Model-based Testing Theory and Application

Franz Wotawa
Technische Universität Graz
Institute for Software Technology
8010 Graz, Inffeldgasse 16b/2, Austria
wotawa@ist.tugraz.at

Point of departure

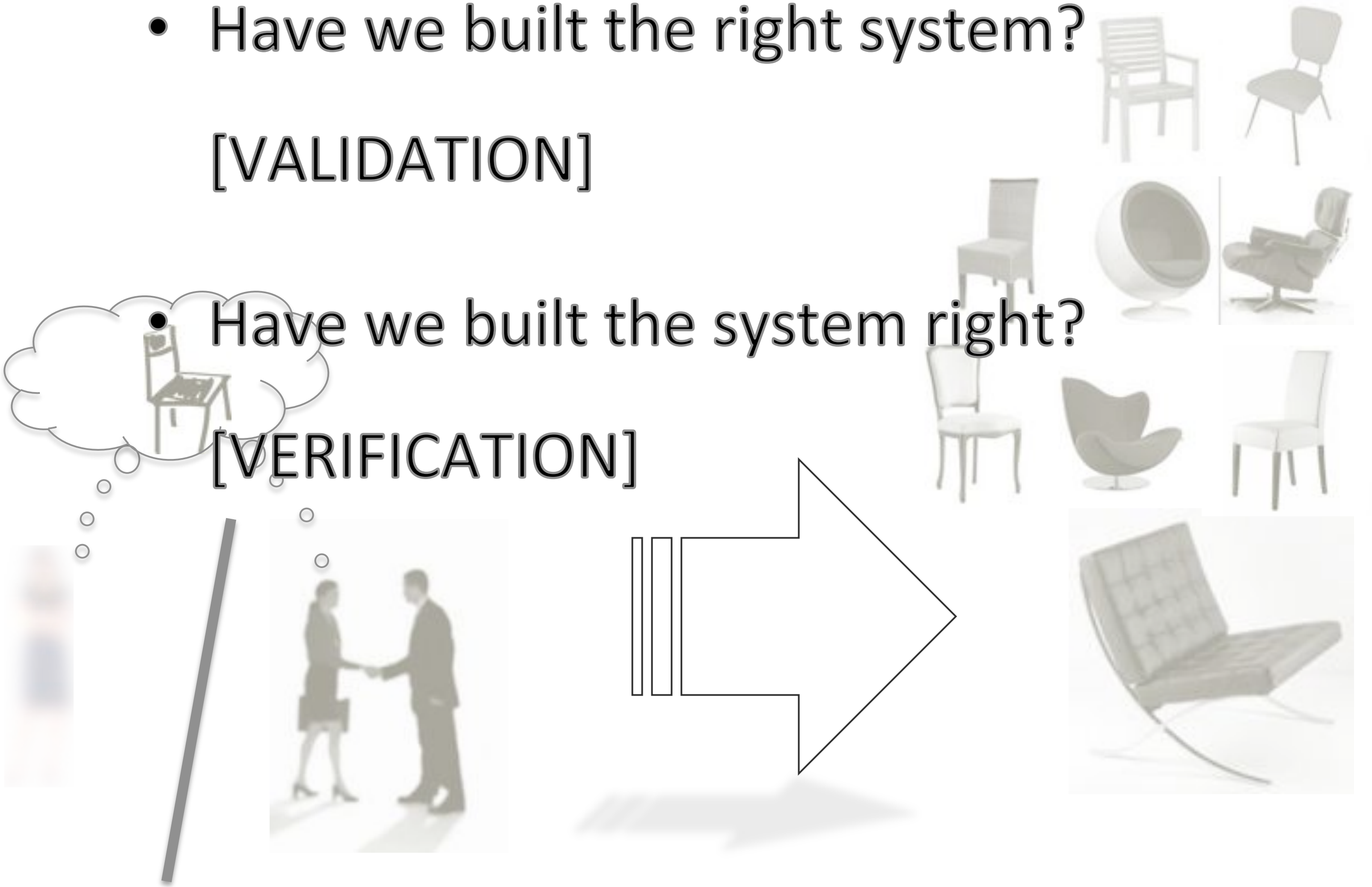
Open questions

- Have we built the right system?

  [VALIDATION]

- Have we built the system right?

  [VERIFICATION]



V&V activities

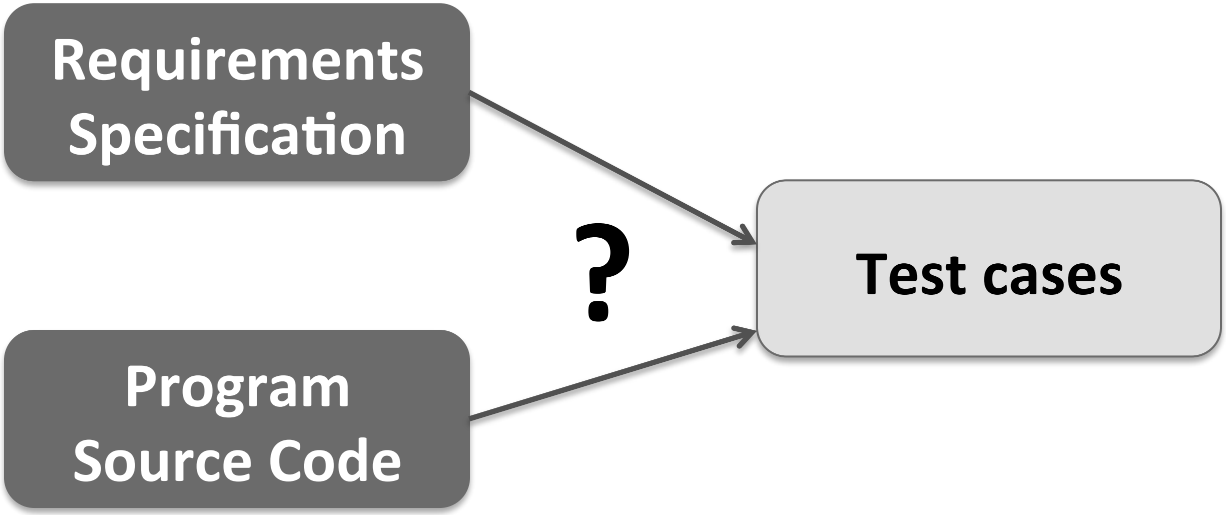
- Check correctness and expectations!
  - Formal *verification*
  - **Testing**
    (for verification and validation)

Testing framework

- Requirements
- Specification
- Source code
- Tests are characterized by:
  - Input values
  - Expected output values
- Test suite = set of tests
- Program is “correct” iff all tests are fulfilled!

