest physical memory. The VM tenant may request for more or less physical memory by changing the VM's configuration, and the software stack will re-adapt to the new configuration. However, it is not always obvious whether a VM should be given more or less physical memory. Due to inherent complexity of the software stack and its intertwined memory management mechanisms (Sec. II), even an experienced system administrator with comprehensive memory usage statistics logs at hands can have a hard time determining the right amount of physical memory as required by a guest VM. On the other hand, for a public IaaS cloud, the physical memory is still considered as valuable resource. It is generally not practical to assume a tenant can always begin with the maximum guest physical memory size setting and gradually reduce the size till the degradation of application performance becomes noticeable (as a practice for attaining cost-effective performance for the workload on the leased VM).

We define the memory demand of a guest virtual machine as the minimum amount of guest physical memory that is required for its application workload to reach an acceptable level of performance. Assuming the execution of application workload `W` on a virtual machine `V` that is equipped with `x` amount of guest physical memory, the time it takes for the execution of the workload to complete is defined as $\mathbf{T_V}(W, x)$, which is assumed to be a monotonically decreasing function with respect to `x`. We define the memory demand $\mathbf{MD_V}(W)$ as

$$\mathbf{MD_V}(W) := \inf\{x: \mathbf{T_V}(W, x) - \mathbf{T_V}(W, \infty) \leq T_{thres}\}$$

,where T_{thres} is a pre-determined positive constant

Note that memory demand is not necessarily the size of the guest virtual memory as allocated by the workload. One has to consider how often each memory page is accessed by the workload as well when reasoning about the execution time. On the other hand, the working set size of the workload could be a good approximation of the memory demand. However, the classical working set model and its estimation mechanism were designed for the case where the working set size is smaller than the physical memory size. Also, the working set model by itself lacks quantification of the performance impact on the application workload as a result of changing the guest physical memory size of a VM. Traditionally, all these issues do not cause much concern as a system's physical memory configuration is considered as given and will remain mostly fixed. However, with the advent of virtualization-based IaaS cloud service, the bargaining of physical memory resource (and CPU, network, storage, etc) is now at the core of business. In the following, we will introduce NIMBLE, a system that we built for projecting memory demand of guest VMs on IaaS cloud platform.

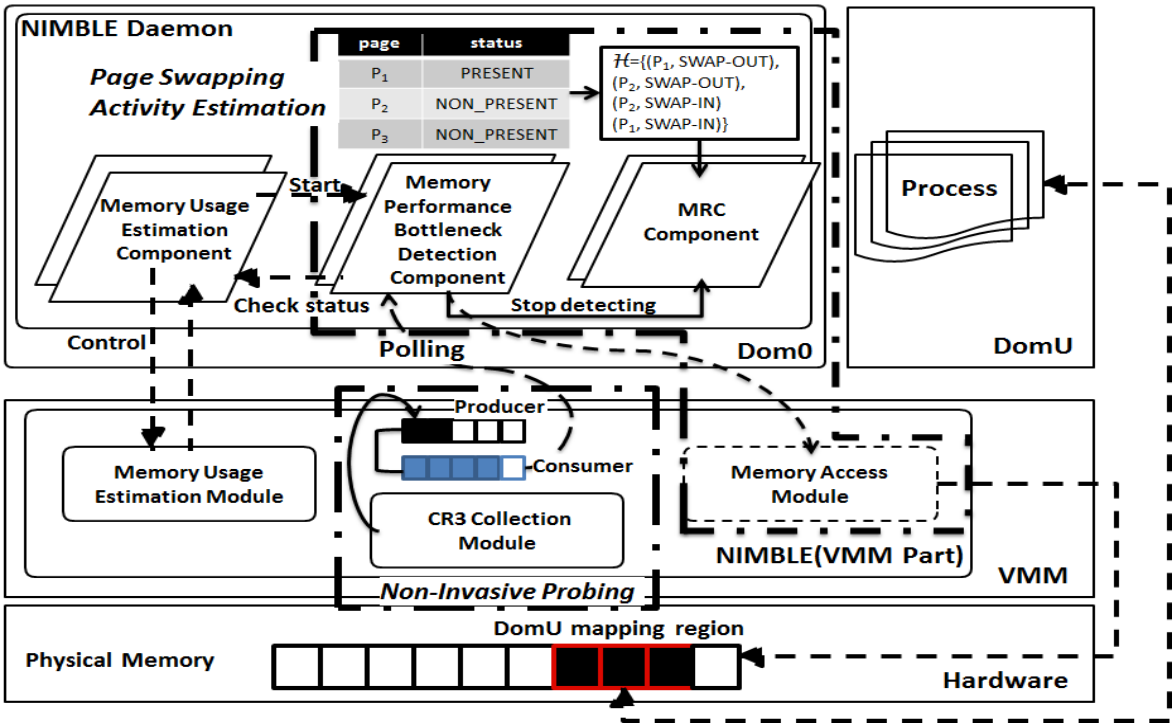


Fig. 1. NIMBLE Architecture

A. *Architecture of NIMBLE*

The architecture of NIMBLE is given in Fig. 1. NIMBLE consists of a daemon program in the management virtual machine Dom0[11] and a bunch of modules within the hypervisor. The hypervisor modules include the *memory usage estimation module*, the *CR3 collection module* and the *memory access module*. The *memory usage detection module* constantly monitors the memory utilization of a VM, the *CR3 collection module* periodically samples the values of the CR3 register, and the *memory access module* allows the NIMBLE daemon to introspect a DomU VM's memory space.

NIMBLE monitors a DomU VM for high memory utilization through the *memory usage estimation module*. The *memory usage estimation module* uses the same technique employed by Vmware[12]. During each operation cycle, it first disables the access permissions of some randomly selected pages in the EPT. When access to these pages are attempted, corresponding page fault events will be recorded, and the *memory usage estimation module* can derive the memory utilization level based on the frequencies of the recorded events. The technique is shown to incur minimal performance overhead. Only when a VM's memory utilization exceeds a threshold will NIMBLE proceed with the memory demand projection procedure, which we will discuss in the following sections.

B. *Non-Invasive Probing*

NIMBLE probes a VM to collect the performance readings needed for the projection of the memory demand of the VM. The probing mechanism assumes the VM is equipped with a balloon driver[6] (or equivalent mechanisms that can be used to dynamically adjust the amount of available guest physical memory). Other than the prerequisite balloon driver, the probing mechanism is OS agnostic and requires no additional modifications to guest VMs. We also designed the probe in such a way that it incurs minimal overhead on the guest VM.

The probing mechanism works by hooking Xen hypervisor's VCPU scheduler. The hook is referred to as the *CR3 collection module*. Right before each VCPU is about to be scheduled for running, the hook will read the CR3 register value from the VCPU register record and append the CR3 value to the tail of the *CR3 event queue* maintained by NIMBLE in the hypervisor. On x86, the CR3 register is used for storing the base address of the root page table. By collecting the CR3 register values, NIMBLE learns about the activations of the guest memory address spaces (i.e. each address space corresponds to a set of pages indexed by the page table pointed by CR3). The activation of memory space is an important clue for projecting memory demand as it indicates not only an extent of memory that is used by the guest but also how often the memory is being used.

The NIMBLE daemon running in Dom0 will read the CR3 values from the *CR3 event queue* (sequentially from head to tail). For each CR3 value read from the queue, the NIMBLE daemon will asynchronously traverse the page tables pointed by the CR3 value. The NIMBLE daemon will inspect each page table entry and determine the status of the memory page, which can be *present in memory* (`PRESENT`), *swapped out* (`NON_PRESENT`), or *unused* (`UNUSED`). For a memory page `P`, we use the notation `STATUS(P,i)` to denote `P`'s status as inspected by NIMBLE at the i^{th} time. We have the index $i \geq 1$.

C. *Page Swapping Activity Estimation*

Assuming a guest VM employs virtual memory management, when the guest is under memory stress, one can expect increased page swapping activities in the guest. The paging swapping activities will cause degradation of application performance, as the paging itself does not contribute to the progress of the application workload while it does eat up data bus and CPU bandwidths. A well-designed guest virtual memory management mechanism shall avoid unnecessary page swapping. Based on the assumption, we would like to project the memory demand of a guest by

estimating its impact on the page swapping activity of the guest. Specifically, we would like to determine the amount of extra physical memory that is needed for reducing the level of page swapping activity of a guest down below a target threshold.

In NIMBLE, we first build up a *page swapping activity history* $\mathcal{H}$ for the combination of a guest VM and a given application workload. A *page swapping activity history* $\mathcal{H}$:={S_1,S_2,..,S_k,S_{k+1},…,S_N} is a sequence of page swapping events sorted in ascending time order. A page swapping event S_k is a tuple (P, i), which corresponds to the swap-out of a memory page P followed by the swap-in of the same page P that coincides with the i^{th} time of P being inspected by NIMBLE. Formally speaking, that is

$S_k := (P, i) \rightarrow$ status(P,i-2)==PRESENT and
status(P,i-1)==NON_PRESENT and
status(P,i)==PRESENT

After a hypothetical change of the guest physical memory size and re-execution of the application workload, we may expect a different page swapping activity history $\mathcal{H}'$. Pertaining to the problem of memory demand projection, we are especially interested in knowing about the length $|\mathcal{H}'|$ of the hypothetical history. Specifically, we would like to minimize the length in a cost-effective manner.

