

Current State and Challenges for Model-Based Security Testing

Michael Felderer

Research Group Quality Engineering
Institute of Computer Science
University of Innsbruck, Austria

SECTEST 2015

Graz, Austria

April 13, 2015

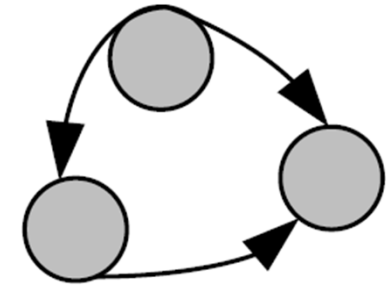
Dr. Michael Felderer



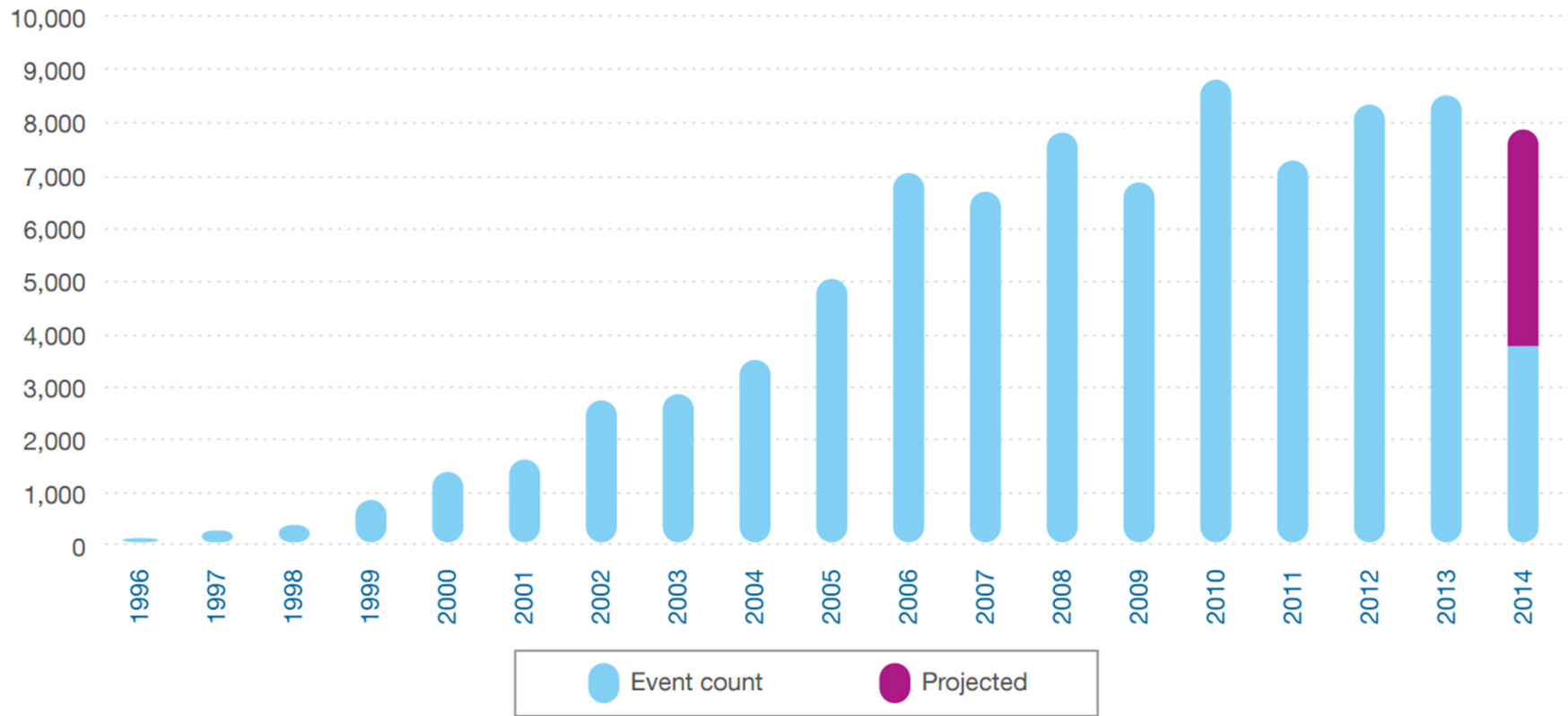
- Senior Researcher at University of Innsbruck
- Research Interests
 - Software and Security Testing
 - Empirical Software and Security Engineering
 - Model Engineering
 - Risk Management
 - Software and Security Processes
 - Software Quality
 - Research-Academia Collaboration
- Professional Experience
 - Software and Security Engineering Consultant for QE LaB Business Services
 - Developer, trainer and consultant for mid-sized ERP provider

Overview

- Motivation
- Model-Based Testing (MBT)
- Model-Based Security Testing (MBST)
- Taxonomy of Model-Based Security Testing Approaches
- Systematic Mapping of available MBST Approaches
- Current State of MBST
- Directions and Challenges for Research and Application of MBST