Test characterization

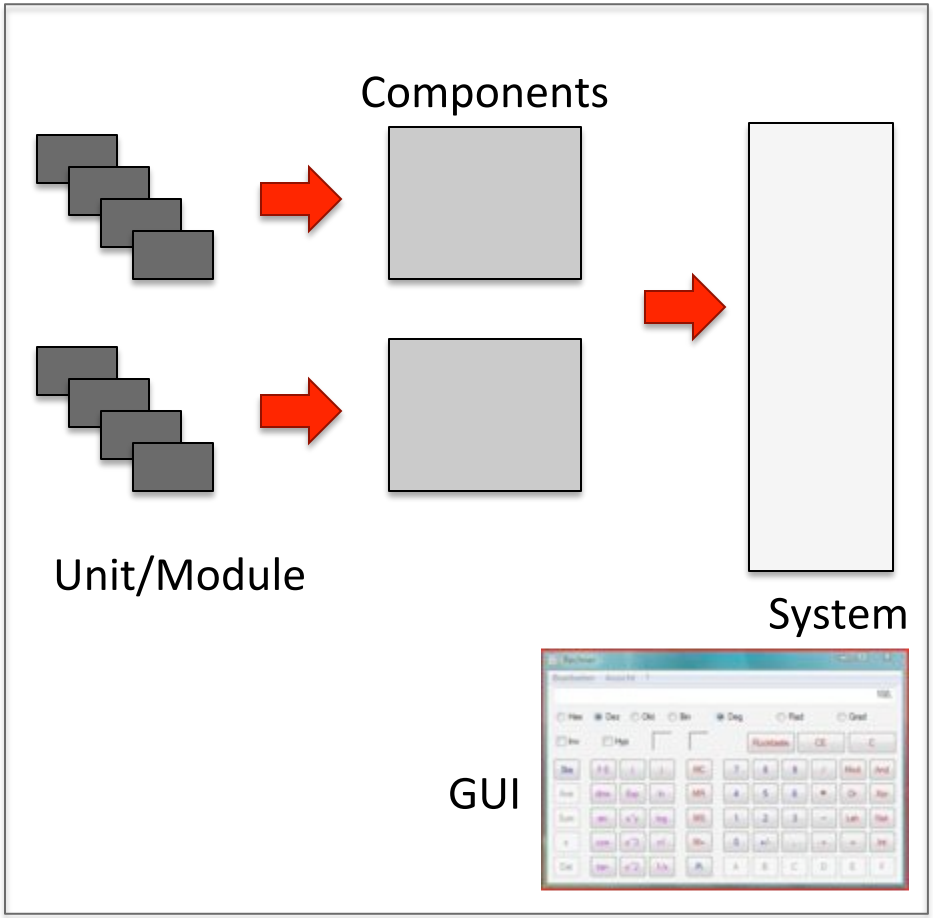
- *Which information is available?*
  - **Black-box** vs. **White-box** testing



  - **Examples**: Model-based testing, Equivalence-based methods, Combinatorial testing, Coverage-based methods, Random-Testing (Monkey Testing, Fuzz Testing,..)

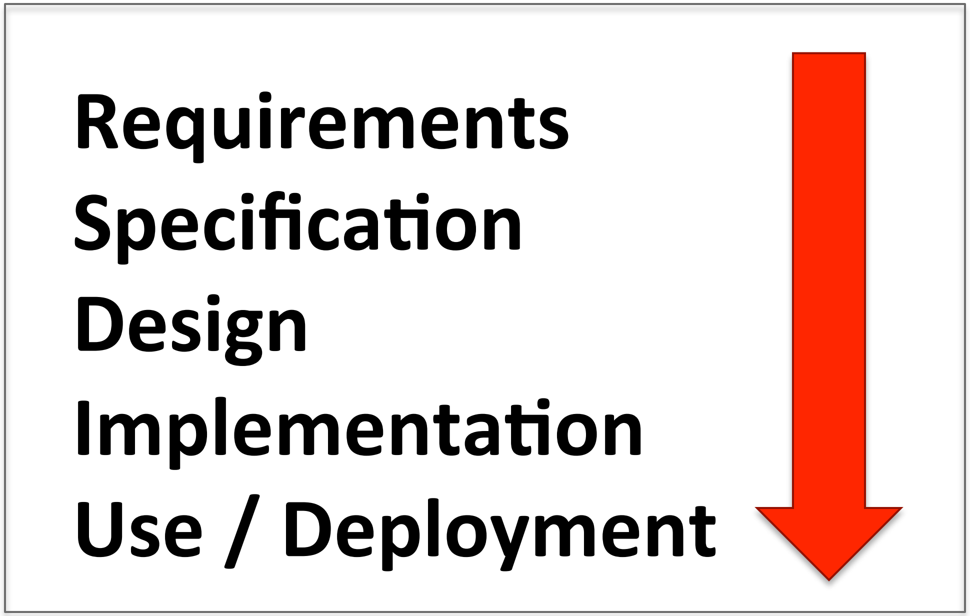
Testing – a program-centric view

- Which part of the program to be tested?
  - Unit-Tests
  - Component tests
  - Integration tests
  - System tests
  - User-interface Testing



Testing – a process-oriented view

- At which part of the development process testing is done?
  - Verification (Unit-Tests, regressions tests,…)
  - Validation



Requirements
Specification
Design
Implementation
Use / Deployment

What should I test?

- Functionality
- Robustness
- Usability

100,

○ Hex ● Dez ○ Okt ○ Bin ● Deg ○ Rad ○ Grad

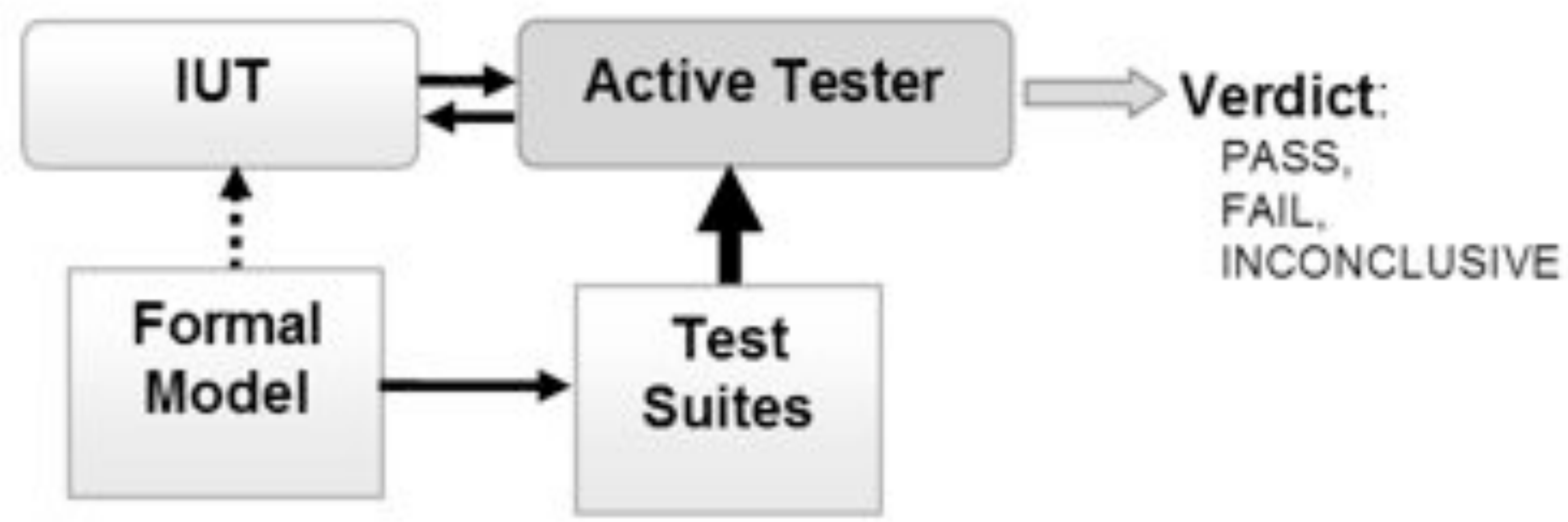
□ Inv □ Hyp

Rechner
Bearbeiten Ansicht ?
Rücktaste CE C
Sta F-E ( ) MC 7 8 9 / Mod And
Ave dms Exp ln MR 4 5 6 * Or Xor
Sum sin x^y log MS 1 2 3 - Lsh Not
s cos x^3 n! M+ 0 +/- , + = Int
Dat tan x^2 1/x Pi A B C D E F