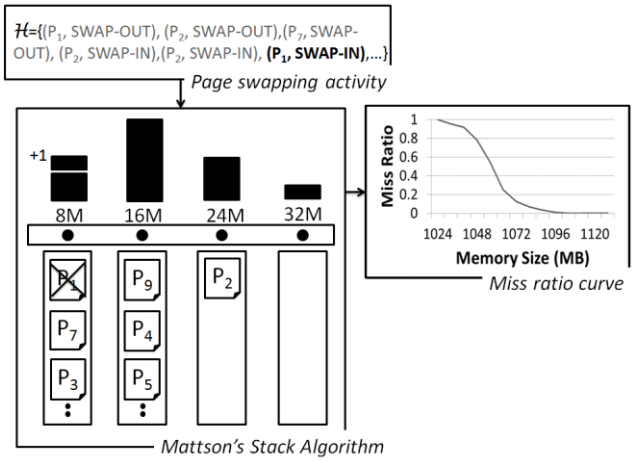


Fig. 2. Mattson's Stack Algorithm and Miss Ratio Curve

Assume that the guest virtual memory manager employs a LRU-like page swapping mechanism, the relation between available physical memory space and expected number of page swapping activities $|\mathcal{H}'|$ can be modeled by the miss ratio curve (MRC)[1] as shown in Fig. 2. The MRC curve can be built by using the Mattson's stack algorithm[1] with input of the page eviction events[13]. Following the algorithm, NIMBLE will sequentially scan trough the *paging activity history* $\mathcal{H}$ of a VM from head to tail. Each swapping-out event will be pushed to the stack. Later, when a swapping-in event is encountered, NIMBLE will look for the corresponding swapping-out event in the stack (it is the nearest swapping-out event for the same memory page that precedes the swapping-in event). If such a swapping-out event is found, its stack depth (distance between the last event pushed to the stack and the swapping-out event) will be calculated as in the Mattson's stack algorithm. In running the Mattson's stack algorithm, one may aggregate consecutive pages into a large pseudo page and run the algorithm with the large sized pseudo page to improve performance at the cost of prediction granularity.

D. Application Exeuction Time Prediction

While the MRC curve can predict the level of page swapping activity for a given combination of workload and a hypothetical guest physical memory size, it is not clear whether the reduced page swapping activity will have significant impact on the application performance. Generally speaking, the application performance can be quantified by the time it takes to complete the execution of the application workload on the given guest VM. In NIMBLE, we model the execution time of running application workload W on a guest VM V with x amount of physical memory by:

$$T_V(W, x) = T_V(W,\infty) + |\mathcal{H}| * T_S$$

, where T_S is the time cost of a page swapping action

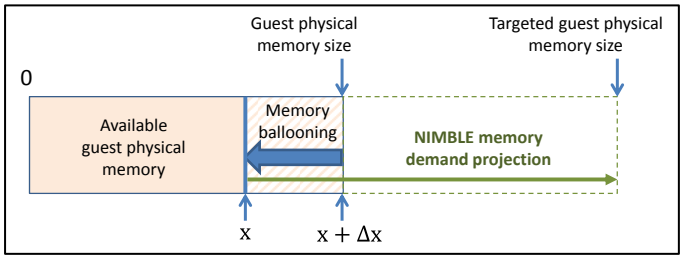


Fig. 3. Application Execution Time Prediction via Memory Ballooning

To determine the values of $T_V(W,\infty)$ and T_S, we may use the balloon driver to adjust the available guest physical memory size by a small Δx amount and rerun the workload to acquire a different page swapping activity history $\mathcal{H}'$ as shown in Fig. 3. We can then solve the following equations to determine the values of $T_V(W,\infty)$ and T_S.

$$\begin{cases} T_V(W,x) = T_V(W,\infty) + |H| * T_S \\ T_V(W,x+\Delta x) = T_V(W,\infty) + |H'| * T_S \end{cases}$$

If the application workload W is uniform, the measurement of $T_V(W,x+\Delta x)$ can be carried out following the measurement of $T_V(W,x)$. There will be no need for rerunning the application. Also, the Δx may be negative. It depends on whether the balloon driver was initially inflated or deflated.

IV. Evaluation

We built a NIMBLE prototype on top of Xen 4.2.1 for evaluation. The testbed environment consists of a server machine with Intel Xeon E5620 16-core processor, 64GB RAM and 1 TB disk. We use Fedora 18 Linux (kernel 3.6.10) and Windows Server 2008 R2 as the guest operating systems. Each DomU VM is given 4 VCPUs. We employed four benchmarks in the experiments. They are h2, tradebeans, y-cruncher[14] and 7zip compression[15]. Both the h2 and the tradebeans benchmarks are part of the Dacapo[16] benchmark suite, which is written in Java. The y-cruncher benchmark is a multi-threaded PI calculation program. The 7zip compression involves the compression of a 551MB rmvb file. The guest physical memory provisioned to the DomU VM is 1024MB for h2, tradebeans, and y-cruncher. For 7zip, we used a DomU VM with 2048MB memory.