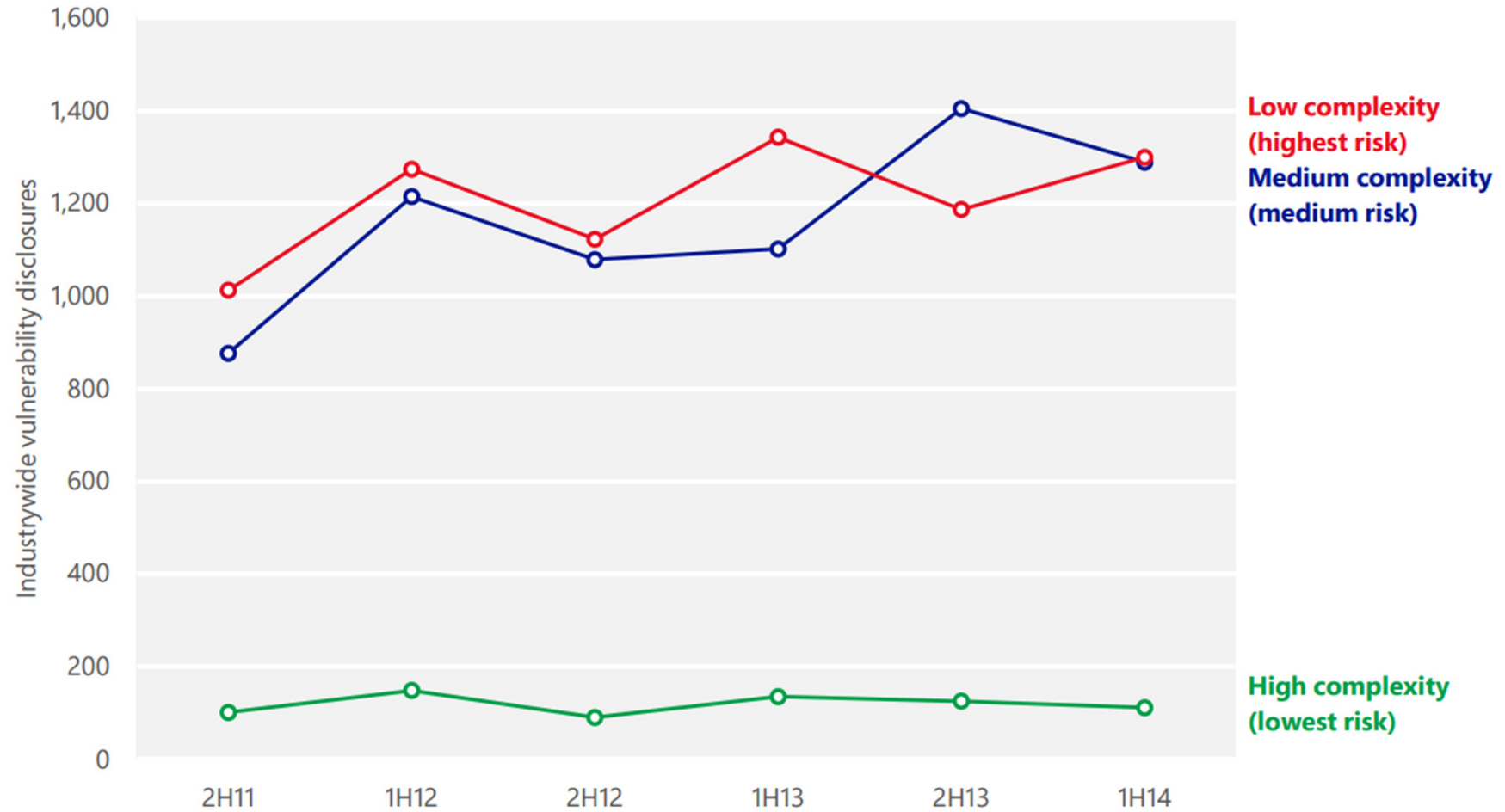


Motivation – Vulnerability Disclosure Growth By Year



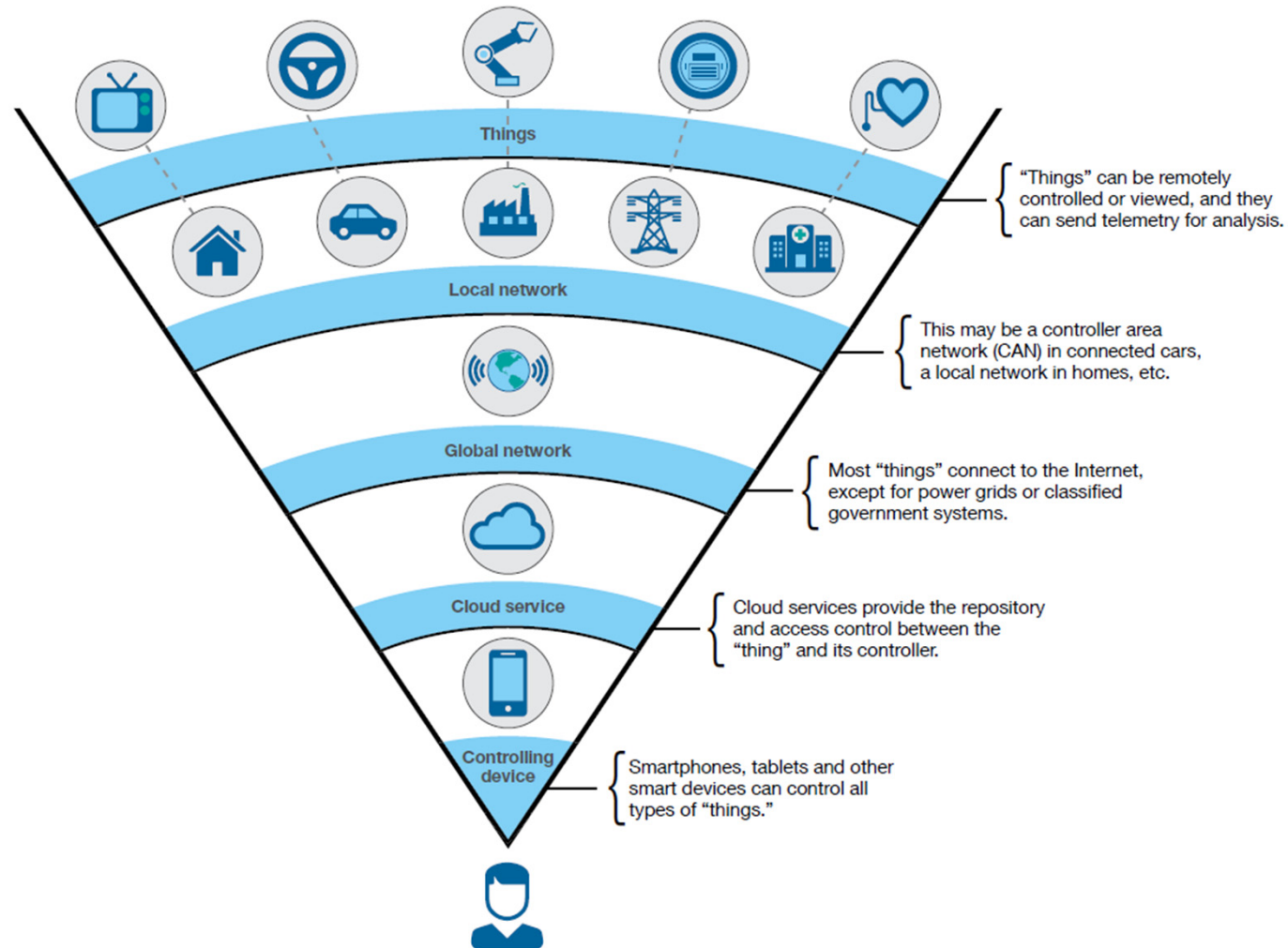
IBM X-Force Threat Intelligence Quarterly, 3Q 2014

Motivation – Vulnerability Complexity



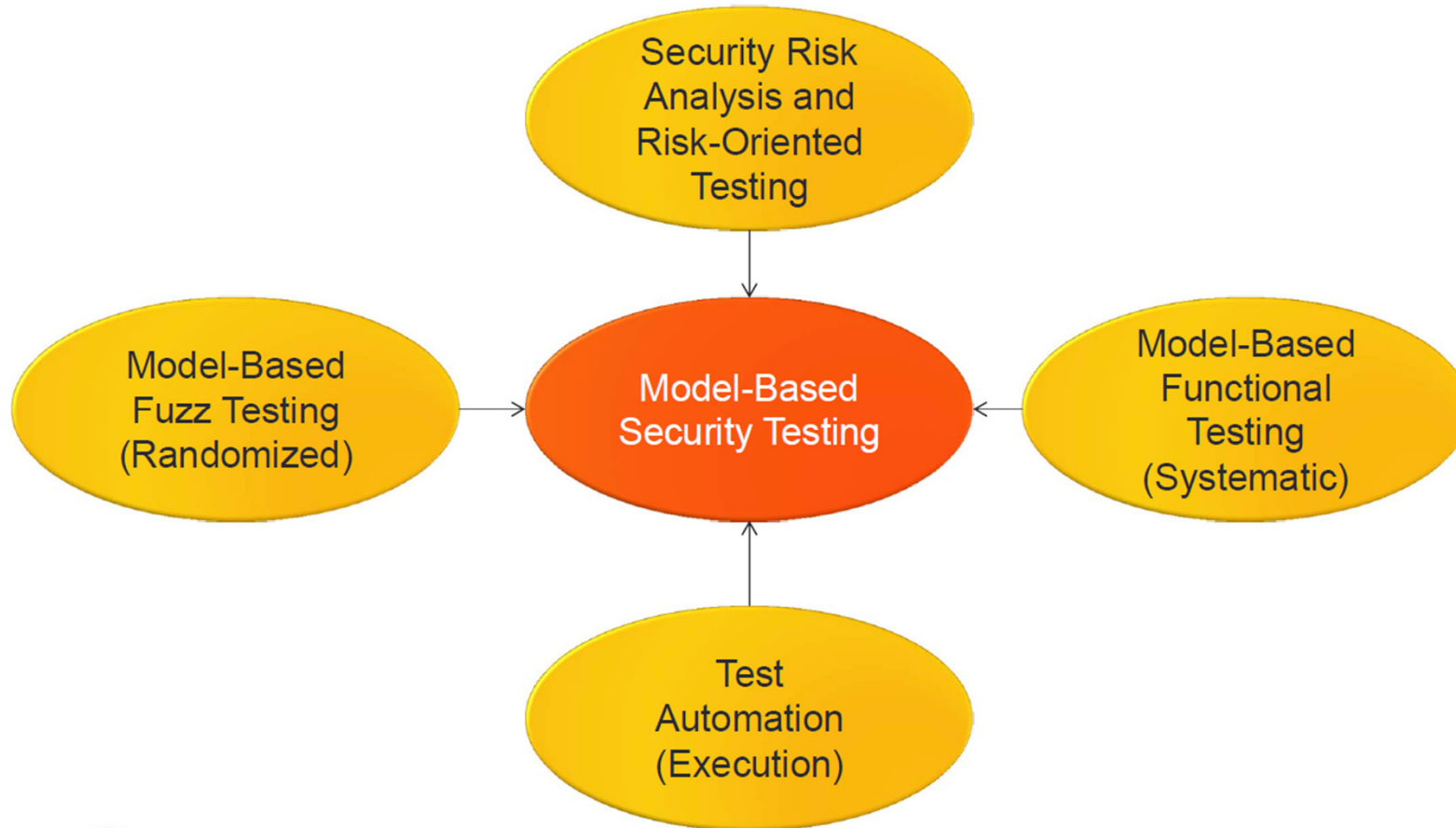
Microsoft Security Intelligence Report, Volume 17 | January through June, 2014

Internet of Things – System Complexity, Heterogeneity and Evolution



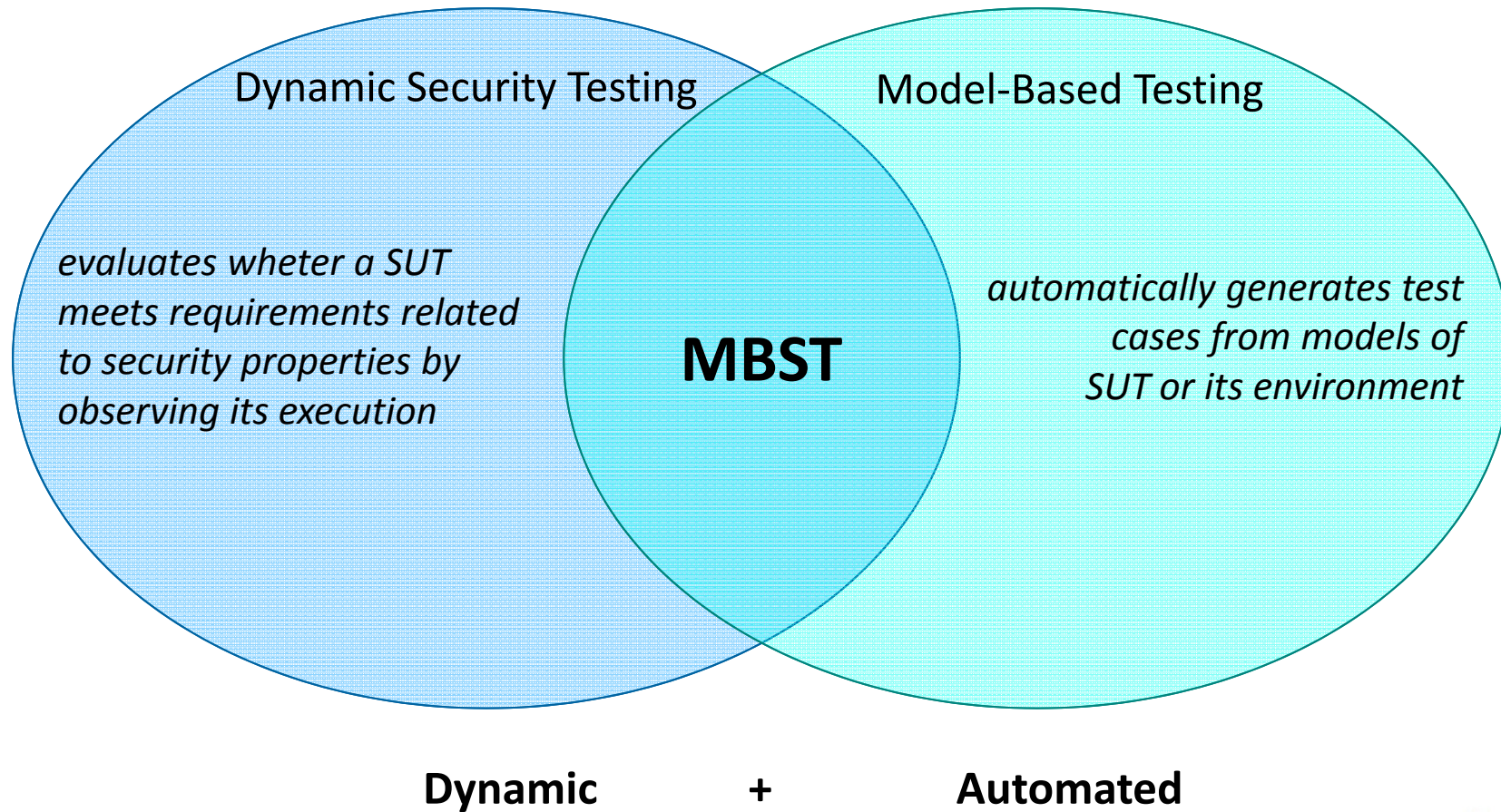
IBM X-Force Threat Intelligence Quarterly, 4Q 2014