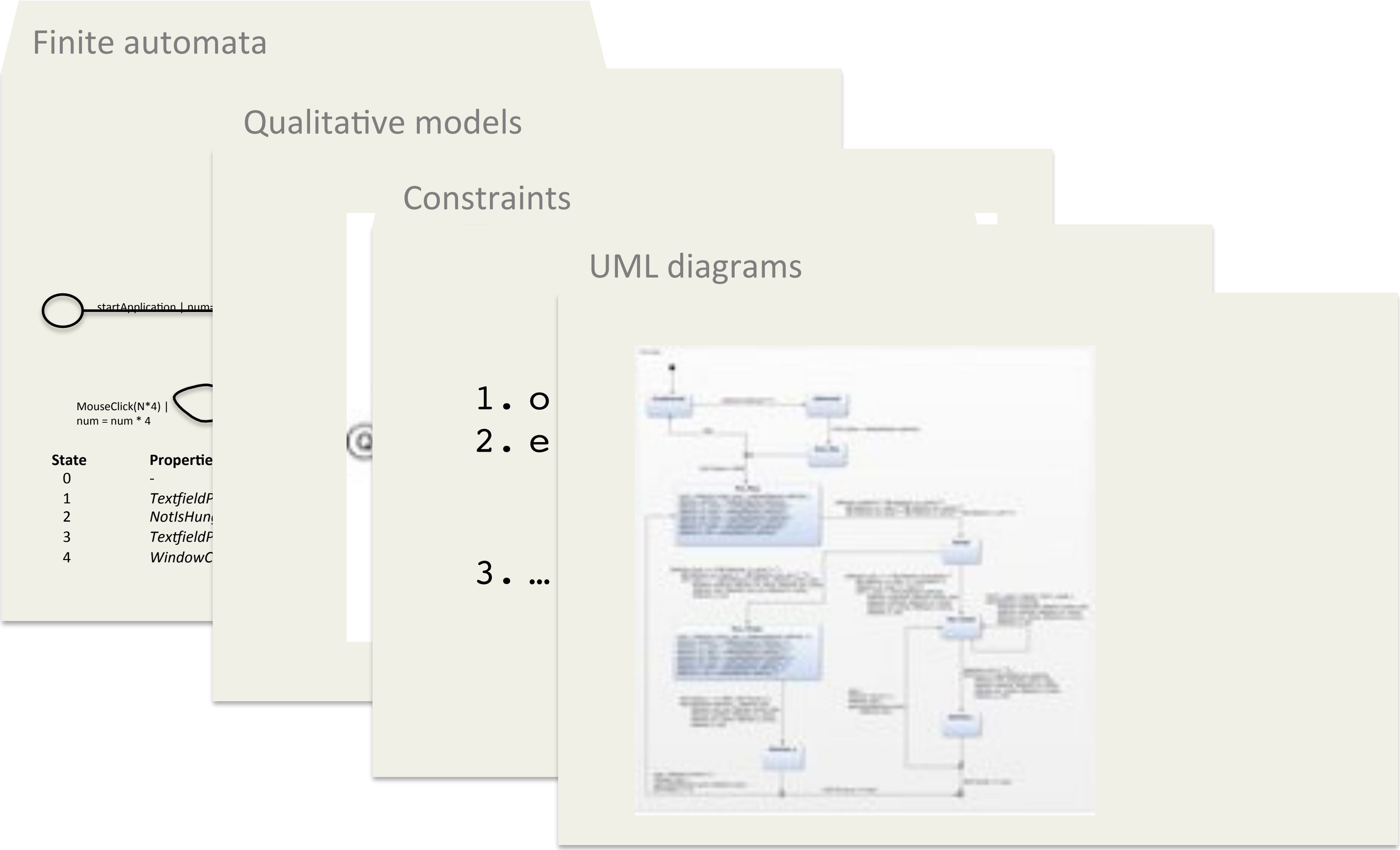
Test automation

- 2 Levels:
  - Automated test case generation
    - From models or the source code (Oracle problem)
  - Automated test execution (e.g. JUnit)
    - Challenges because of different interfaces (Web, different OS platforms, databases, GUIs,…)
    - Hardware In the Loop (HIL) testing

Model-based testing



Model?



Finite automata

State	Propertie
0	-
1	*TextfieldP*
2	*NotIsHun*
3	*TextfieldP*
4	*WindowC*

Qualitative models

Constraints

```
1. o
2. e


3. …
```

UML diagrams

Test case generation

- Directly from the model
  - Equation solving (constraints)
  - Traversing a graph
  - Combination of solving and graph traversal

- Feasible (at least for smaller models)
- Orthogonal to manual testing
- Focus (but not necessarily) system testing

TWO CASE STUDIES

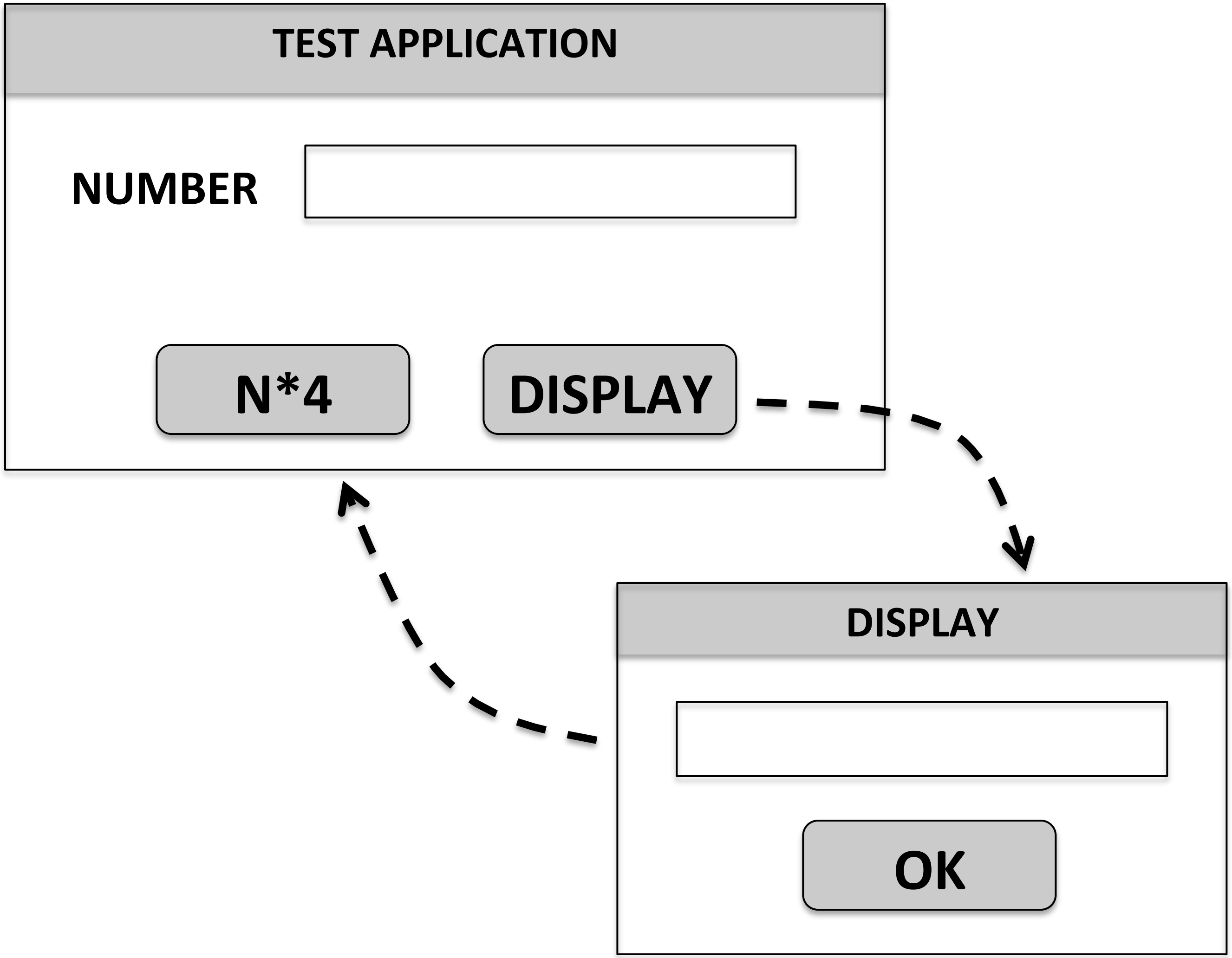
GUI Savvy End-to-End Testing with Smart Monkeys