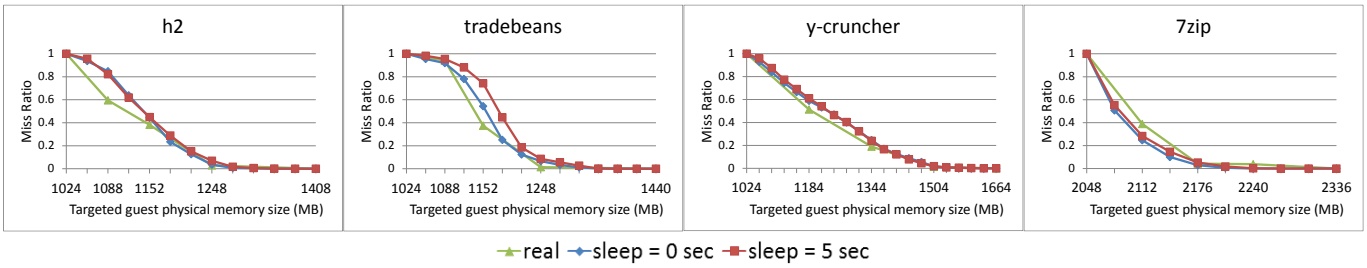


Fig. 4. MRC curve estimation for Fedora 18 guests

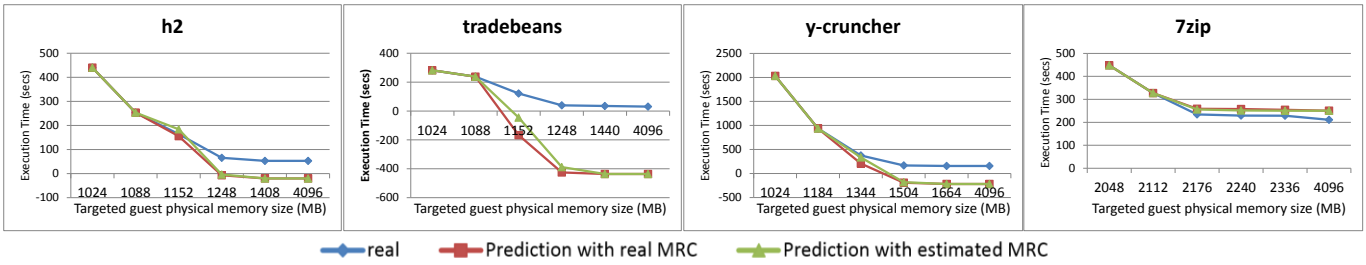


Fig. 5. Execution time prediction for Fedora 18 guests

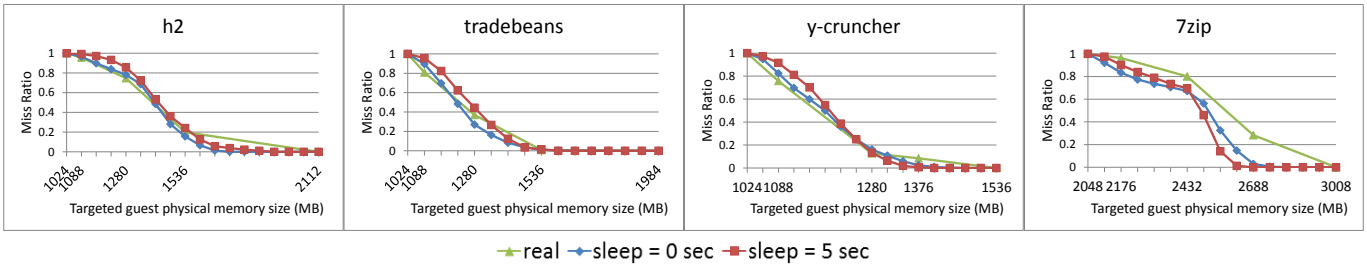


Fig. 6. MRC curve estimation for Windows Server 2008 guests

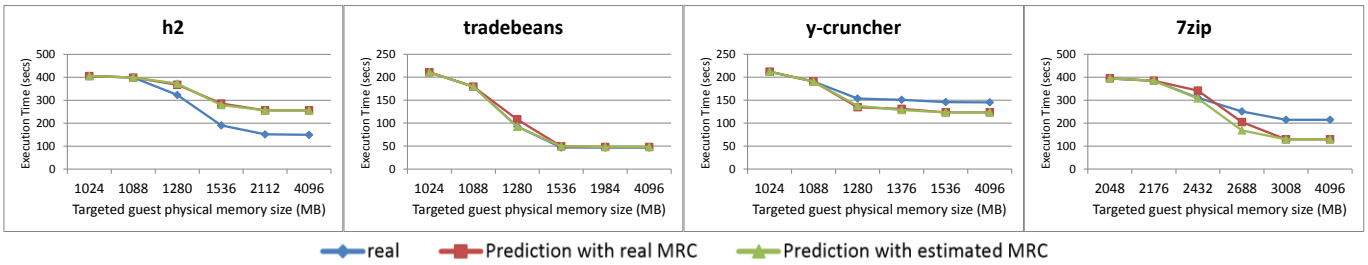


Fig. 7. Execution time prediction for Windows Server 2008 guests

A. *Demand Projection for Linux Guests*

We first look at the MRC curves built from the non-invasive probing and Mattson's stack algorithm. By design, the NIMBLE daemon polls the *CR3 event queue* in the hypervisor for CR3 sample data. When a new CR3 sample data is available, the NIMBLE daemon will immediately scan (`sleep`=0) the page tables and update the *page swapping activity history*. Here we also conducted an experiment with 5 seconds sleep delay (`sleep`=5) between the scans. The estimated MRC curves for Fedora 18 Linux guests are shown in Fig. 4. Overall, we can see that in both cases the estimated MRC curves are pretty close to the real MRC curves. This implies that NIMBLE can accurately project the level of page swapping activities for different guest physical memory sizes.

Fig. 5 presents the application execution time predicted by NIMBLE for Fedora 18 guests. For comparison, we also present the time prediction based on the real MRC curve from Fig. 4. Overall, we can see that the time prediction is pretty accurate for the `7zip` benchmark. Noticeable prediction errors were observed for `h2`, `y-cruncher`, and `tradebeans` as the targeted guest physical memory size goes farther away from the provisioned size. Nevertheless, the trends are still accurate enough for assisting the VM tenants in the selection of VM memory size. On the other hand, the swap cost `Ts` may not be stable enough to be treated as a constant (Sec. III.D). We believe the errors were largely due to the simplified model. However, from a practical point of view, we also notice that the errors in Fig. 5 can be amended by capping the predicted time at 0 (i.e. do not allow negative execution time). This

amendment heuristic will effectively address the excessive time prediction error for the `tradebeans` benchmark and will also improve the prediction results of `h2` and `y-cruncher` as well. We could possibly explore using more complex models (e.g. one with varying swap cost) in the future, but we suspect a complex model will require more comprehensive observations, some of which may require invasive probing and involve high overhead.

B. Demand Projection for Windows Guests

NIMBLE is designed to be OS agnostic, so we also tested the prototype against Windows Server 2008 guests. The estimated MRC curves for the four benchmarks are shown in Fig. 6. The predicted application execution times are shown in Fig. 7. Overall, we see that NIMBLE memory demand projection is effective for Windows Server 2008 guests as well. There is noticeable but relatively minor error in the MRC curve estimation for `7zip`. There are errors in the execution time prediction for `h2` and `7zip`, but none of them are as excessive as the `tradebeans` benchmark on Fedora 18 and the trends are all accurate. On the other hand, we also notice that the heuristic of capping execution time at 0, which we proposed in the evaluation for Fedora 18 guests, will not work here, as all the time predictions are greater than 0.

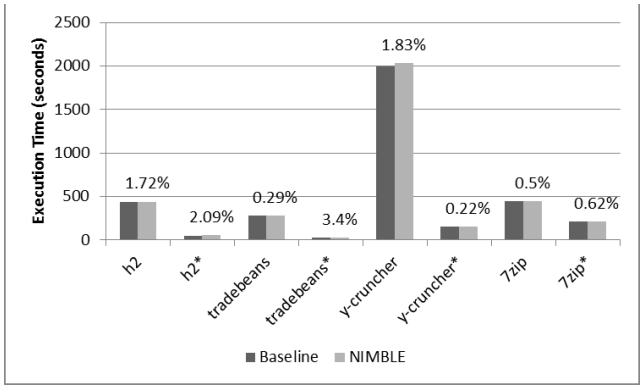


Fig. 8. Performance overhead (Fedora 18)

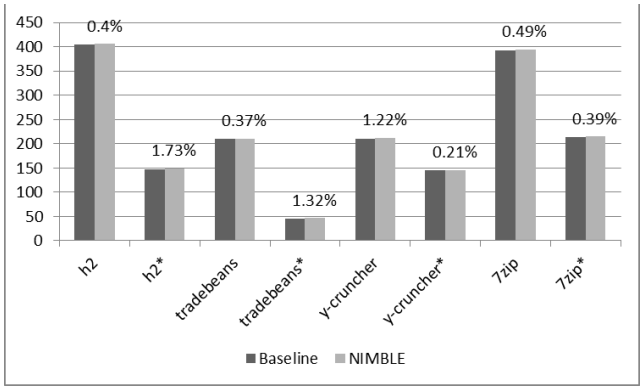


Fig. 9. Performance overhead (Windows Server 2008)

C. Performance Overhead of NIMBLE

The performance overheads incurred by NIMBLE are negligible as shown in Fig. 8 and Fig. 9. As the performance overhead is sensitive to the guest physical memory size, we also conducted a set of experiments, in which the guest VMs were given sufficient memory (4096MB). The corresponding results are superscripted by star signs.

V. Conclusion and Future Work

The use of virtualization technology in IaaS cloud environment enables the provisioning of hardware resources and consolidation of application workload. However, it is not always clear how much physical memory should be provisioned for a VM to achieve cost-effective performance for its application workload. We develop a system called NIMBLE to project the memory demand of a virtual machine on IaaS cloud environment. NIMBLE predicts the application execution time for each targeted guest physical memory size. This allows VM tenants to know if a leased VM requires more memory resource and importantly how much performance improvement is expected.

The experiment results indicate that NIMBLE can effectively project memory demand for selected benchmark workloads on both Linux and Windows guest VMs. The results also indicate that NIMBLE incurs negligible performance overhead. We do notice that there are some cases of noticeable prediction errors. A possible explanation is that the MRC model may not sufficiently capture the behavior of the guest virtual memory management, notably the combination of the `7zip` benchmark and the Windows Server 2008 guest. On the other hand, the swap cost T_s may not remain constant through the timespan of an application workload. We plan to explore these further in the future work.

Acknowledgment

This study is supported by the Cloud Computing Center for Mobile Application of the Industrial Technology Research Institute. The study is also partially supported by the Ministry of Science and Technology of the Republic of China under grant number 101-2221-E-009-076-MY3.

References

[1] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou, and S. Kumar, "Dynamic tracking of page miss ratio curve for memory management," in The 11th ACM International Conference on Architectural Support For Programming Languages and Operating Systems, 2004, pp. 177-188.

[2] W. Zhao, Z. Wang, and Y. Luo, "Dynamic memory balancing for virtual machines," ACM SIGOPS Operating Systems Review, vol. 43, pp. 37-47, 2009.

[3] Y. Niu, C. Yang, and X. Cheng, "Dynamic Memory Demand Estimating Based on the Guest Operating System Behaviors for Virtual Machines," in The 9th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA), 2011, pp. 81-86.

[4] C. Min, I. Kim, T. Kim, and Y. I. Eom, "VMMB: virtual machine memory balancing for unmodified operating systems," Journal of Grid Computing, vol. 10, pp. 69-84, 2012.

[5] S. T. Jones, A. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "MemRx: What-If Performance Prediction for Varying Memory Size," Technical Report 1573, Department of Computer Sciences, University of Wisconsin-Madison2006.

[6] XenServer.org. (2014-9-2). XCP FAQ: Dynamic Memory Control. Available: http://wiki.xenproject.org/wiki/XCP_FAQ_Dynamic_Memory_Control

[7] A. S. Tanenbaum and H. Bos, Modern Operating Systems, 4 ed.: Prentice Hall, 2014.
[8] K. Nance, M. Bishop, and B. Hay, "Virtual machine introspection: Observation or interference?," IEEE Security & Privacy, vol. 6, pp. 32-37, 2008.
[9] AMD, "AMD-V Nested Paging," 2012.
[10] Intel, "Intel® 64 and IA-32 Architectures Software Developer Manuals," 2014.
[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, et al., "Xen and the art of virtualization," in ACM Symposium on Operating Systems Principles, 2003, pp. 164-177.
[12] I. VMWare, "Understanding Memory Resource Management in VMware ESX 4.1," 2010.
[13] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Geiger: monitoring the buffer cache in a virtual machine environment," in The 12th ACM International Conference on Architectural Support for Programming Languages and Operating Systems 2006, pp. 14-24.
[14] A. J. Yee. (2010). y-cruncher - A Multi-Threaded Pi-Program. Available: http://www.numberworld.org/y-cruncher/
[15] I. Pavlov. 7-Zip. Available: http://www.7-zip.org/
[16] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur, et al., "The DaCapo benchmarks: Java benchmarking development and analysis," in ACM Sigplan Notices, 2006, pp. 169-190.