MBST @ SECTEST 2012 Keynote

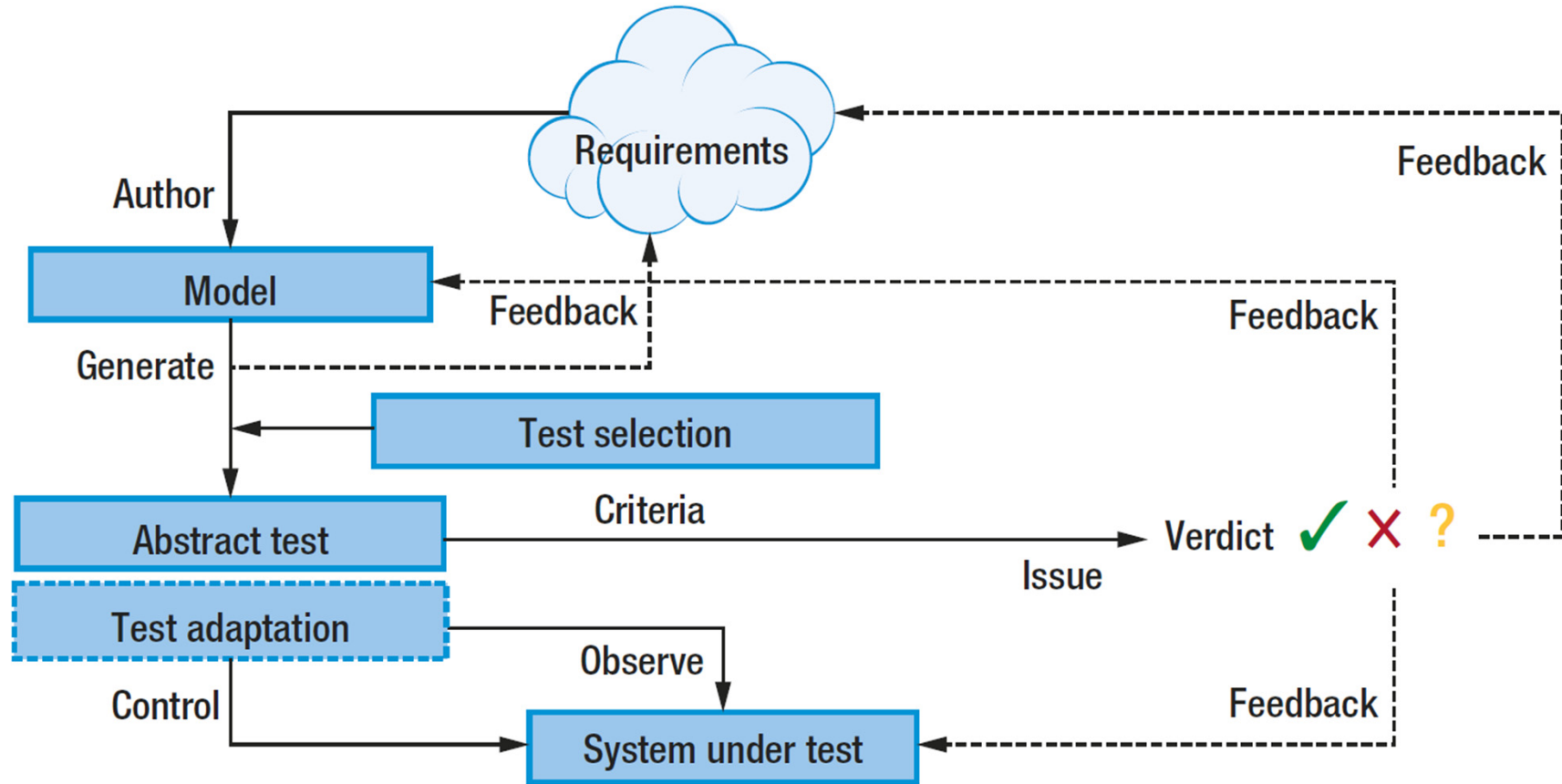


Schieferdecker, I., Großmann, J., Schneider, M. Model-Based Fuzzing for Security Testing, SECTEST 2012

Model-Based Security Testing (MBST)

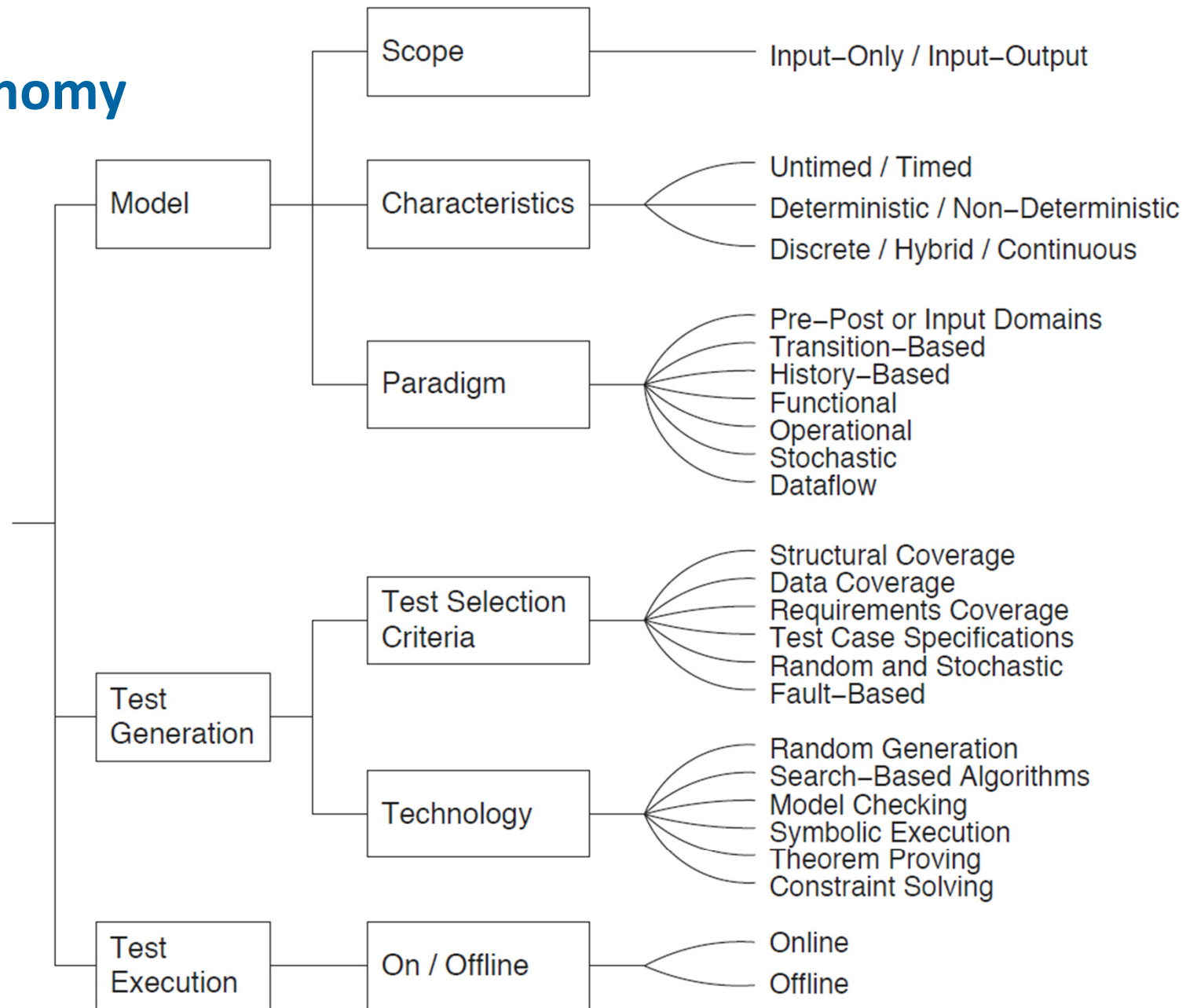


Model-Based Testing (MBT) – Steps and Elements



Schieferdecker, I. (2012). Model-based testing. *IEEE Software*

MBT Taxonomy



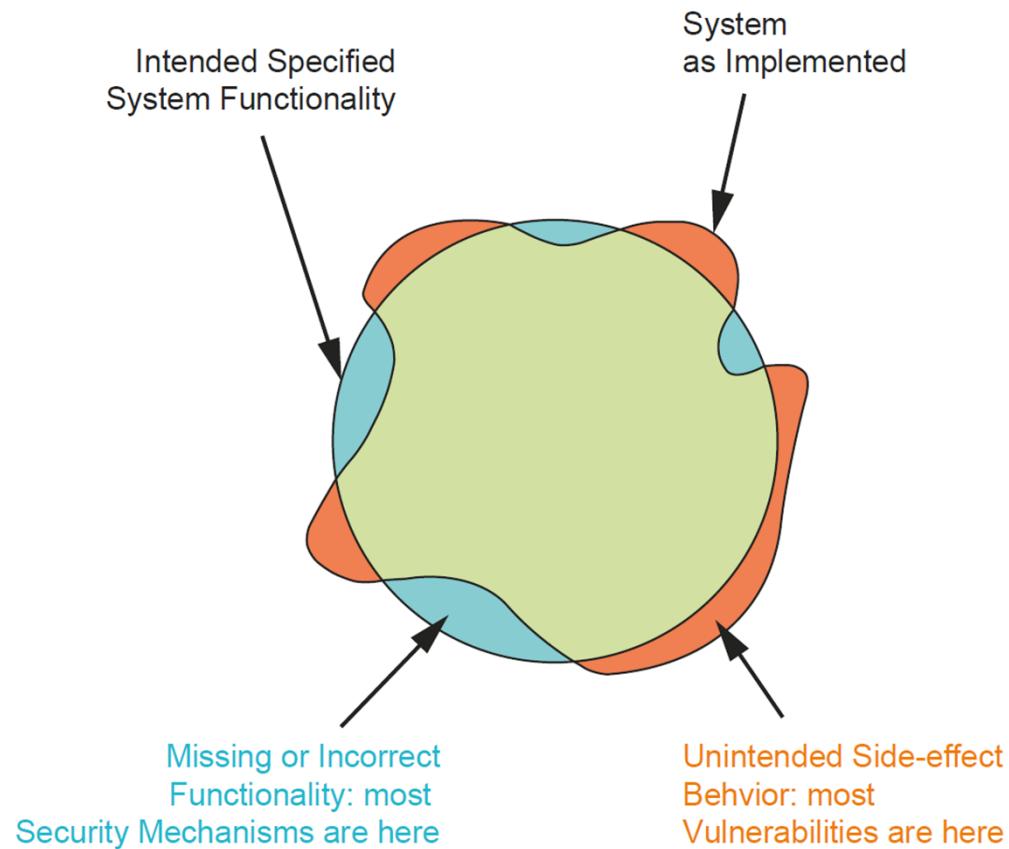
Utting, M. et al. (2012). A taxonomy of model-based testing