Birgit Hofer, Bernhard Peischl and **Franz Wotawa**

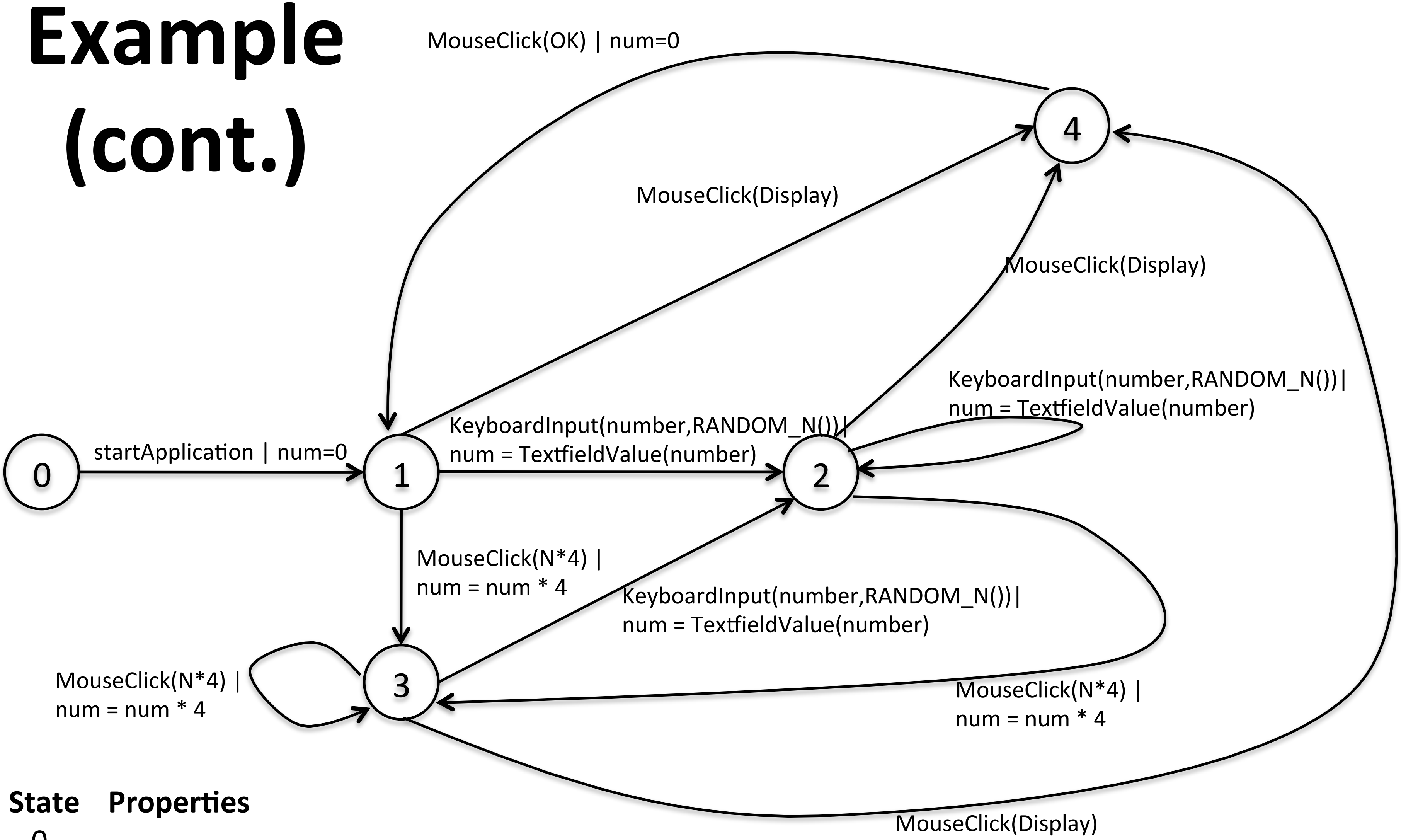
Technische Universität Graz

Institute for Software Technology

Example

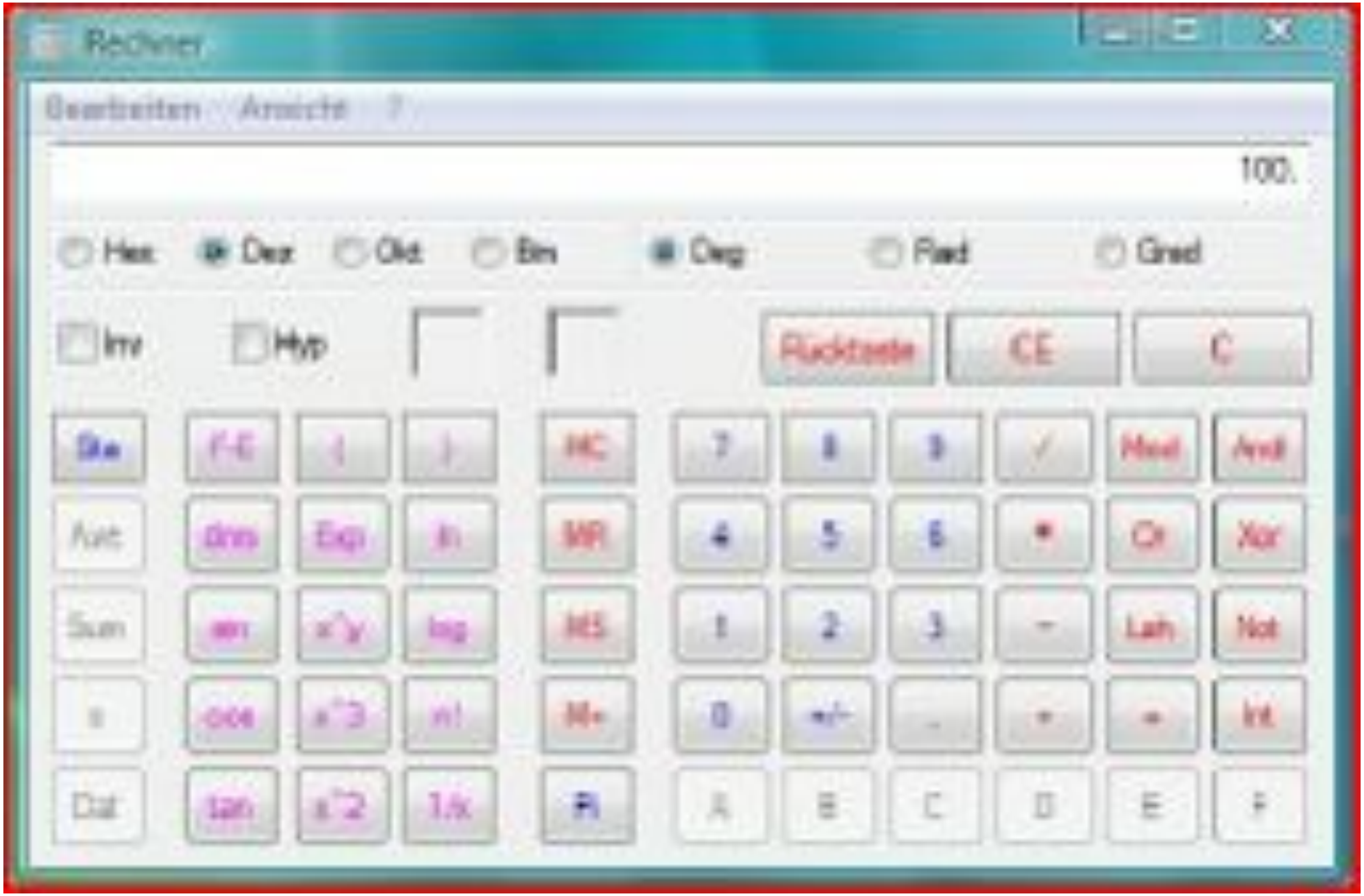


Example (cont.)



State	Properties
0	-
1	*TextfieldProperty(number, 0) ⋀ NotIsHungProperty()*
2	*NotIsHungProperty()*
3	*TextfieldProperty(number,num) ⋀ NotIsHungProperty()*
4	*WindowCaptionProperty(DISPLAY) ⋀ NotIsHungProperty()*

Windows Calculator Case



100.

○ Hex ● Dez ○ Okt ○ Bin

● Deg ○ Rad ○ Grad

□ Inv □ Hyp

Found faults

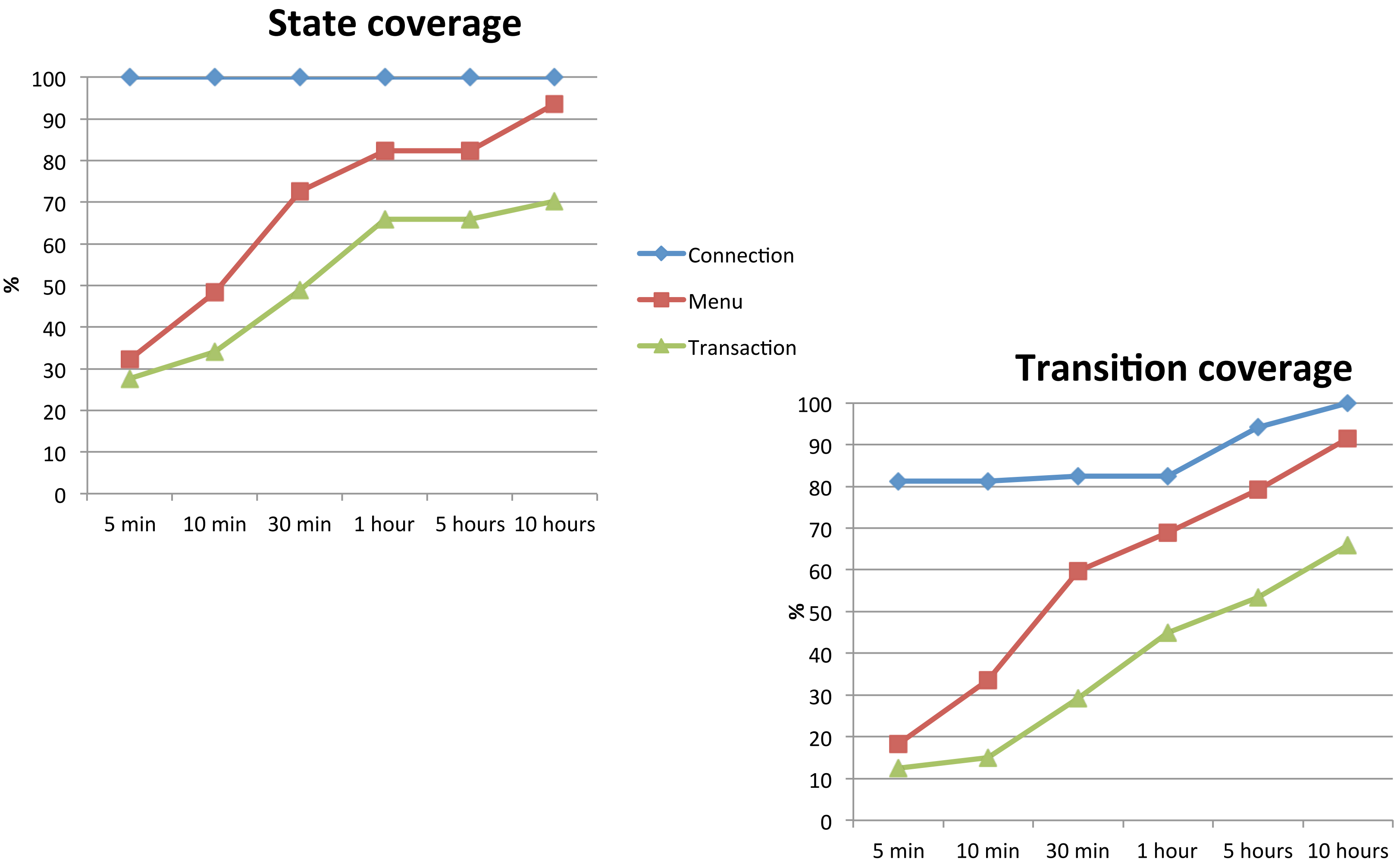
- Event sequenz 1st fault:
  - The monkey produces a division by zero (e.g. *65 / 0*),
  - then it opens the menu item *?/Help*.
  - The value in the text field changes from the error message *‘Division by 0 not possible’* to a number.

- Event sequence 2nd fault:
  - The monkey produces a division by zero,
  - then it opens the menu item *?/Info*.
  - The info menu does not appear

FileZilla Case

- Open Source FTP client (www.filezilla.de)
- 3 Models:
  - Connecting to server (quickconnect bar)
  - Test of menu items (offline test)
  - File operations (transfer, delete,…)
- Models have in sum 113 states and 301 transitions

Model coverage



Code coverage

- **Function coverage**
  - Up to 55 % after 1 hour of testing
- **Condition coverage**
  - Up to 26 % after 10 hours of testing
- **Reasons / Explanations**:
  - Models do not cover the whole functionality
  - Not all GUI elements used in models
  - Not all parts of the code can be tested using the GUI

Fault detection capabilities

- 3 faults introduced in original source code
- All faults found (after 10 hours)
- On average 30 minutes to detect a fault