Yi-Yung Chen *received the B.S and M.S degrees in Computer Science from National Chiao Tung University, Taiwan in 2012 and 2014. His research interests include virtualization technology and operating systems.*

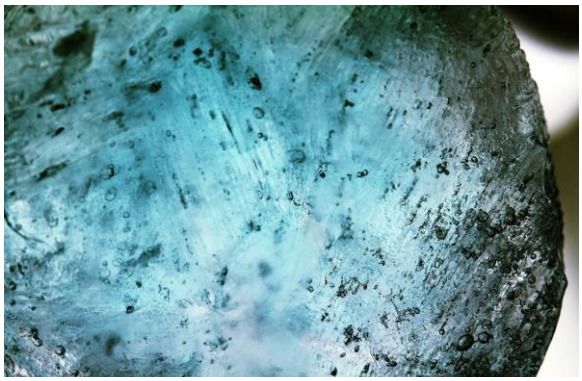
Yu-Sung Wu *received the B.S. degree in Electrical Engineering from National Tsing Hua University, Taiwan in 2002, and the Ph.D. degree in Electrical and Computer Engineering from Purdue University, West Lafayette, Indiana in 2009. He is an assistant professor in the Department of Computer Science, National Chiao Tung University, Taiwan, where he leads the Laboratory of Security and Systems. His research interests include security, dependability, and systems.*

Menq-Ru Tsai *received the B.S. and M.S. degrees in Computer Science from National Chiao Tung University, Taiwan in 2011 and 2013 respectively. His research interests include security, system and front end technology.*

Exploit Generation from Software Failures

Shih-Kun Huang
Information Technology Service Center
National Chiao Tung University
Hsinchu, Taiwan
skhuang@cs.nctu.edu.tw

Han-Lin Lu
Department of Computer Science
National Chiao Tung University
Hsinchu, Taiwan
luhl@cs.nctu.edu.tw



***Abstract*—We normally monitor and observe failures to measure the reliability and quality of a system. On the contrary, the failures are manipulated in the debugging process for fixing the faults or by attackers for unauthorized access of the system. We review several issues to determine if the failures (especially the software crash) are reachable and controllable by an attacker. This kind of efforts is called exploitation and can be a measurement of the trustworthiness of a failed system.**

Keywords—automatic exploit generation; control flow hijaction

I. INTRODUCTION

The failure exploitation methods are to manifest the room for security breaches from the observed failures. The motivation of this type of work is rooted from generating attack inputs to compromise the system and prioritize the bug fixing order. Since failure of software is inevitable and if there are a large number of failures, we need a systematic approach to judge whether they are exploitable. In Miller et al.'s crash report analysis, the authors analyze crashes by BitBlaze [1]. Compared with !exploitable [2], the results show that exploitable crashes could be diagnosed in a more accurate way. Moreover, crash analysis plays an important role to prioritize the bug fixing process [3]. A proven exploitable crash should be the top priority bug to fix. A general review and recent advances are described in [4]. Research insight about exploit generation is analyzed in [5]. Recent work has proved feasibility for common linux and windows applications [6-8], Microsoft office [9, 10] and web applications [11].

Software crash is a special case of control-flow hijacking by the exception handler, raised by the protection hardware. If the program accesses an invalid address (program counter, or memory data access), the memory protection hardware will be signaled. It is due to an incorrect memory update of run-time context or data pointers. If the update is derived from user inputs, run-time context (especially program counter or called instruction pointer and frame pointer) and pointers can be manipulated. If we manipulate the run-time context by deriving the user input, this exploitation process is called failure hijacking. For example, the instruction pointer (or program counter) IP can be related to the failure input with a set of constraints as follows:

$$IP = F(failure\text{-}input)$$

We are able to control the value of IP by resolving the above constraint. The function with path condition is constructed by a failure input feeding to a concolic execution[12].

A failure is the observable event that violates predefined specification. Software crash is a kind of failure that raised from the execution environment, such as run time protection added by the compiler or address protection by the memory management unit. However, many types of failures may not be easily detected unless a predefined specification is enforced by a violation checker. To our problem setting, if the failure is to be controlled by an attacker, we should have tagged the source of the attacker input and monitor the potential outcome to see if the tagged input will eventually influence the failure. There are several types of failures, some of which will raise run-time exceptions, and some of which won't. We view exploit as the manipulation of the software. Exploit generation process is to find input that will control the software. For example, a program written in C is listed in the following:

```
int f(int x) {
  int y = x + 10;
  if (y > 0)
    return y;
  else
    return x;
}
```

If we want to obtain f(x) = 100, what is the value of x? Since in the function f(x), two possible conditions must be explored:

(1) y >0 and x+10 > 0 which is called a path condition and we add a constraint of x+10 = 100. The solution is x = 90.
(2) y <=0 and x + 10 <= 0 and we add another constraint of x = 100. No solution is found for x <= -10 and x = 100.

The final solution is x =90 to obtain f(x) = 100. The above process is called symbolic execution since we treat x and y as symbolic variables and don't assume any concrete values as the

values of x and y. A special case of symbolic execution is to build the first set of path conditions according to an initial input. The variables are still treated as symbolic. If the evaluated results of any path condition is false, the negation is added to the path condition. Otherwise, the original path condition is added. For example, with the initial input of 100 for a concolic execution, f(x) is expressed as x+10 > 0 and f(x)=x+10. If we feed an initial input that will trigger a failure in the software by a concolic execution, we will obtain a set of path conditions that is a precise symbolic model of the failure. For example, if we have a Microsoft office RTF file that will crash the Microsoft Office Word software, we can feed the RTF file as the initial input by concolic execution of Word 2010, and obtain a precise symbolic model like:

Path conditions: $C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge ...$
EIP = $F(I_0,I_1,I_2,...)$
EBP = $F'(I_0,I_1,I_2,...)$
ESP = $F''(I_0,I_1,I_2,...)$

Where C_i is the path condition and I_i is the input to be manipulated.

The following of Fig.1 is a partial listing of the EIP constraint after concolic execution with a failure as the initial input.

(Concat w20 (Extract w8 0 (Add w20 ffffffa9 (Or w20 (And w20 (Add w20 (Or w20 (ZExt w20 (Extract w8 0 (Shl w20 (ZExt w20 (Extract w8 0 (Add w20 ffffffd0 (Or w20 (ZExt w20 (Read w8 9 eip)) 7c810000)))) 4))) 207200) (Or w20 (Or w20 (ZExt w20 (Read w8 a eip)) 7c810000) 20)) ff) 7c810000))) (Concat w18 (Extract w8 0 (Add w20 ffffffa9 (Or w20 (And w20 (Add w20 (Or w20 (ZExt w20 (Extract w8 0 (Shl w20 (ZExt w20 (Extract w8 0 (Add w20 ffffffd0 (Or w20 (ZExt w20 (Read w8 7 eip)) 7c810000)))) 4))) 207200) (Or w20 (Or w20 (ZExt w20 (Read w8 8 eip)) 7c810000) 20)) ff) 7c810000))) (Concat w10 (Extract w8 0 (Add w20 ffffffa9 (Or w20 (And w20 (Add w20 (Or w20 (ZExt w20 (Extract w8 0 (Shl w20 (ZExt w20 (Extract w8 0 (Add w20 ffffffd0 (Or w20 (ZExt w20 (Read w8 5 eip)) 7c810000)))) 4))) 207200) (Or w20 (Or w20 (ZExt w20 (Read w8 6 eip)) 7c810000) 20)) ff) 7c810000))) (Extract w8 0 (Add w20 ffffffd0 (Or w20 (And w20 (Add w20 (Or w20 (ZExt w20 (Extract w8 0 (Shl w20 (ZExt w20 (Extract w8 0 (Add w20 ffffffa9 (Or w20 (Or w20 (ZExt w20 (Read w8 3 eip)) 7c810000) 20))))
….

Fig. 1. The Microsoft Word EIP Constraint with a Failure Input

We are able to control the value of EIP, EBP, or ESP by solving the above constraints. The solution of $I_0,I_1,..,I_n$ are the exploit input of the failure. Failures of stack overflow and uninitialized uses can be modeled in the above similar way. Situations like format string and heap corruption is treated by introducing pseudo symbolic variables for assuming the variable referred by the pointer that is symbolic is probably symbolic. To resolve the pseudo symbolic variables, we first obtain a solution assuming pseudo symbolic variables are symbolic. By searching the memory contents that meet the solutions of the pseudo symbolic variable, we can resolve the values of the symbolic pointers that refer to the pseudo symbolic variables.

II. EXPLOITS WITH SHELLCODE AND ANTI-MITIGATIONS

To launch a practical manipulations of the failures, a set of malicious commands called shell code must be supplied. For example, to execute arbitrary code, the memory location of MEM[X] is injected with the malicious code $D_0,D_1,...$ and the EIP is resolved with the value of X by solving the constraints:

Path conditions: $C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge ...$
MEM[X] = $D_0 = F(I_0,I_1,I_2,...)$
MEM[X+1]= $D_1 = F'(I_0,I_1,I_2,...)$
...
EIP = X = $F''(I_0,I_1,I_2,...)$

The concolic execution is performed under the CRAX framework[10] and depicted in Fig. 2.

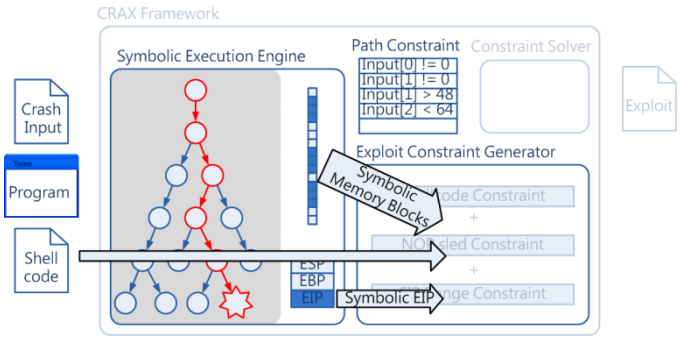


Fig. 2. Concolic Execution for Constructing Failure Constraints

Given a crash input, the target program, and the shell code, the exploit can be produced.