Benefits of MBT



- Test models enable
 - **objective** and **systematic** test procedures
 - **knowledge** sustainment
 - **early test specification** and **documentation** fostering **communication**
 - test quality **assessment**
 - test **reuse**, **scalability** and **evolution** fostering **regression testing**
 - **technology-independence** integrating different **levels of abstraction**
 - **automated** test generation and evaluation
- Especially **beneficial for security testing** which tends to be
 - **unstructured**
 - **not reproducible** and **undocumented**
 - **lacking detailed rationales** for test design
 - **dependent on ingenuity** of single testers or hackers

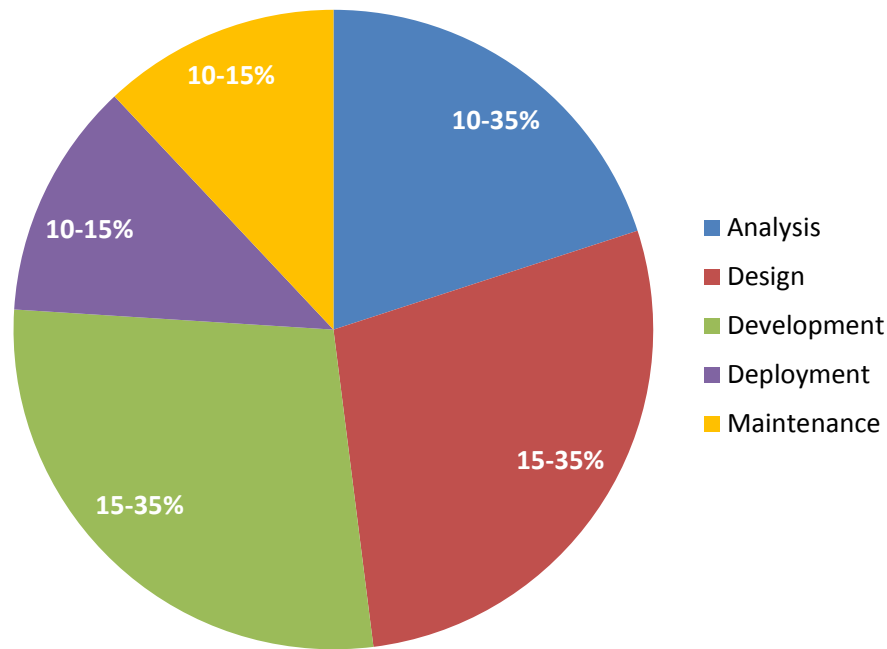
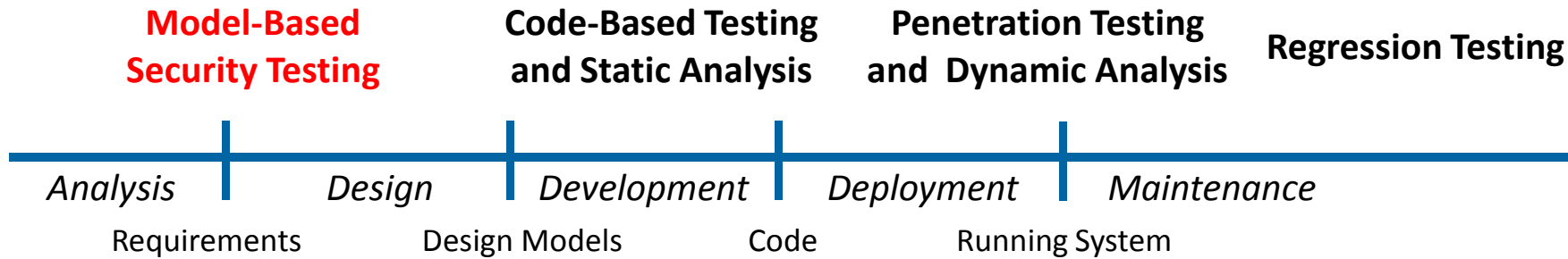
Difficulty of Security Testing and MBT



Based on Thompson, H (2003).
Why security testing is hard. IEEE
Security & Privacy.

- Functional testing focuses on presence of correct behavior not **absence of additional behavior**
- **Non-functional negative requirements** typical for security but hard to model
- Modeling causes initial effort and requires expertise as well as suitable tool support

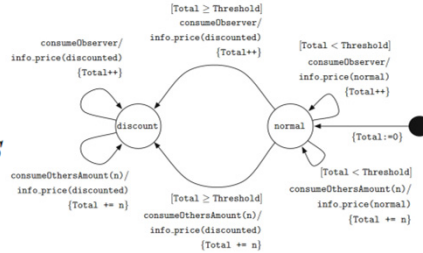
Security Testing Techniques and Model-Based Security Testing



OWASP Foundation. (2013). OWASP Testing Guide v4

Test Model of System Security

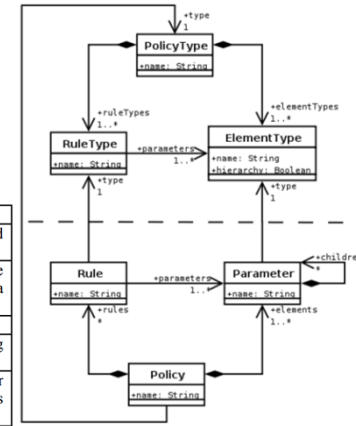
$\text{sent}(A, A, B, M, C) \xrightarrow{\text{intercept}(A,B,M,C)} \text{ik}(M)$
 $\text{sent}(A, A, B, M, C) \xrightarrow{\text{overhear}(A,B,M,C)} \text{ik}(M) \cdot LHS$



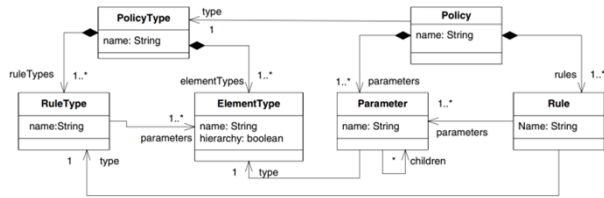
$G \forall (\text{state}_{sp}(7, SP, [C, \dots, URI, \dots]) \Rightarrow \exists O \text{state}_c(2, C, [SP, \dots, URI, \dots]))$

Properties

Operator Name	Definition
RTT	Rule type is replaced with another one
PPR	Replaces one rule parameter with a different one
ANR	Adds a new rule
RER	Removes an existing rule
PPD	Replaces a parameter with one of its descending parameters



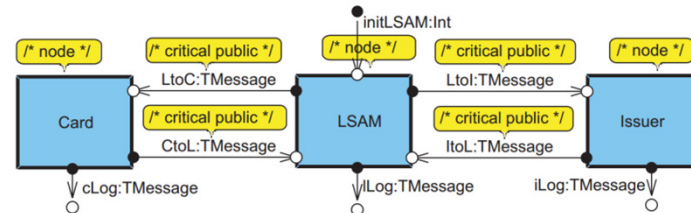
Vulnerability Model



POLICY LibraryOrBAC (OrBAC)

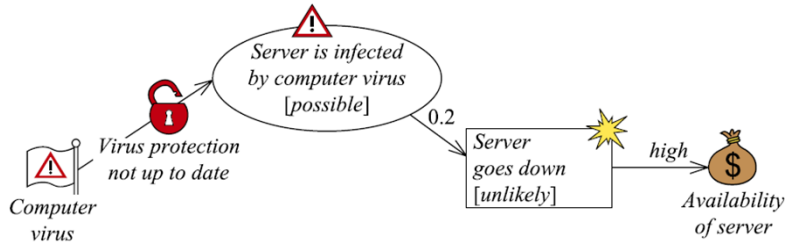
- R1 -> Permission(Library Student Borrow Book WorkingDays)
- R2 -> Prohibition(Library Student Borrow Book Holidays)
- R3 -> Prohibition(Library Secretary Borrow Book Default)
- R4 -> Permission(Library Personnel ModifyAccount UserAccount WorkingDays)
- R5 -> Permission(Library Director CreateAccount UserAccount WorkingDays)

Policy Instance and Meta Model

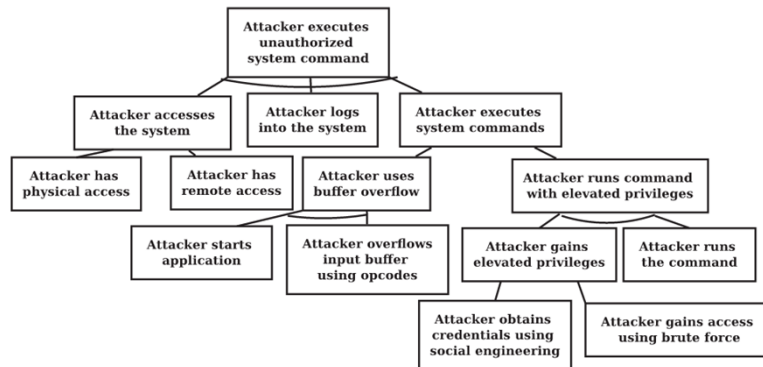


Security Mechanism

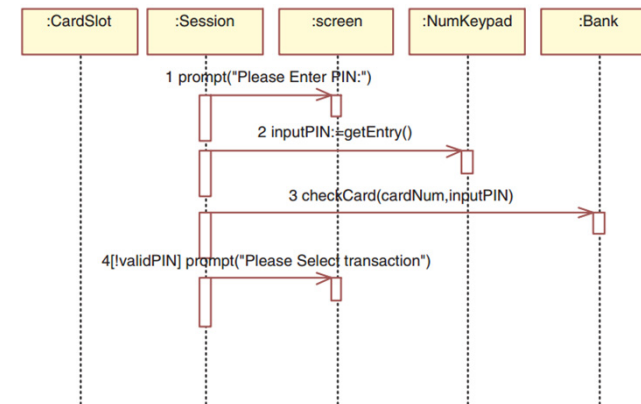
Security Test Models of the Environment



Threat Model + Risk

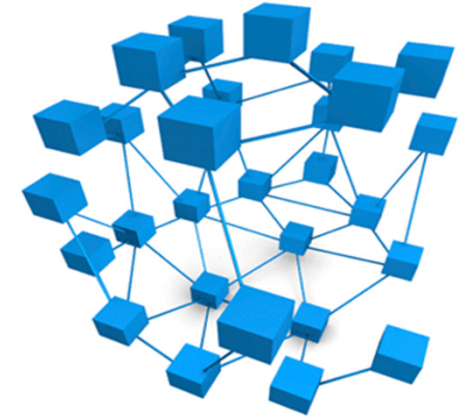


Threat Tree Model



Attack Model

Motivation for MBST Classification



- **MBST is active research area**
 - 3 SECTEST keynotes since 2011 including this one
 - Many approaches available (we counted 119)
- **Framework to understand, categorize, assess, and compare** approaches essential to guide further research and application
 - Clarify key issues, show alternatives and directions for further research
 - Tailoring, selection or integration of approaches
- **Mapping of the field**
- **Requirements**
 - **Based on available classifications** of MBT and security testing
 - **Evaluation and Application**