Coverage Based Testing with Test Purposes

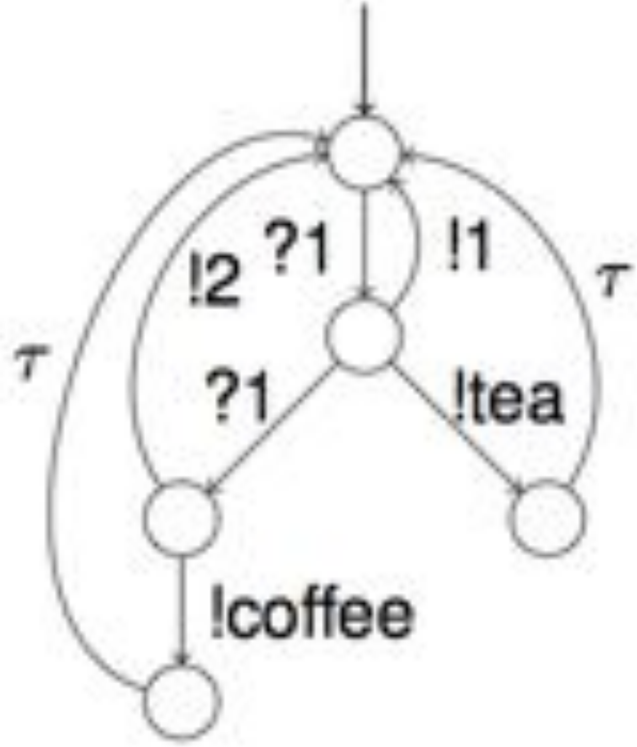
Gordon Fraser Martin Weiglhofer
Franz Wotawa

Institute for Software Technology
Graz University of Technology

QSIC 2008

LTS Model

- Labeled Transition Systems (LTS)

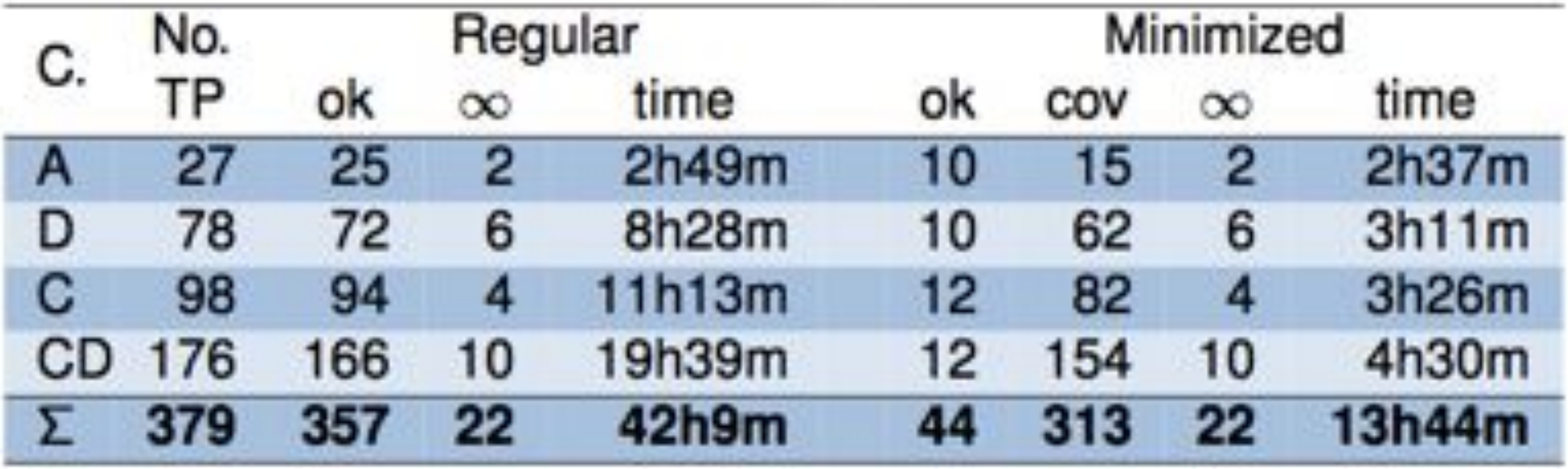


Test case generation

- Test purpose based
- Traversing the model
- Result: Sequence of inputs and outputs

- Case study SIP registrar (VoIP telephony)

Results test generation



C.	No.	Regular			Minimized			
	TP	ok	∞	time	ok	cov	∞	time
A	27	25	2	2h49m	10	15	2	2h37m
D	78	72	6	8h28m	10	62	6	3h11m
C	98	94	4	11h13m	12	82	4	3h26m
CD	176	166	10	19h39m	12	154	10	4h30m
Σ	**379**	**357**	**22**	**42h9m**	**44**	**313**	**22**	**13h44m**

C.	OpenSER				commercial			
	✓	✗	?	↯	✓	✗	?	↯
A	40	1	9	1	27	15	8	2
D	100	16	28	3	72	44	28	3
C	124	25	39	3	86	65	37	3
CD	224	41	67	3	158	109	65	3
Σ	**488**	**83**	**143**	**3**	**343**	**233**	**138**	**3**

C.	OpenSER				commercial			
	✓	✗	?	↯	✓	✗	?	↯
A	12	0	8	0	3	10	7	1
D	10	2	8	2	2	10	8	2
C	12	2	10	2	2	12	10	2
CD	11	2	11	2	1	13	10	2
Σ	**45**	**6**	**37**	**2**	**8**	**45**	**35**	**2**

WHAT’S ABOUT SECURITY TESTING?

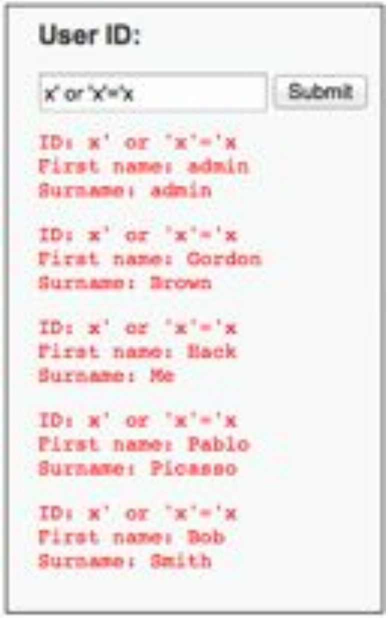
Applications to security testing

- Test case generation based on models of attack patterns!

- **Literature**:
  - Franz Wotawa, *Trust but Verify*, In Proc. ASQT 2012.
  - Josip Bozic and Franz Wotawa*, XSS Pattern for Attack Modeling in Testing*, In Proc. Automation of Software Test (AST), 2013.
  - Josip Bozic and Franz Wotawa, Security Testing Based on Attack Patterns, In Proc. 5th Intl. Workshop on Security Testing (SECTEST), 2014.
  - Josip Bozic, Dimitris E. Simos, and Franz Wotawa*, Attack Pattern-Based Combinatorial Testing* , In Proc. Automation of Software Test (AST), 2014.

Vulnerability Detection

SQLI: x' OR 'x'='x



User ID:

x' or 'x'='x

```
ID: x' or 'x'='x
First name: admin
Surname: admin

ID: x' or 'x'='x
First name: Gordon
Surname: Brown

ID: x' or 'x'='x
First name: Hack
Surname: Me

ID: x' or 'x'='x
First name: Pablo
Surname: Picasso

ID: x' or 'x'='x
First name: Bob
Surname: Smith
```

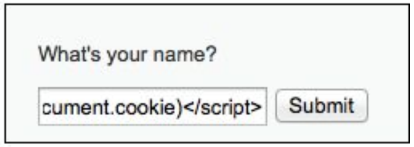
>Success!

```
     <pre>ID: x' or 'x'='x<br>First name: admin<br>Surname: admin</pre><pre>ID: x' or
'x'='x<br>First name: Gordon<br>Surname: Brown</pre><pre>ID: x' or 'x'='x<br>First name:
Hack<br>Surname: Me</pre><pre>ID: x' or 'x'='x<br>First name: Pablo<br>Surname:
Picasso</pre><pre>ID: x' or 'x'='x<br>First name: Bob<br>Surname: Smith</pre>
```