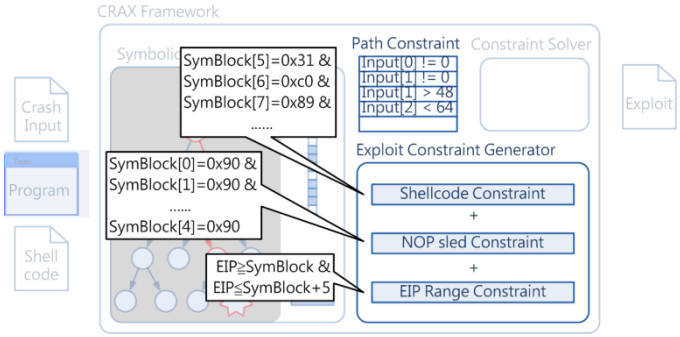


Fig. 3. The Generated Failure Constraints

Other types of attacks such as SQL injection is to find potential manipulations to the query strings to the SQL server. We can also build concolic constraints of web applications by feeding failure inputs. If the query to the SQL server is found to be symbolic, arbitrary SQL injection attacks may be constructed. If the output as the HTML response is symbolic, arbitrary Javascript code is very likely constructed as Cross site script (XSS) attacks. The generation is listed in Fig. 4. We extend these types of exploitation as follows.

a. SQL injection
Path conditions: $C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge ...$
SQL query = Q = $F(I_0,I_1,I_2,...)$

b. Cross site scripting
Path conditions: $C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge ...$
HTML Output response = R = $F(I_0,I_1,I_2,...)$
c. Command injection
Path conditions: $C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge ...$
string to the system() function = S = $F(I_0,I_1,I_2,...)$

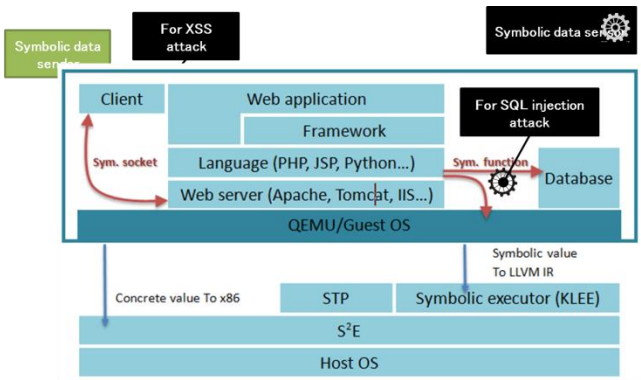


Fig. 4. Concolic execution of Web applications

Even if we can control the instruction pointer, several guards in front of the failures must be escaped. The first guard is the path constraint to reach the failure site without mitigations (surviving security attacks).

Many systems will be with protections such as data execution prevention (DEP) and address space layout randomization (ASLR)[13]. In such a system, executable code may not be injected and a return-oriented programming (ROP)[14] payload built through the application code must be constructed. The exploit constraint is changed into:

Path conditions: $C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge ...$
Search ROP Gadgets in $m_0,m_1,m_2,....$ locations of the application code
STACK[X] = $M_0 = F(I_0,I_1,I_2,...)$
STACK[X+1]= $M_1 = F(I_0,I_1,I_2,...)$
…
The R_1 is the location containing the instruction of "ret" of the code and the starting of the gadget.
EIP = $R_1 = F(I_0,I_1,I_2,...)$

III. The Considerations of the Environment Model

Since the inputs to the target applications are through the operating system environment, for example, the file inputs, environment variables, or network socket, we must be able to feed inputs as symbolic through the OS environment which is called the environment model. There are two possible implementations. The first is to intercept the system calls or revise the standard library functions to mark the inputs as symbolic for concolic execution. This method is used by KLEE[15], AEG[7] and Mayhem[8]. Another solution is to use the whole system emulation like S2E[16] which is based on KLEE and QEMU. By using mmap() system call as in Fig. 5, or RAM disk, we can feed any environment input as symbolic variables to the target applications. The first implementation will be with limited supports of system functions intercepted. The second implementation will support all types of environments. To support an end-to-end approach of exploit generation, environment models of symbolic inputs must be supported. Otherwise, revisions of source is needed like the Heelan's method [6].

```
1.  #include <sys/mman.h>
2.  int main(){
3.
4.      int fd;
5.      pid_t pid;
6.      int i;
7.      char *ptr;
8.      fd = open("./test",O_RDWR);
9.      struct stat b;
10.     fstat(fd , &b);
11.     ptr=mmap(0,b.st_size,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);
12.     ptr[0] = 'a';
13.     s2e_make_symbolic(ptr,b.st_size, "buf" );
14.     if((pid = fork())==0){
15.         execv("./tested program", NULL);
16.     }else{
17.         wait();
18.         close(fd);
19.         s2e_kill_state(0, "program terminated");
20.     }
21.     return 0;
22. }
```

Fig. 5. The Symbolic File Environment by mmap()

IV. Supporting Binary Programs and Dealing with Large Inputs

To support binary programs of exploit generation, we must perform concolic execution over binary programs. Instrumentation over binary programs is needed. There are several concolc execution supports over binary program. Mayhem is based on PIN [17], Catchconv[18] is based on Valgrind[19] and S2E is based on QEMU. Another issue of binary programs for exploit generation is to use concrete address for symbolic memory. Conventional symbolic execution is to use abstract address and these addresses cannot be used for practical exploits. The concolic execution in S2E are treated differently in the host and the guest OS. In the guest OS, all addresses are concrete while abstract in the host OS. Since our exploits are for the guest OS, the concrete addresses meet the need for exploits of binary programs.

A. Dealing with large inputs

The primary steps are to crash the software and control the crash from carefully crafted inputs based on the crash input. There are two techniques to craft the inputs for easier crash manipulation: (1) search the influence over the crash by injecting special patterns of input [20]. (2) find the critical fragments of the input(called hot spot) that will influence the crash (or failure) by tainted input analysis [21]. Since the path conditions, EIP, shell code, and other constraints contain inputs as symbolic variables, the input size will influence the exploit generation process. For example, if the input size is 1024 bytes, there may be several large constraints with thousands of variables. The constraint resolution time is exponentially proportion to the size of input variables as listed in Fig. 6.

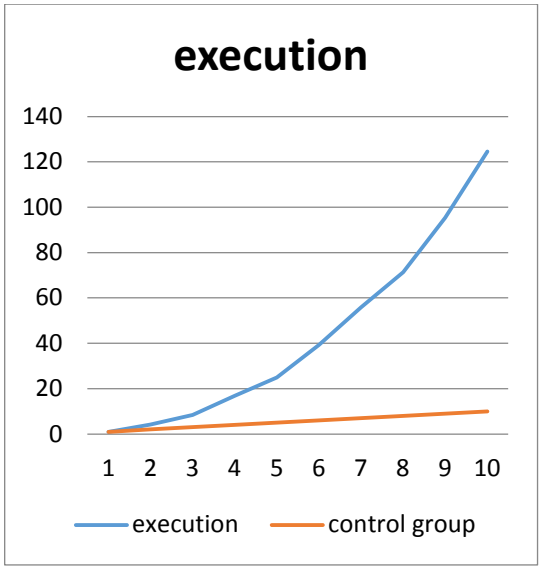


Fig. 6. The Execution time in seconds for symbolic input size from 100 to 1000 bytes

We have proposed an adaptive input selection method by dividing the input into several small size of symbolic inputs to track the influence. Table 1 shows the performance improvement of the adaptive input selection. Originally, if we use the input length of 5000, the explore time is 1388 seconds. If we divide the input into 20 bytes of small chunks, the total explore time is reduced to 11.7 seconds. The improvement is significant.

Table 1. The Performance Improvement of Adaptive Input Selection

Prog.	Input Length	Explore Time	Exploit Gen. Time	Explore Time (Adaptive)	Exploit Gen. Time (Adaptive)
Unrar	5000	1388.5	2569.8	11.7	1.8
Mplayer	145	145.8	151.2	3.3	0.3

V. Conclusion

Failure Exploitation is firstly to construct a set of failure conditions by initially feeding the failure input for concolic execution. We manipulate the failure path condition in the failure exploitation process to hijack the failure for exploitation generation. The failure hijacking is to compute a pre-destined value of instruction pointer (IP) in the relation of IP=F(failure-input). The software exploitation process will be a good measurement of trustworthiness for a failed system.

References

[1] C. Miller, J. Caballero, N. M. Johnson, M. G. Kang, S. McCamant, P. Poosankam, et al., "Crash analysis with BitBlaze," at BlackHat USA, 2010.

[2] M. S. E. C. M. S. S. Team. (2009 March). !exploitable Crash Analyzer - MSEC Debugger Extensions. Available: http://msecdbg.codeplex.com/

[3] D. Kim, X. Wang, S. Kim, A. Zeller, S.-C. Cheung, and S. Park, "Which crashes should I fix first?: Predicting top crashes at an early stage to prioritize debugging efforts," Software Engineering, IEEE Transactions on, vol. 37, pp. 430-447, 2011.