Main Source

SOFTWARE TESTING, VERIFICATION AND RELIABILITY
Softw. Test. Verif. Reliab. 2014; 00:1-28
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/stvr

Model-Based Security Testing: A Taxonomy and Systematic Classification

Michael Felderer^{1*}, Philipp Zech¹, Ruth Breu¹,
Matthias Büchler², Alexander Pretschner²

¹Institute of Computer Science, University of Innsbruck, Technikerstr. 21a, Innsbruck, Austria
²Software Engineering, TU München, Boltzmannstr. 3, Garching, Germany

SUMMARY

Model-based security testing relies on models to test whether a software system meets its security requirements. It is an active research field of high relevance for industrial applications, with many approaches and notable results published in recent years. This article provides a taxonomy for model-based security testing approaches. It comprises filter criteria (model of system security, security model of the environment, and explicit test selection criteria) as well as evidence criteria (maturity of evaluated system, evidence measures, and evidence level). The taxonomy is based on a comprehensive analysis of existing classification schemes for model-based testing and security testing. To demonstrate its adequacy, 119 publications on model-based security testing are systematically extracted from the literature and classified according to the defined filter and evidence criteria. The state of the art of model-based security testing as well as directions of future research are discussed. Copyright © 2014 John Wiley & Sons, Ltd.

Received ...

KEY WORDS: model-based security testing; security testing; model-based testing; classification; taxonomy

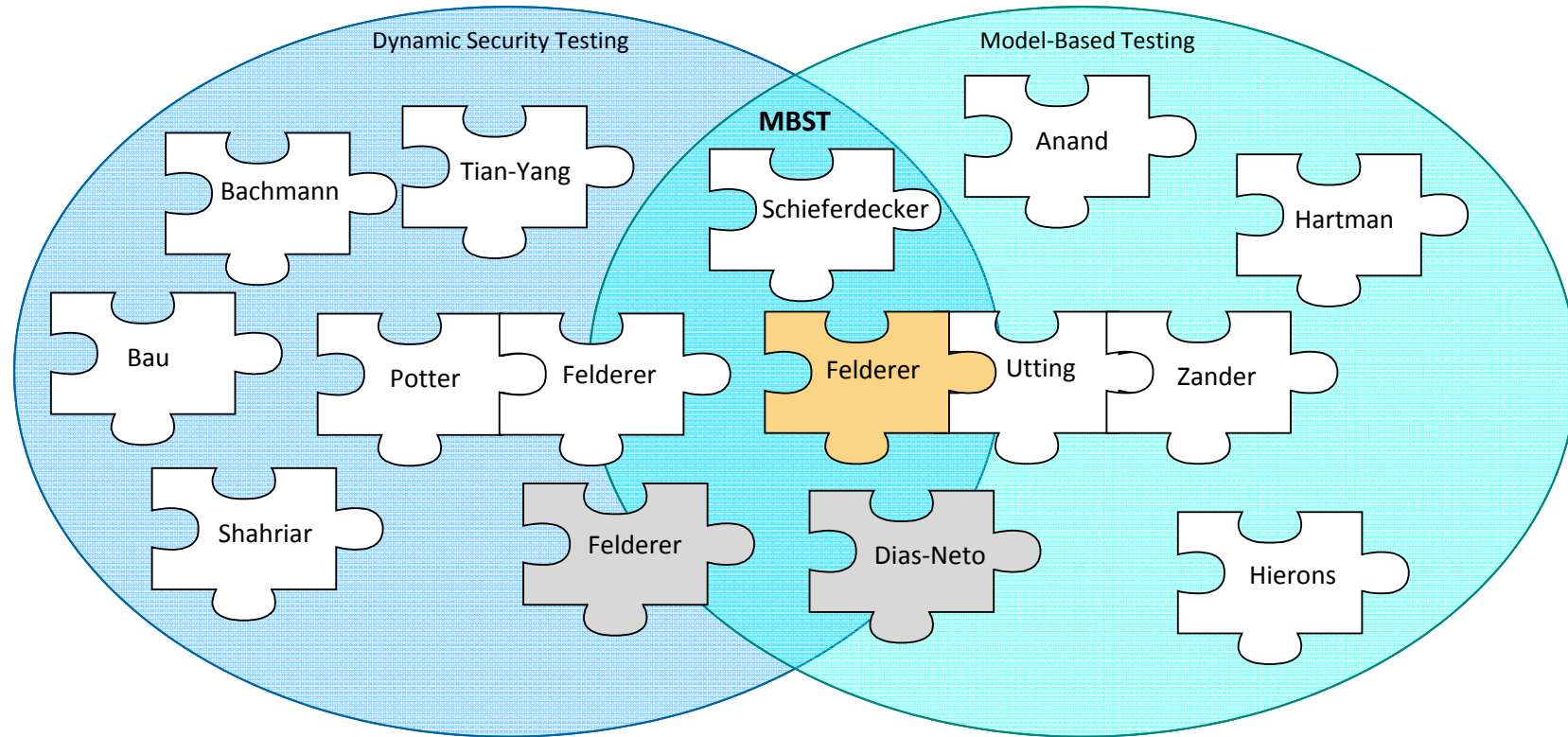
1. INTRODUCTION

Modern IT systems based on concepts like cloud computing, location-based services or social networking are permanently connected to other systems and handle sensitive data. These interconnected systems are subject to security attacks that may result in security incidents with high severity affecting the technical infrastructure or its environment. Exploited security vulnerabilities can cause drastic costs, e.g., due to downtimes or the modification of data. A high proportion of all software security incidents is caused by attackers who exploit known vulnerabilities [1]. An important, effective and widely applied measure to improve the security of software are security testing techniques which identify vulnerabilities and ensure security functionality.

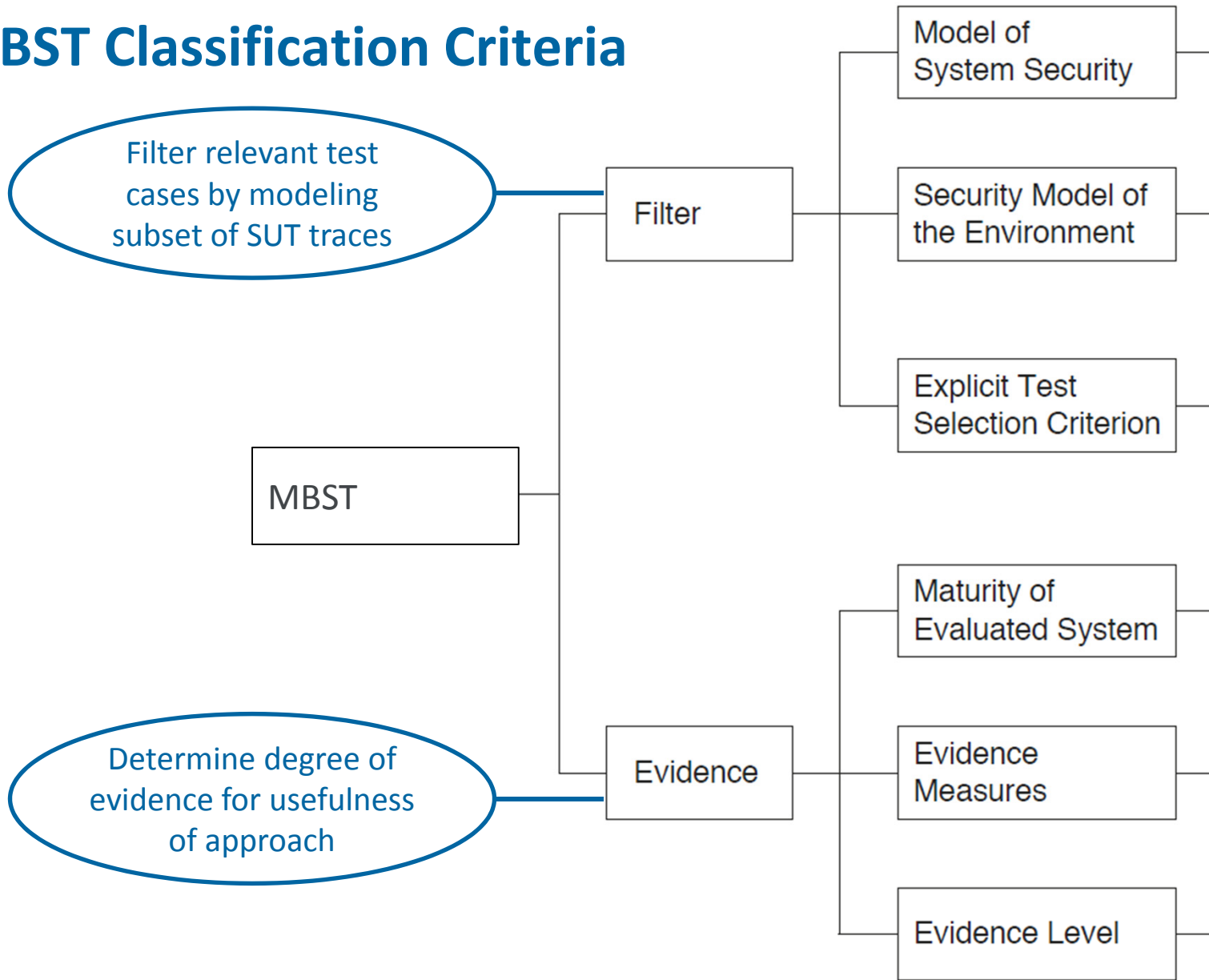
In this article, testing refers to active, dynamic testing, where the behavior of a system under test (SUT) is checked by applying intrusive tests that stimulate the system and observe and evaluate the system reactions [1]. Security testing validates software system requirements related to security properties like confidentiality, integrity, availability, authentication, authorization and non-repudiation. Sometimes security properties come as classical functional requirements, e.g., 'user accounts are disabled after three unsuccessful login attempts' which approximates one part of an