Vulnerability Detection

XSS: `<script>alert(document.cookie)</script>`

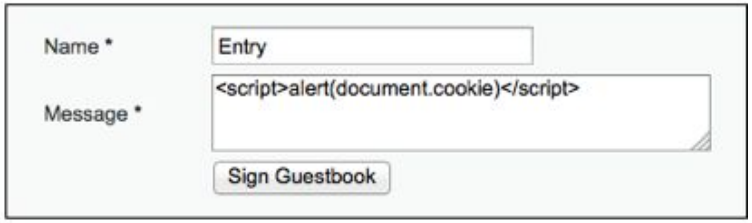
reflected



What's your name?

cument.cookie)</script>

stored



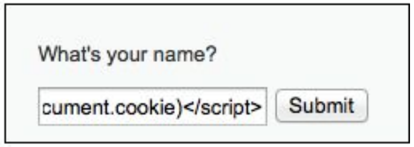
Name * Entry

Message * <script>alert(document.cookie)</script>

Vulnerability Detection

XSS: `<script>alert(document.cookie)</script>`

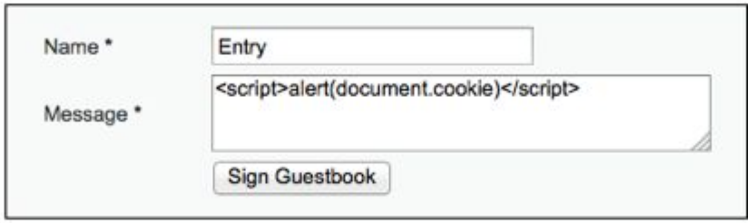
reflected



What's your name?

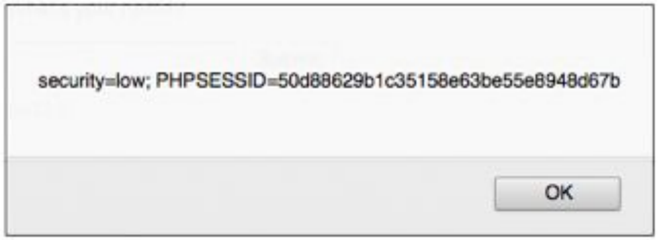
cument.cookie)</script>

stored



Name * Entry

Message * <script>alert(document.cookie)</script>

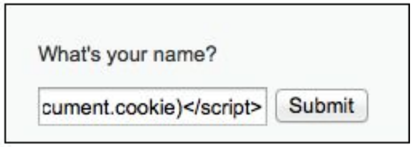


```
security=low; PHPSESSID=50d88629b1c35158e63be55e8948d67b
```

Vulnerability Detection

XSS: `<script>alert(document.cookie)</script>`

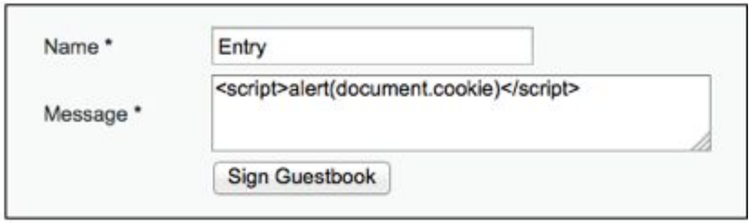
reflected



What's your name?

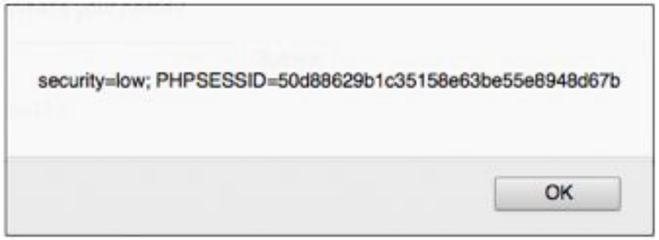
cument.cookie)</script>

stored



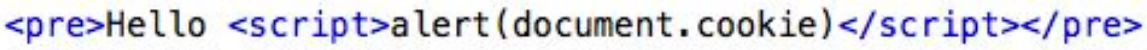
Name * Entry

Message * <script>alert(document.cookie)</script>



security=low; PHPSESSID=50d88629b1c35158e63be55e8948d67b

> Success!

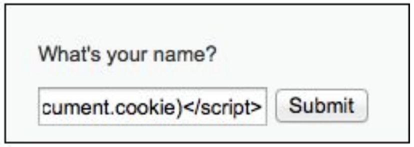


```
<pre>Hello <script>alert(document.cookie)</script></pre>
```

Vulnerability Detection

XSS: `<script>alert(document.cookie)</script>`

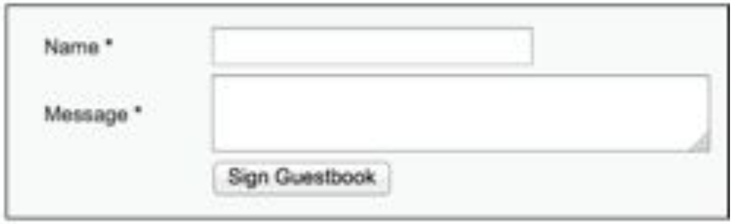
reflected



What's your name?