[4] T. Avgerinos, S. K. Cha, A. Rebert, E. J. Schwartz, M. Woo, and D. Brumley, "Automatic exploit generation," Communications of the ACM, vol. 57, pp. 74-84, 2014.

[5] J. Vanegue, S. Heelan, and R. Rolles, "SMT Solvers in Software Security," in WOOT, 2012, pp. 85-96.

[6] S. Heelan, "Automatic generation of control flow hijacking exploits for software vulnerabilities," M.Sc. thesis, University of Oxford, 2009.

[7] T. Avgerinos, S. K. Cha, B. L. T. Hao, and D. Brumley, "AEG: Automatic Exploit Generation," in NDSS, 2011, pp. 59-66.

[8] S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley, "Unleashing mayhem on binary code," in IEEE Symposium on,Security and Privacy, 2012, pp. 380-394.

[9] S. K. Huang, M. H. Huang, P. Y. Huang, C. W. Lai, H. L. Lu, and W. M. Leong, "CRAX: Software Crash Analysis for Automatic Exploit Generation by Modeling Attacks as Symbolic Continuations," in IEEE Sixth International Conference on Software Security and Reliability (SERE), 2012, pp. 78-87.

[10] S. K. Huang, M. H. Huang, P. Y. Huang, H. L. Lu, and C. W. Lai, "Software Crash Analysis for Automatic Exploit Generation on Binary Programs," IEEE Transactions on Reliability, vol. 63, pp. 270-289, 2014.

[11] S. K. Huang, H. L. Lu, W. M. Leong, and H. Liu, "CRAXweb: Automatic Web Application Testing and Attack Generation," in IEEE 7th International Conference on Software Security and Reliability (SERE), 2013, pp. 208-217.

[12] K. Sen, "Concolic testing," in Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, 2007, pp. 571-572.

[13] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh, "On the effectiveness of address-space randomization," in Proceedings of the 11th ACM conference on Computer and communications security, 2004, pp. 298-307.

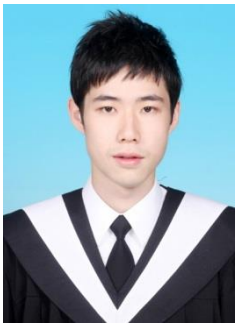
[14] R. Roemer, E. Buchanan, H. Shacham, and S. Savage, "Return-oriented programming: Systems, languages, and applications," ACM Transactions on Information and System Security (TISSEC), vol. 15, p. 2, 2012.

[15] C. Cadar, D. Dunbar, and D. R. Engler, "KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs," in OSDI, 2008, pp. 209-224.

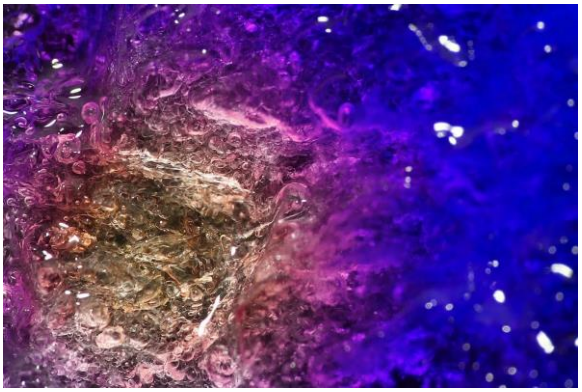
[16] V. Chipounov, V. Kuznetsov, and G. Candea, "S2E: A platform for in-vivo multi-path analysis of software systems," ACM SIGARCH Computer Architecture News, vol. 39, pp. 265-278, 2011.

[17] V. J. Reddi, A. Settle, D. A. Connors, and R. S. Cohn, "PIN: a binary instrumentation tool for computer architecture research and education," in Proceedings of the 2004 workshop on Computer architecture education: held in conjunction with the 31st International Symposium on Computer Architecture, 2004, p. 22.
[18] D. A. Molnar and D. Wagner, "Catchconv: Symbolic execution and run-time type inference for integer conversion errors," Tech. Rep. UC Berkeley EECS, 2007-23, 2007.
[19] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," in ACM Sigplan Notices, 2007, pp. 89-100.
[20] H. Moore, "The metasploit project," http://www.metasploit.com.
[21] T. Wang, T. Wei, G. Gu, and W. Zou, "TaintScope: A checksum-aware directed fuzzing tool for automatic software vulnerability detection," in IEEE Symposium on Security and Privacy, 2010, pp. 497-512.

Shih-Kun Huang *received his B.S. (1989), M.S. (1991) and Ph.D. (1996) in Computer Science and Information Engineering from the National Chiao Tung University, and was an assistant research fellow at the Institute of Information Science, Academia Sinica between 1996 and 2004. Currently he is the deputy director of Information Technology Service Center, and jointly with the Department of Computer Science, National Chiao Tung University. Dr. Huang's research integrates software engineering, and programming languages to study cyber security and software attacks. He is the Principal Investigator of the project on Exploit Generation from Software Crash (CRAX and CRAXweb).*



Han-Lin Lu *received the B.S. degrees in Department of Transportation Technology and Management, and M.S. degrees in Science in Computer Science and Engineering from the National Chiao-Tung University, Taiwan in 2010, and 2012 respectively. He is currently pursuing the Ph.D. degree at the Institute of Science in Computer Science and Engineering of National Chiao-Tung University. His research interests include software quality, network security, and software security.*



The Achilles Heel of Antivirus

Chia-Wei Wang ,Chong-Kuan Chen, Shiuhpyng Shieh,
Department of Computer Science,
National Chiao Tung University, Hsinchu City, Taiwan
{chiaweiw|ckchen|ssp}@cs.nctu.edu.tw

***Abstract*—Antivirus has become one of the most important security guardians against malware for Internet users. The protection provided by antivirus had great success in detecting malware in the past decade. However, modern malware evolves with mutation and anti-antivirus techniques, thereby effectively hindering the detection. The evidence shows that only 51% of the antivirus software can successfully detect the up-to-date malware. Furthermore, due to the lack of protection, antivirus itself is vulnerable to be disarmed once and for all. In this article, the weaknesses of antivirus software are examined, and the possible solutions are proposed to cope with the problems.**

Keywords—antivirus, malware, anti-AV

I. INTRODUCTION

Malware has imposed a noticeable issue of information security ever since the evolution of the computer technology. While computer systems are widely utilized, the sensitive data stored in the storage devices becomes the profit-making target for adversaries. Instead of manually intruding each user's system, adversaries usually spread out the crafted malware in an automatic way as an effective weapon to steal and even sabotage the private data stored in the system.

For instance, the recent CryptoLocker malware, spread in the attachments of social-engineering emails, encrypts all the data in the system storage upon being executed. A victim user is then extorted for the decryption or the data erasure otherwise. In order to prevent users from accessing malware, antivirus software (AV) as the front-line defender is designated to alert users to the malicious file in the real-time paradigm.

In the good old days, AV had great success in the malware detection due to the high barrier to craft new, novel malware, which is undetectable by AVs. However, nowadays there exist various tools that can be utilized to easily mutate an existing malware instance or even customize its functionality for different purposes. Thus, the rapid growth of newly generated malware instances exhausted the resources of AVs to cope with them. Moreover, certain anti-antivirus techniques are also applied to malware for their counterattack against AV. Solely relying on AVs as the only defense against malware is insufficient. In this article, the weaknesses of antivirus are presented, and suggestions as the possible complements to existing AVs are also proposed.

The rest of this paper is organized as follows. Section II overlooks the defects of conventional signature-based detection, which is applied in most AVs against malware. Section III gives the analysis of anti-AV techniques of modern malware. The anticipation of AV vendors is discussed in Section IV. Section V concludes this paper.

II. PITFALLS OF SIGNATURE-BASED DETECTION

Signature-based detection has long been a common approach to the detection of malware. It has high performance and low false positive rate. For each collected malware instance, a security expert analyzes the malware with reverse-engineering techniques. The specific sequence of bytes is then extracted from the malware as its unique signature. A target file matching the signature is recognized as malware. At present, this approach still dominates the detection model of AVs. The effectiveness of signature-based detection is hindered by both malware mutation and the time lag of signature generation. These two factors will be elaborated below, and the solutions will be given.