*Correspondence to: Institute of Computer Science, University of Innsbruck, Technikerstr. 21a, 6020 Innsbruck, Austria.
Email: michael.felderer@uibk.ac.at

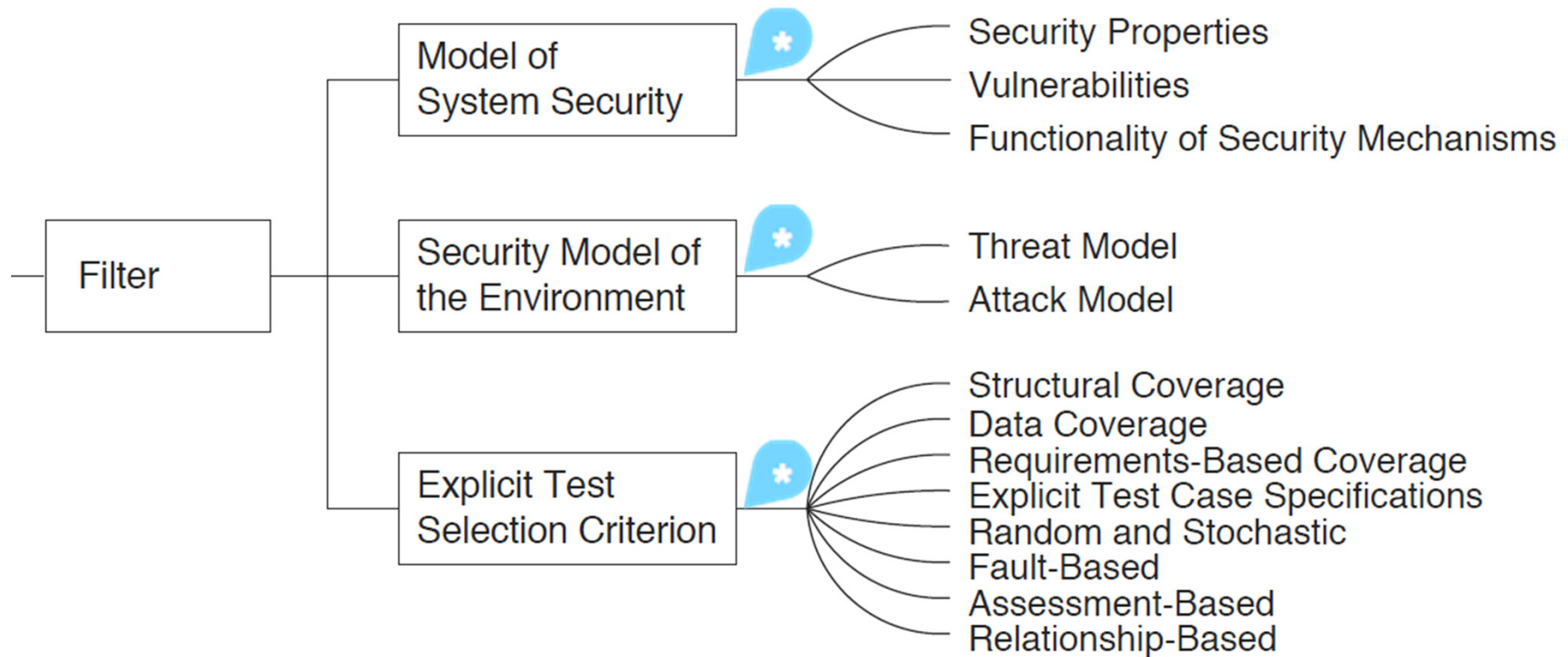
Classifications of Security Testing and Model-Based Testing




MBST Classification Criteria



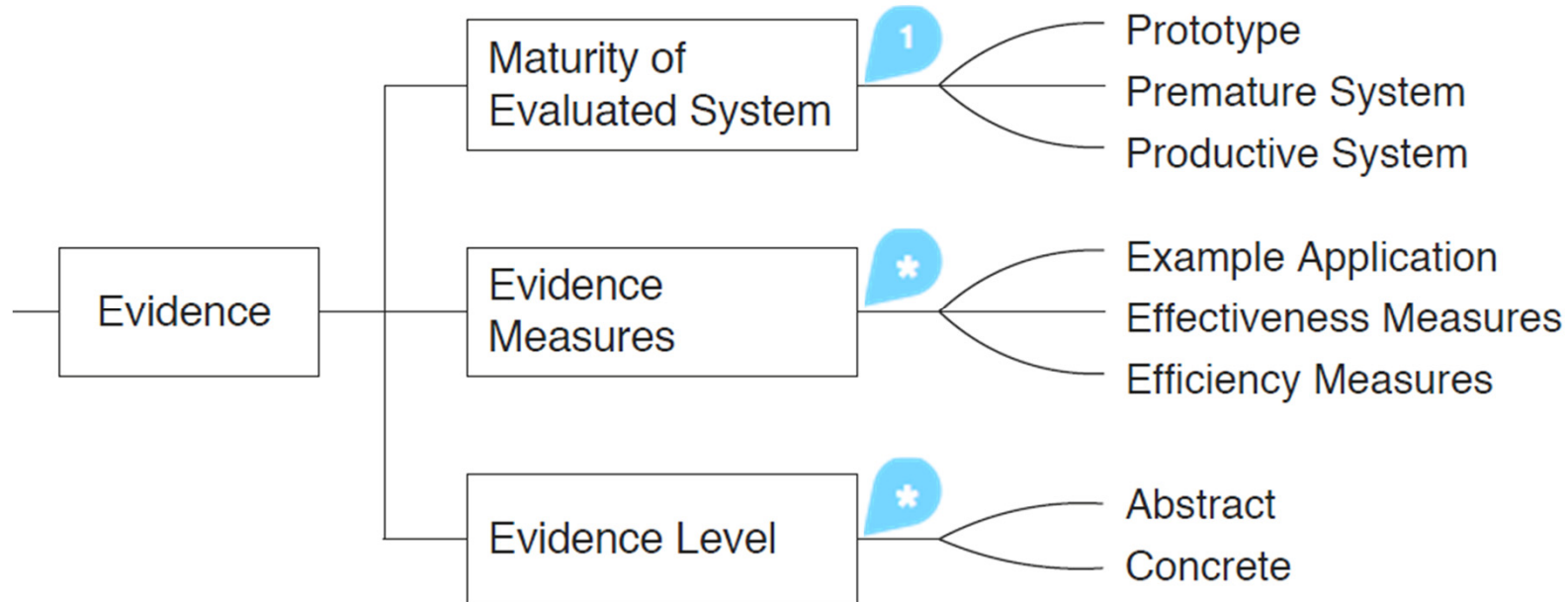
MBST Classification - Filter Criteria and Values



 multi-select

 single-select

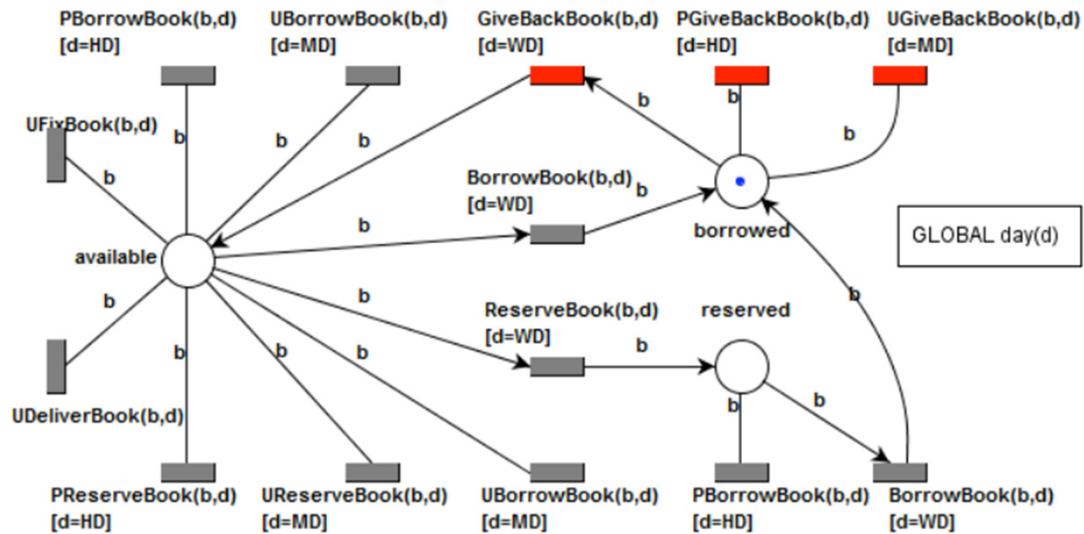
MBST Classification - Evidence Criteria and Values



multi-select

single-select

Example Classification – Xu et al.



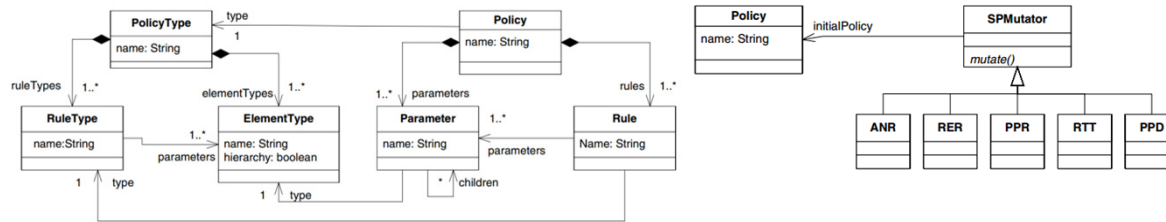
```

public void test12() throws exception {
    System.out.println("Test 12");
    ContextManager.currentContext=ContextManager.workingday;
    doPermittedReserve(Book1Title);
    assertTrue(isBookReserved(Book1Title));
    ContextManager.currentContext = ContextManager.workingday;
    doPermittedBorrow(Book1Title);
    assertTrue(isBookBorrowed(Book1Title));
    ContextManager.currentContext = ContextManager.holiday;
    doProhibitedGiveBack(Book1Title);
    assertTrue(isBookBorrowed(Book1Title));
}
    
```

Criterion	Value
MoSS	Security Mechanism
SModE	-
TSC	Structural Coverage
MoES	Prototype
EM	Effectiveness
EL	Executable

Xu, D. et al. (2012). A model-based approach to automated testing of access control policies. SACMAT 2012

Example Classification – Mouelhi et al.



POLICY LibraryOrBAC (OrBAC)

- R1 -> Permission(Library Student Borrow Book WorkingDays)
- R2 -> Prohibition(Library Student Borrow Book Holidays)
- R3 -> Prohibition(Library Secretary Borrow Book Default)
- R4 -> Permission(Library Personnel ModifyAccount UserAccount WorkingDays)
- R5 -> Permission(Library Director CreateAccount UserAccount WorkingDays)

Operator Name	Definition
RTT	Rule type is replaced with another one
PPR	Replaces one rule parameter with a different one
ANR	Adds a new rule
RER	Removes an existing rule
PPD	Replaces a parameter with one of its descending parameters

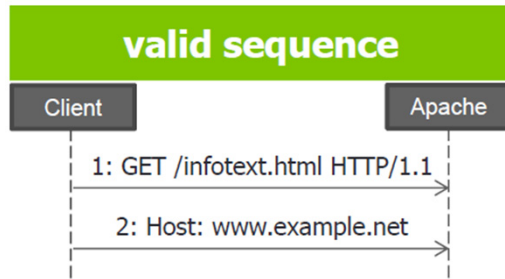
```
public aspect PEPAspect {
    // PEP Joinpoint for borrow
    before(User user, Book book) throws
    SecuritPolicyViolationException :
    borrowBookCall(user, book) {

        // Call to check for security rule
        checkSecurity(user.getRole(), BORROWMETHOD
        , BOOKVIEW, getTemporalContext());
    }
}
```

Criterion	Value
MoSS	Security Mechanism + Vulnerabilities
SMoE	-
TSC	Structural Coverage + Fault Based
MoES	Prototype
EM	Effectiveness
EL	Executable

Mouelhi, T. et al. (2008). A model-based framework for security policy specification, deployment and testing. MoDELS 2008

Example Classification – Schneider et al.



Repeat
Message
2: Host



Criterion	Value
MoSS	-
SModE	Attack
TSC	Fault Based
MoES	Prototype
EM	Example
EL	Executable

Schneider, M. et al. (2013). Online Model-Based Behavioral Fuzzing. ICSTW 2013

Systematic Selection and Classification of MBST Publications

1. Selection of Publications

- 1.1 Search Strategy
- 1.2 Stepwise Paper Selection

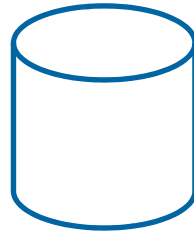
2. Paper Classification

- 2.1 Bibliographic Information (Title, Year, Publisher)
- 2.2 Filter Criteria and Evidence Criteria
- 2.3 Comment field for remarks resolved by all three classifying researchers

3. Threats to Validity

- 3.1 Publication Bias
- 3.2 Threats to the Identification of Publications
- 3.3 Threats to Classification of Publications

Search Strategy



76 papers in reference DB



recall: 100%

("model based" OR automata OR "state machine" OR
"specification based" OR policy OR policies OR
"threat model" OR mutation OR risk OR fuzzing)
AND (security OR vulnerability OR privacy OR cryptographic)
AND (test OR testing)



ACM DL DIGITAL LIBRARY

Enter words, phrases or names below. Surround phrases or full names with double quotation marks.

SEARCH

Words or Phrases

Find [any field] with

all of this text (and)

any of this text (or)

none of this text (not)

Names

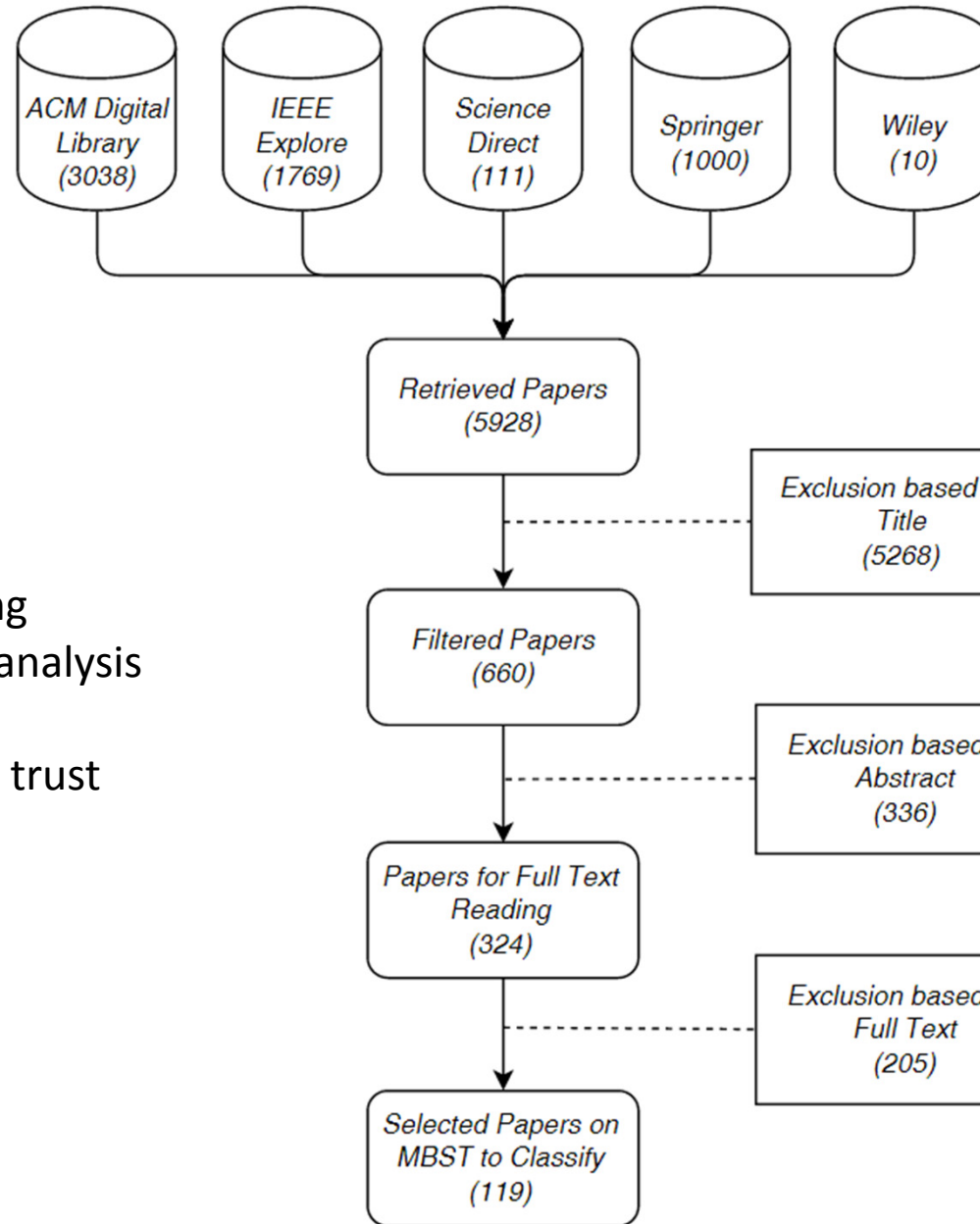
Find [any field] with

names

using all any none of the names

1996 <= year <= 2013

Paper Selection



Selection Criteria:

- Explicit models
- Dynamic, active testing
no monitoring, static analysis
- Only security
no robustness, safety, trust

Current State of MBST – Security Test Models

	SecP	V	FSecM	SecP+V	SecP+FSecM	V+FSecM	n.s.	Sum
T	1	0	3	0	0	0	2	6
A	2	0	1	0	0	0	20	23
T+A	1	1	0	0	1	0	3	6
n.s.	15	7	42	2	9	9	0	84
Sum	19	8	46	2	10	9	25	119

T – Threat, A – Attack, SecP – Security Properties, V – Vulnerabilities, FSecM – Functionality Security Mechanisms

- **All 119 publications can be classified**
- **Models of system security** are much more frequent
 - 84 papers exclusive MoSS, 25 exclusive SMOE, 10 combined
- **FSecM** are most chosen type of models (e.g., access control models)
- MBST based on **vulnerabilities** is not common
- 35 papers consider SMOE, 29 of these chose **Attack Models**

Current State of MBST – Test Selection Criteria

	SecP	V	FSecM	T	A	Sum
SC	13	7	38	7	16	81
DC	0	1	10	2	1	14
RC	3	0	4	2	0	9
TCS	9	1	9	1	6	26
RS	1	1	5	0	0	7
FB	6	9	12	3	5	35
AB	0	1	0	0	3	4
RB	2	2	2	0	1	7

SC – Structural Coverage, DC – Data Coverage, RC – Requirements Coverage, TCS – Explicit Test Case Specifications
RS – Random and Stochastic, FB – Fault-Based, AB – Assessment-Based, RB – Relationship-Based
T – Threat, A – Attack, SecP – Security Properties, V – Vulnerabilities, FSecM – Functionality Security Mechanisms

- **Structural Coverage** predominant
- **Fault-Based** and **Explicit Test Case Specification** are present
- **Other criteria** still rarely applied (e.g., DC, RS, AB, RB)

Current State of MBST – Evidence

	Abs	Exec	Abs + Exec	Sum
Prototype				
Ex	36	22	23	81
Effe	3	15	7	25
Effi	3	9	7	19
Sum	42	46	37	125
Premature				
Ex	1	0	1	2
Effe	0	2	1	3
Sum	1	2	2	5
Production				
Ex	3	3	1	7
Effe	2	7	0	9
Effi	1	0	0	1
Sum	6	10	1	17

Ex – Example, Effe – Effectiveness, Effi – Efficiency, Abs – Abstract, Exec - Executable

- **Example application on prototypes** most frequent
- **Efficiency** has rarely been investigated
- Approaches still rarely applied in **practice**

Current State of MBST – Security Test Models vs.