A. The time race from malware appearence to signature generation

Although the signature-based approach empowers AVs to detect malware, it cannot guarantee the complete coverage of malware detection. One of the main reasons is due to the lack of malware signatures during the time window from the appearance of a new malware instance to its corresponding signature generation. Signature generation is labor intensive and requires significant human effort. For an AV vendor, a malware instance in the wild must be firstly collected so that the corresponding signature for the detection can be extracted. In contrast, malware authors intend to release malware only if the newly generated one does not match any known, recognized signatures of AVs. Consequently, the newly generated malware can evade the detection due to the lack of corresponding signature of AVs. In the undetectable time window, AV's protection is vulnerable and fragile. The active time, which can be referred as the burst of initial infection of malware, is relatively short in the time window. According to the report of FireEye [1], the active time is only two hours. It is crucial for the vendors to equip their AVs with the new signature in such a short period of time. Fig.1 gives the statistical results given by Lastline [2] to illustrate the percentage of existing AVs which successfully detect a new malware instance in terms of days since the first appearance of the malware. The X-axis represents the time window in days whereas the Y-axis denotes the percentage of AVs which

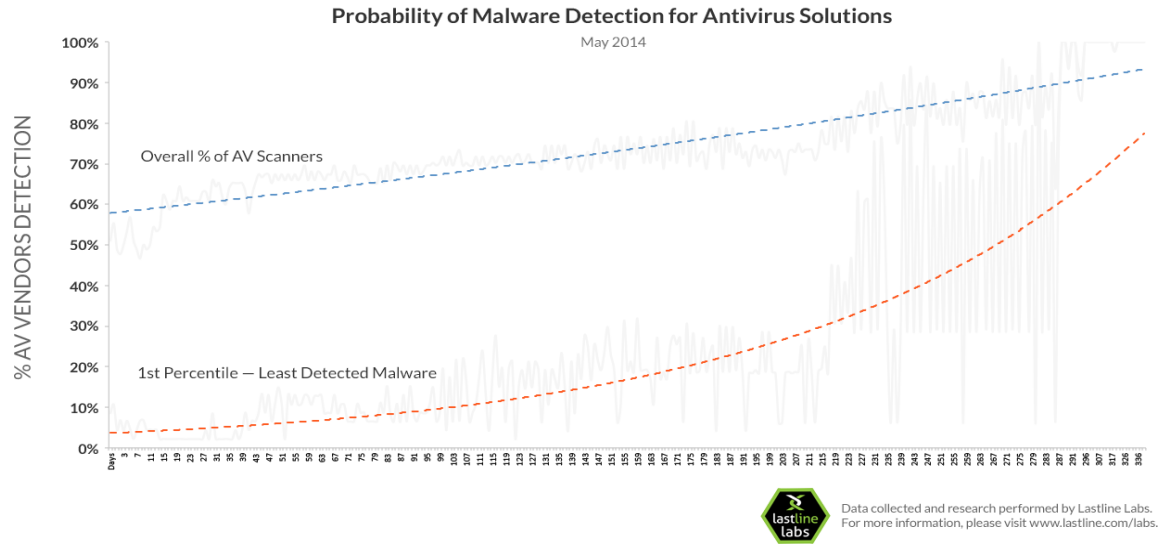


Fig. 1. Percentage of AVs to detect malware in the respect of the number of days that the malware appeared

successfully detect the malware. As shown in the figure, only 51% of the AV vendors effectively detect new malware encountered in the early stage. Certain least detected, tenacious malware even remains undetected by some AVs for a year.

B. Write once, mutate everywhere

To evade the signature-based detection, the mutation techniques are employed by malware authors against AVs. In general, the mutation is realized by replacing the binary code of malware while retaining the equivalent functionality. Therefore, it is not applicable solely using signature with fixed pattern matching to recognize mutated malware. To make detection even harder, malware embedded with a mutation engine is able to self-mutate whenever it propagates. As a result, the "write once, mutate everywhere" strategy adopted by malware authors leads to a large number of undetectable time windows aforementioned, which exhausts AV vendors to develop the signatures for each mutated malware instance [5].

In spite of different appearance of the mutant malware instances, their malicious behavior remains the same. For instance, two mutants from the same origin may both perform the same network transmission. Therefore, the invariant of behavior gives us the opportunity to detect mutated malware. The detection of malware can be significantly improved by using the behavior analysis to complement the conventional signature-based detection.

On the other hand, to shorten the time window before the signature generation, an automatic process should be considered. Heuristic approaches such as machine learning can generate the detection model for known malware automatically. Using the heuristic approaches to detect similar malware, security experts can focus only on the brand new malware. Thereafter, the time windows needed to generate signatures can be diminished.

With runtime behavior as the signature to detect malware, the problem of malware mutation can be mitigated. If a malware instance can be recognized, the effort to further analyze the corresponding mutant can be saved. Meanwhile, the heuristic approach can be conducted to shorten the time windows as aforementioned. With behavior-based signature and the heuristic approach, AVs can be more effective in response to a new malware.

III. ANTI-AV: OFFENSE IS THE BEST DEFENSE

To conceal its activities, malware may attempt to use some anti-AV techniques to attack AVs. We will elaborate these techniques and propose the solutions.

A. Hook sabotage

Modern malware is equipped with anti-AV techniques to disarm AVs and thus circumvent their detection. This countermeasure is mostly achieved by sabotaging the checkpoints pre-installed by an AV. Recall that AVs must intercept certain operations issued by users to prevent him/her from mistakenly triggering malware in the real time paradigm. Therefore, an AV usually injects additional checkpoints, also known as hooks, into the system service procedures for the interception.

Figure 2 illustrates the concept of hooks injected by an AV. As shown in Figure 2 (a), the common design of system service procedure of OS consists of service request, service dispatcher, and service handler. Figure 2 (b) gives the system service procedure patched by an AV installed. The hooks injected at the service dispatcher detours a request issued by a user to the security module of the AV before it can be delivered to the service handler. The request for a system service such as the file read/write operation will be validated if any recognized malware is being interacted. The request is forwarded to the original service handler to serve the request only if the validation pass or aborted otherwise.

TABLE I. SEVEN TERMINATOR PROGRAMS AND THE ANTIVIRUS SOFTWARE THAT THEY CAN CLOSE

Antivirus / Method used	Avast	Avria	Norton	Kaspersky	NOD32
1. Process termination	V	V	V	X	X
2. Mouse simulator	V	V	V	V	V
3. Registry modification	V	V	V	V	V
4. Close Message	V	X	X	X	V
5. Null debugger	V	V	X	V	V
6. Dll unloading	X	V	V	V	X
7. Thread termination	X	X	X	X	X

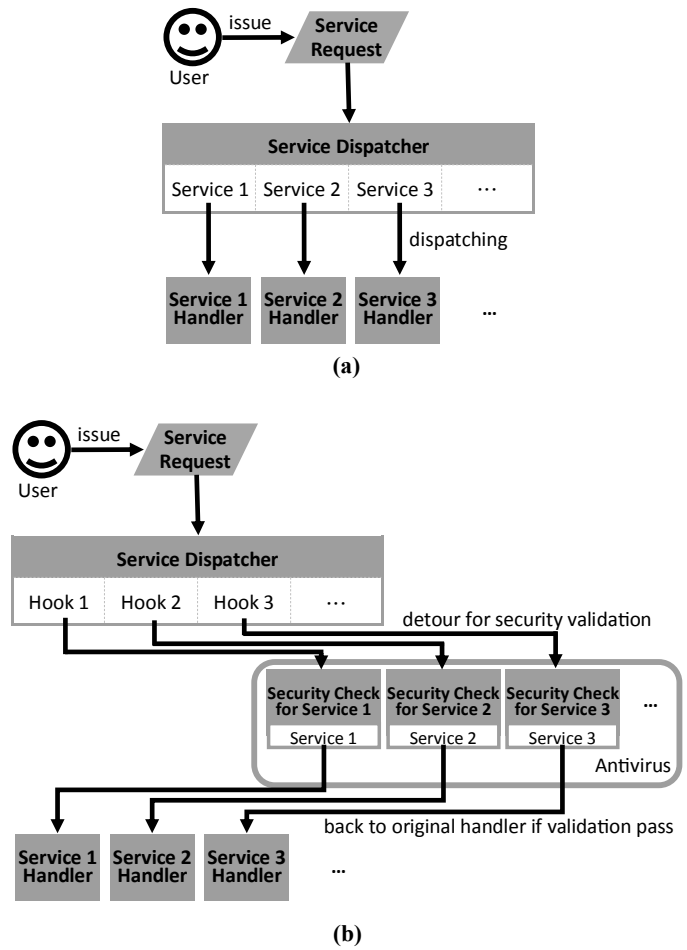


Fig. 2. (a) The vanilla system service procedure (b) The system service procedure with the patch of antivirus for malware detection

The service dispatcher can be referred to as the interface to various system functionalities and is rarely changed for compatibility in different versions of an OS. Considering the applicability, AVs also choose the service dispatcher as the injection points for their hooks. Taking advantage of this convention, malware authors equip their malware with the ability to compare the system service dispatcher with the vanilla one to diagnose the possible presence of an AV. Moreover, by replacing the dispatcher with the vanilla one, the hooks of the AV can be un-installed, leading to the circumvention of the security validation once and for all.

In addition to the hook-sabotaging techniques introduced, other approaches disabling AVs also exist. Table I gives the results when applying 7 common AV terminators against five well-known AVs [6]. The result indicates that even without the in-depth knowledge regarding the system service dispatcher to perform the un-hook patch, it is still possible to disable the AV in the straightforward way.

To prevent AVs from being terminated by anti-AV malware, Hsu et al. proposed the ANSS (ANtivirus Software Shield) [6]. The basic idea is to complement the checkpoints for the security of AVs. In general, the coverage of the checkpoints of AVs is limited to the detection to malicious files

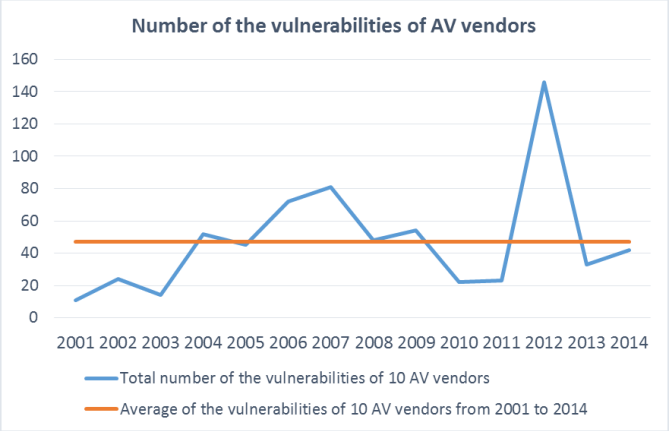


Fig. 3. Number of AVs' vulneraibilties from 2001 to 2014

being accessed. For malware that survives the detection, the AV is completely exposed without additional protection. In addition to the original checkpoints, ANSS further inserts checkpoints into specific system service procedures, which may be utilized to terminate the AV. The security of AV itself can be therefore enhanced.