	Prot	Pre	Prod	Ex	Effe	Effi	Abs	Exec	Abs+Exec
SecP	27	3	1	25	5	5	17	5	9
V	17	0	2	13	9	4	4	11	4
FSecM	56	3	6	49	19	13	24	24	16
T	10	1	1	11	3	3	5	4	3
A	21	2	5	25	4	2	10	10	8
Sum	131	9	15	123	40	27	60	54	40

T – Threat, A – Attack, SecP – Security Properties, V – Vulnerabilities, FSecM – Functionality Security Mechanisms
 Prot – Prototype, Pre – Premature, Prod – Productive, Ex – Example, Effe – Effectiveness, Effi – Efficiency, Abs – Abstract, Exec - Executable

- **Example application based on prototypes** very common for all models
- **Attacks** are relatively frequently applied in **productive systems**, SecP not
- **Effectiveness and Efficiency** for Vulnerabilities and FSecM relatively common
- Evaluation on **abstract and executable level equally common**, except for Security Properties abstract and for vulnerability executable prevailing

Directions and Challenges – Security Test Models

- **Vulnerability models** are underrepresented
 - Vulnerability models are hard to design, compared to classical fault-based software testing models but in practice one often starts with hypotheses about vulnerabilities
- More research devoted to testing functionality of security mechanisms than **security properties** as such
 - Security mechanism does not always implement a system-wide security property such as confidentiality
 - Relationship between local and system-wide mechanisms relevant
- Cross-fertilization with **safety and reliability community** could be beneficial

Current Related Publications on Fault Models

FEATURE: SOFTWARE TESTING

Using Defect Taxonomies for Testing Requirements

Michael Felderer, University of Innsbruck
Armin Beer, Beer Test Consulting

// The proposed requirements-based-testing approach seamlessly integrates defect taxonomies into a standard test process and improves the effectiveness and efficiency of system testing. The researchers successfully applied this approach to industrial projects at a public health insurance institution. //

SYSTEMATIC DEFECT management based on bug-tracking systems such as Bugzilla¹ is well established and has been successfully used in many software organizations. Defect management weighs the failures observed during test execution according to their severity and forms the basis for effective defect taxonomies. In practice, most defect taxonomies are used only for the a posteriori allocation of testing resources to prioritize failures for debugging. Thus, these taxonomies' full potential to control and improve all the steps of testing has remained unexploited. This is especially the case for testing a system's user requirements. System-level defect taxonomies can improve the design of requirements-

based tests, the tracing of defects to requirements, the quality assessment of requirements, and the control of the relevant defect management. So, we developed *requirements-based testing with defect taxonomies* (RTDT). This approach is aligned with the standard test process and uses defect taxonomies to support all phases of testing requirements. To illustrate this approach and its benefits, we use an example project (which we call Project A) from a public health insurance institution.

Project A
In this institution, all projects require an iterative and incremental development process and a standard test process based on the International Software Testing Qualifications Board's test process.² Project A developed a Web application to help employees care for handicapped people and manage the related cases. Its development took approximately nine months, four iterations, and a project staff of about seven. The application uses a Web browser as a client to display the GUI and a server for data management and program control. The architecture is service-oriented, various applications support the different business processes, and the users have role-specific access to those applications.

Requirements-Based Testing with Defect Taxonomies
Requirements-based testing (RT) dynamically validates whether a system fulfills its specification.³ The standard process has four phases: test planning, test design, test execution, and test evaluation. RTDT adds four more phases: requirements specification, choosing RT or RTDT, creating

2 IEEE SOFTWARE | PUBLISHED BY THE IEEE COMPUTER SOCIETY 0740-7459/15/0311-00 © 2015 IEEE

Felderer, M. & Beer, A. (2015). Using Defect Taxonomies for Testing Requirements. IEEE Software

A Generic Fault Model for Quality Assurance

Alexander Pretschner¹, Dominik Holling¹,
Robert Eschbach², and Matthias Gemmar²

¹ Technische Universität München, Germany
{pretchn,holling}@in.tum.de

² itk Engineering, Germany
{robert.eschbach,matthias.gemmar}@itk-engineering.de

Abstract. Because they are comparatively easy to implement, structural coverage criteria are commonly used for test derivation in model- and code-based testing. However, there is a lack of compelling evidence that they are useful for finding faults, specifically so when compared to random testing. This paper challenges the idea of using coverage criteria for test selection and instead proposes an approach based on fault models. We define a general fault model as a transformation from correct to incorrect programs and/or a partition of the input data space. Thereby, we leverage the idea of fault injection for test assessment to test derivation.

We instantiate the developed general fault model to describe existing fault models. We also show by example how to derive test cases.

1 Introduction

Partition-based testing [23] relies on the idea of partitioning the input domain of a program into blocks. For testing, a specified number of input values is usually drawn randomly from each block. The number of tests per block can be identical, or can vary according to a usage profile. Sometimes, the blocks of the partition are considered to be "equivalence classes" in an intuitive sense, namely in that they either execute the same functionality, or are likely to provoke related failures. Code coverage criteria, including statement, branch and various forms of condition coverage, naturally induce a partition of the input domain: in a control flow graph, every path from the entry to the exit node (or back to the entry node) of a program represents all those input data values that, when applied to the program, lead to the respective path being executed. Since this same argument also applies to different forms of condition coverage, coverage-based testing can be seen as an instance of partition-based testing.

More than twenty years ago, Weyuker and Jeng have looked into the nature of test selection based on input domain partitions [25]. They contrasted partition-based testing to random testing; more specifically, to test selection that uniformly samples input values from a program's input domain. To keep the model simple, their criterion to contrast these two forms of testing measures the probability of detecting at least one failure.

They show that depending on how the failure-causing inputs are distributed across the input domain, partition-based testing can be better, the same, or worse

A. Moreira et al. (Eds.): MODELS 2013, LNCS 8107, pp. 87–103, 2013.
© Springer-Verlag Berlin Heidelberg 2013

Pretschner, A. et al. (2015). A Generic Fault Model for Quality Assurance. MODELS 2013

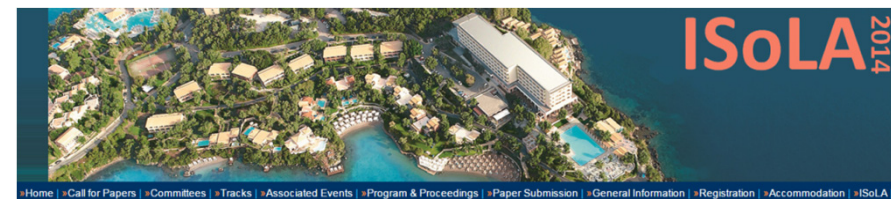
Directions and Challenges – Test Selection



- **Coverage criteria** are very popular, but not clear whether these test cases are effective because relationship to fault distribution in software is normally missing
 - Investigate which coverage criteria are effective (and efficient)
- **Test selection criteria specifically relevant** for MBST are **underrepresented** in actual research
 - Data Coverage
 - Random and Stochastic
 - Risk-based
 - Regression
- Risk-based testing has to be seen in **process context**

Risk-Based Testing Activities

- Workshop series on Risk Assessment and Risk-driven Testing (**RISK**)
- Special Track on RBT at 6th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (**ISoLA 2014**)
- Special Section on RBT in International Journal on Software Tools for Technology Transfer, **STTT 16(5)**, 2014



Directions and Challenges - Evidence



- **Evaluation of Effectiveness and Efficiency** underrepresented
- **Empirical Body of Knowledge** on MBST
 - Return on investment of MBST approaches
 - In which context and how can a specific approach be applied or not?
 - Comprises organizational, process, tool and artifact aspects
 - Decoupling from security experts and increase of applicability
 - Creation of accepted and well-founded theories and research issues
- **Refinement of classification**
 - Domain, System Type, Vulnerabilities

Presentation on Efficiency of Model-Based Testing

A Case Study on the Efficiency of Model-Based Testing at the European Space Agency

Stefan Mohacsi
ATOS IT Solutions and Services
Vienna, Austria
stefan.mohacsi@atos.net

Michael Felderer
University of Innsbruck
Innsbruck, Austria
michael.felderer@uibk.ac.at

Armin Beer
BVA & Beer Test Consulting
Vienna, Austria
armin.beer@bva.at

Abstract— In this paper we present the results of an empirical case study performed at the European Space Agency (ESA). In this major project, the various challenges for testing were tackled using a model-based approach for test design and the generation of executable test automation scripts. An evaluation of this approach's efficiency identified significant cost savings and quality improvements.

Index Terms— Model-based testing; test automation; software testing; industrial case study.

I. INDUSTRIAL CONTEXT

The European Space Agency's Multi-Mission User Services (MMUS) infrastructure provides services for earth observation products such as satellite images. Supported services include product catalog search and ordering, mission planning, online information, and documentation services.

II. MODEL-BASED TESTING AT ESA

A. Model-based Test Framework

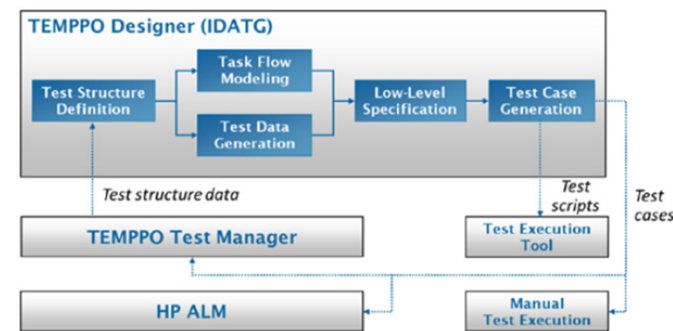


Fig. 1. MBT Framework

Thursday, April 16th, 13:30 – 15:00, Session Testing in Practice 3, ICST 2015

Referenced Papers

- [1] Felderer, M. & Beer, A. (2015). Using Defect Taxonomies for Testing Requirements. IEEE Software (online first)
- [2] Mouelhi, T. et al. (2008). A model-based framework for security policy specification, deployment and testing. MoDELS 2008
- [3] Pretschner, A. et al. (2013). A Generic Fault Model for Quality Assurance. MoDELS 2013
- [4] Schieferdecker, I. (2012). Model-based testing. IEEE Software, 29(1), 14-18
- [5] Schneider, M. et al. (2013). Online Model-Based Behavioral Fuzzing. ICSTW 2013
- [6] Thompson, H. (2003). Why security testing is hard. IEEE Security & Privacy, 1(4), 83-86
- [7] Utting, M. et al. (2012). A taxonomy of model-based testing. Software Testing, Verification and Reliability, 22(2), 297-312
- [8] Xu, D. et al. (2012). A model-based approach to automated testing of access control policies. SACMAT 2012



michael.felderer@uibk.ac.at