B. *Attacking AVs' Vulnerabilities*

To deal with increasing security threat, the functionalities of AVs are further enhanced. In addition to the fundamental malicious file scanning, other features such as spam email and web content filtering are also provided by modern AVs. Along with the growth of implementation complexity of AVs, the security vulnerabilities due to programming bugs are inevitably encountered.

Compared to most malware passively waiting to be triggered by users, the behaviors of AVs are much easier to predict due to the fact that AVs scan every suspicious files. As a metaphor, AV is like a fish, trying to eat the bet. The predictable behavior allows adversaries to focus only on the construction of attacks to vulnerabilities without the consideration of the triggering conditions. AVs' vulnerabilities eventually will be discovered and becomes another threat to users.

According to the top 50 software vendors with the highest number of vulnerabilities given by Common Vulnerabilities and Exposures (CVE) [4], which is a database of vulnerabilities, two AV vendors are included along with other software vendors such as Microsoft, VMware, and Apache. The fact that the two AV vendors are on top of the list implies that AVs are amongst the most vulnerable software systems.

Figure 3 depicted the total number of vulnerabilities each year found in AVs developed by ten well-known AV vendors from 2001 to 2014. On average, 47 vulnerabilities are found annually in the last ten years, shown in Fig. 3. The evidence indicates that the security threats caused by vulnerable AVs deserve more attention.

To secure security tools including the AVs from being compromised, a promising trend is to isolate the tools from the environment where the subject program, which could be a malware, is being executed. The virtual machine technology has the best chance to realize this scenario by dividing the system into the guest OS and the host OS. A security tool operating in the host OS can be cloaked by the boundary

between the guest and the host. However, this VM-based implementation encounters great difficulty in addressing the semantic-gap issue. That is, without the assistance of the guest OS, the host-side security tools are required to interpret the raw bytes in the memory of the guest OS into human-readable information before any analysis can be proceeded. In spite of various efforts, may challenges remain unsolved. The operations and functionalities of the security tools hosted in the host OS are still limited. Further enhancement of the VM-based approaches is desirable due to its powerful isolation.

IV. Scattered Malware Information

For a long time, AV vendors work in the standalone paradigm. Each vendor has its own definition to tag an analyzed malware [7]. This information is hardly synced among different AV vendors. Consequently a malware sample detected by an AV can still escape from another. The cooperation of the AV vendors is expected to construct a more comprehensive defense against numerous malware.

VirusTotal [3], which is the web service organizing the malware analysis result of various AVs, shares the collected samples with their cooperated AV vendors. Through the information sharing, the time windows from malware appearance to signature generation can be shortened. This provides better protection to the end users. Similarly, further collaboration of AV vendors such as sharing malware signatures and naming rules can reduce the response time to a new threat.

V. Conclusion

In this article, three issues including security threats, detection strategy, and collaboration of anti-virus software are discussed. The heuristic approaches such as dynamic behavior analysis can be applied along with the signature-based detection for malware detection for both efficiency and accuracy. By setting up additional checkpoints at the early stage of the installation of a clean OS, the anti-AV attack can be effectively eliminated. Moreover, through the virtual machine technology, the AVs have the opportunity to serve its security purposes while being cloaked by the isolation features provided. While the large amount of malware exhausting individual AV vendors, cross-vendor collaboration such as sharing of malware information among different AVs will greatly raise the barrier for malware to circumvent.

Acknowledgments

This work is supported in part by the Ministry of Science of Technology of Taiwan, Taiwan Information Security Center, Industrial Technology Research Institute of Taiwan, Institute for Information Industry of Taiwan, the International Collaboration for Advancing Security Technology, HTC Corporation, D-Link, Trend Micro, Promise Inc., Chungshan Institute of Science and Technology, Bureau of Investigation, and Chunghwa Telecomm.

References

[1] Ghost-Hunting With Anti-Virus , http://www.fireeye.com/blog/corporate/2014/05/ghost-hunting-with-anti-virus.html

[2] Antivirus Isn't Dead, It Just Can't Keep Up http://labs.lastline.com/lastline-labs-av-isnt-dead-it-just-cant-keep-up.

[3] VirusTotal, https://www.virustotal.com/en/about/

[4] Common Vulnerabilities and Exposures List , https://cve.mitre.org/

[5] P. O'Kane, S. Sezer, and K. McLaughlin. Obfuscation: Thehidden malware. Security & Privacy, IEEE, 9(5):41–47, 2011.

[6] Fu-Hau Hsu, Min-Hao Wu, Chang-Kuo Tso, and Chieh-Wen Chen. "Antivirus Software Shield Against Antivirus Terminators", IEEE transaction on information forensics and security, vol. 7, no. 5, October , 2012.

[7] A. Mohaisen and O. Alrawi. "Av-meter: An evaluation of antivirus scans and labels." In DIMVA, 2014.

Chia-Wei Wang *is a PhD student in the Laboratory for Distributed System and Network Security at the National Chiao-Tung University, Taiwan. His research interests include the virtual machine security and the malware research, Win32 especially. Chiawei has a bachelor degree in computer science from National Sun-Yat Sen University, Taiwan. Contact him at cwwang.cs98g@g2.nctu.edu.tw*

Chong-Kuan Chen *is a PhD student in the Department of Computer Science at National Chiao Tung University, Taiwan. His research interests include network security, system security, and malware analysis. Contact him at ckchen@cs.nctu.edu.tw.*

Shiuhpyng Winston Shieh *is a distinguished professor and the past Chair of the Department of Computer Science, National Chiao Tung University (NCTU), and the Director of Taiwan Information Security Center at NCTU. His research interests include reliability and security hybrid mechanisms, network and system security, and malware behavior analysis. He is actively involved in IEEE and has served as the Reliability Society (RS) VP Tech, and Chair of RS Taipei/Tainan Chapter. Shieh received his PhD in electrical and computer engineering from the University of Maryland, College Park. He (along with Virgil Gligor of CMU) invented the first US patent in the intrusion detection field. He is an IEEE Fellow and ACM Distinguished Scientist. Contact him at ssp@cs.nctu.edu.tw.*

Submission Instructions

Authors should use the designated IEEE Reliability Digest Manuscript Central Website to summit their papers.

Please refer to the following steps to submit your papers:

1. Login to IEEE Reliability Digest Manuscript Central. If you have no account, sign up for one.
2. Click "Authors: Submit an article or manage submissions".
3. Please click "CLICK HERE" at the bottom of this page, and you will be brought to the five-step submission process.
4. You need to 1) choose the section that you are going to submit your paper to; 2) complete the submission checklist; 3) enter the comments for the editor, which is optional; 4) save and continue.
5. If you have any supplementary files, please upload them in step 4.

Manuscript Types

Manuscripts for regular issues fit within the scope of the magazine, but are not intended for a special issue. Special issue manuscripts cover a specific topic scheduled on our editorial calendar. Please select the appropriate issue (manuscript type) when uploading your manuscript. For more information and to see upcoming special issue topics, see our Editorial Calendar at http://rs.ieee.org/reliability-digest/author-guidelines.html.

Typing Specifications

The manuscript should be written in Times New Roman in a double-column format. The typical length of the submitted manuscript is 4 single-spaced pages. The text portion of the manuscript should be in 10-point font and the title should be in 24-point font, bold.

Manuscript Length

The typical length of the submitted paper is 4 pages, including text, bibliography, and author biographies. Please note that proper citations are required.

Illustrations

The illustrations in the articles must be cited in the text and numbered sequentially. Captions that identify and briefly describe the subject are needed as well. In order to avoid dense and hard-to-read illustrations, graphs should show only the coordinate axes, or at most the major grid lines. Line drawings should be clear. To prevent potential layout problems from happening, related figures described within the same section of text should be grouped together as parts (a), (b), and so on.

References

All manuscript pages, footnotes, equations, and references should be labeled in consecutive numerical order they are mentioned in the text. Figures and tables should be cited in text in numerical order.

Biographical Sketch

A brief biographical sketch should contain the full title of the paper and complete names, affiliations, addresses, and electronic mail addresses of all authors. The corresponding author should be indicated.

Please provide a short biography and a picture for each author of your paper at the end of your paper.

The short biography should contain no more than 150 words.

Copyright

IEEE Reliability Society does not own the copyrights of the articles published in Reliability Digest. If you wish to reproduce the copyrighted materials, please contact the copyright owners and seek their permissions. The contents in this website should be copied with use of proper citation.

Special Issue Proposal Submissions

For a special issue in Reliability Digest, experts are welcome

to serve as our guest editors. To know more information, please contact Editor-in-Chief on Reliability Digest, Shiuhpyng Shieh: ssp@cs.nctu.edu.tw.