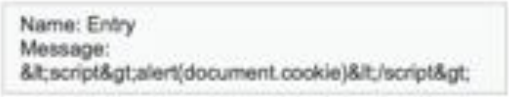
cument.cookie)</script>

stored



Name *

Message *

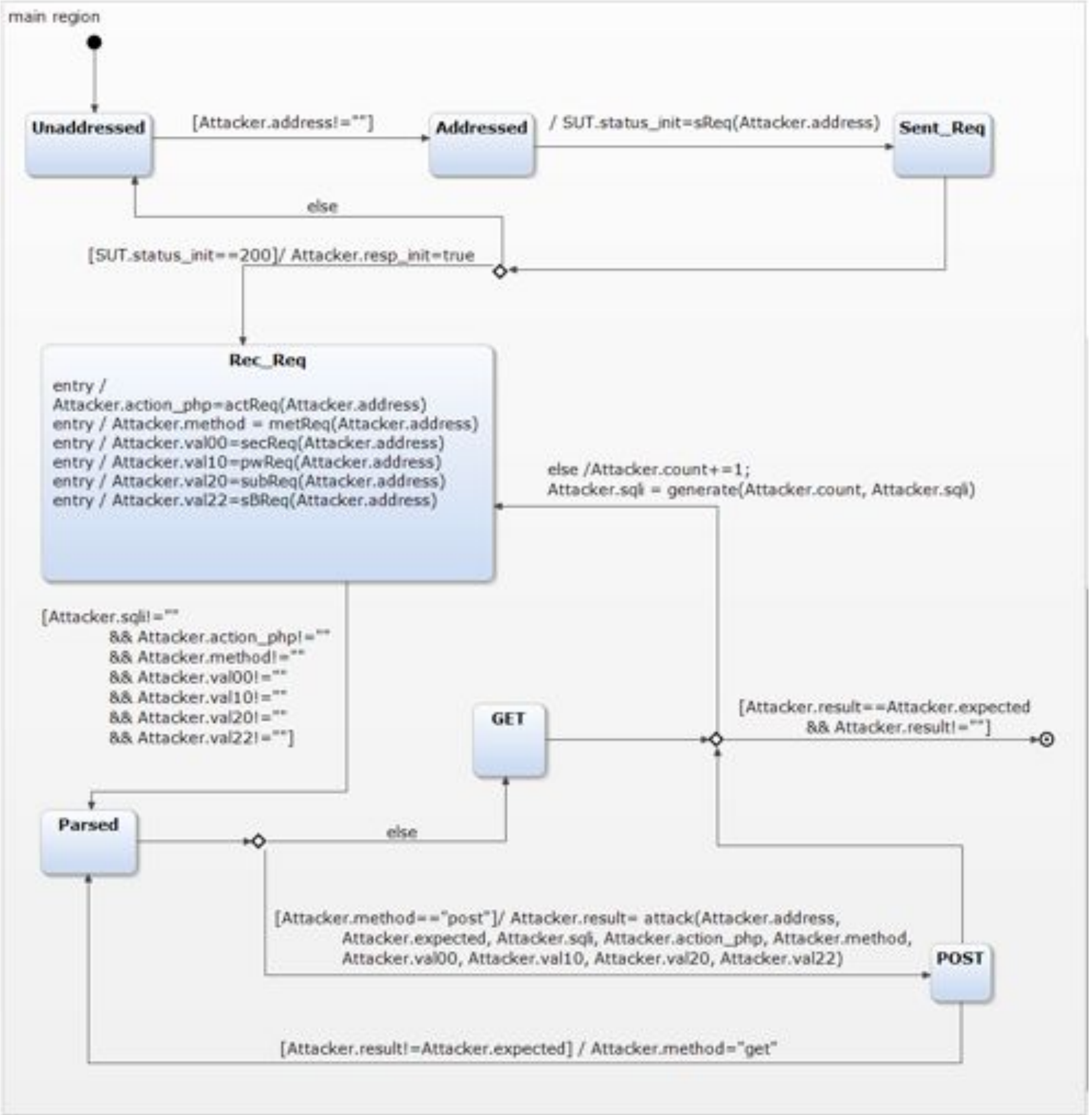


Name: Entry
Message:
<script>alert(document.cookie)</script>

> Failure!

```
<pre>Hello &lt;script&gt;alert(document.cookie)&lt;/script&gt;</pre>
```

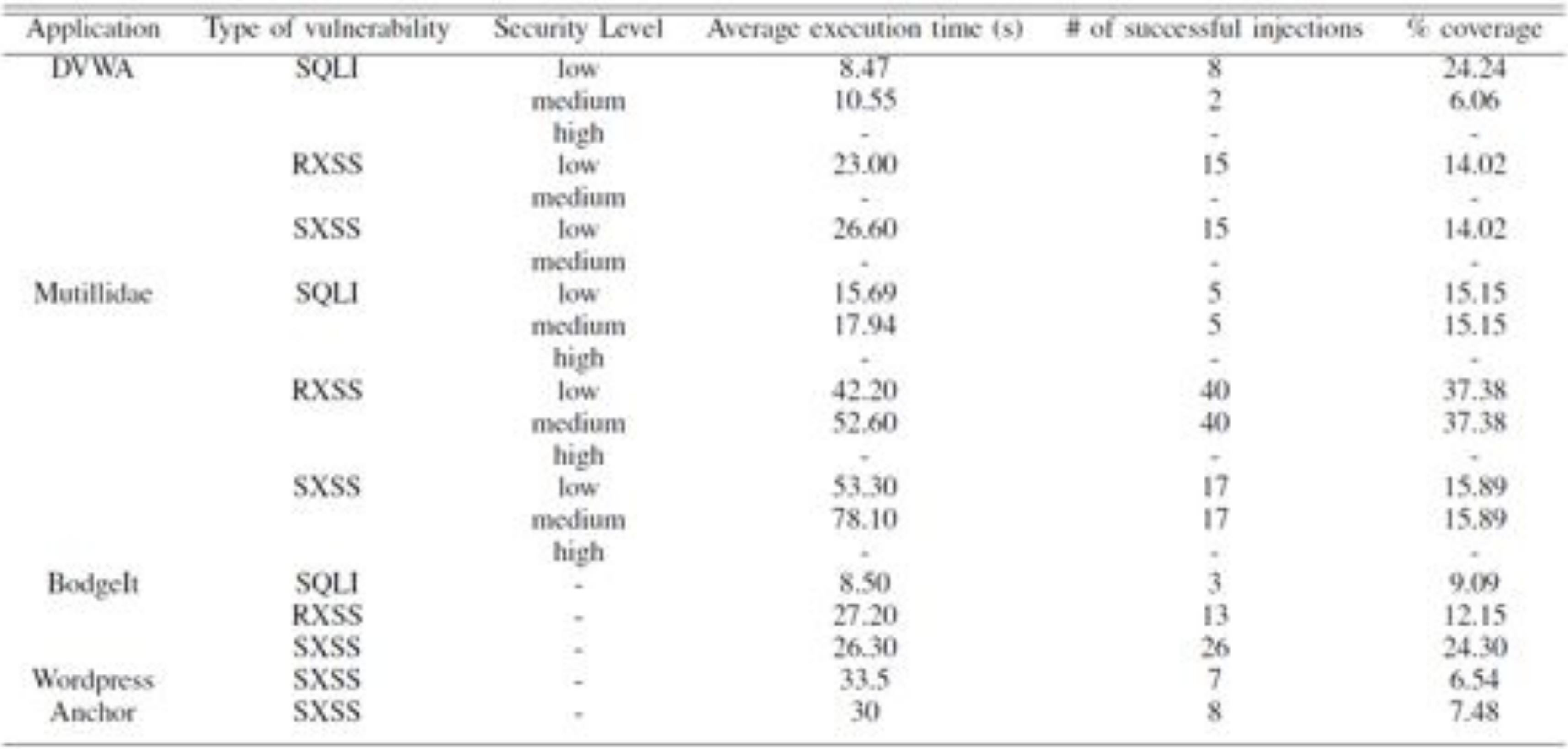
Model-based security testing


main region
Unaddressed
[Attacker.address!=""]
Addressed
/ SUT.status_init=sReq(Attacker.address)
Sent_Req
else
[SUT.status_init==200]/ Attacker.resp_init=true
Rec_Req
entry /
Attacker.action_php=actReq(Attacker.address)
entry / Attacker.method = metReq(Attacker.address)
entry / Attacker.val00=secReq(Attacker.address)
entry / Attacker.val10=pwReq(Attacker.address)
entry / Attacker.val20=subReq(Attacker.address)
entry / Attacker.val22=sBReq(Attacker.address)
else /Attacker.count+=1;
Attacker.sqli = generate(Attacker.count, Attacker.sqli)
[Attacker.sqli!=""
&& Attacker.action_php!=""
&& Attacker.method!=""
&& Attacker.val00!=""
&& Attacker.val10!=""
&& Attacker.val20!=""
&& Attacker.val22!=""]
GET
[Attacker.result==Attacker.expected
&& Attacker.result!=""]
Parsed
else
[Attacker.method=="post"]/ Attacker.result= attack(Attacker.address,
Attacker.expected, Attacker.sqli, Attacker.action_php, Attacker.method,
Attacker.val00, Attacker.val10, Attacker.val20, Attacker.val22)
POST
[Attacker.result!=Attacker.expected] / Attacker.method="get"

Evaluation

- Five applications: NOWASP (Mutillidae) [8], Damn Vulnerable Web App (DVWA) [9], BodgeIt [10], Wordpress [11] and Anchor CMS [12].
- First three contain several security levels with every one having more sophisticated filtering mechanisms.
- Other programs are tested only for the second type of XSS because these are blog software, where posts are stored inside a database.
- All applications have been deployed on an Apache Server and comprise a MySQL database.
- Collection of 33 custom SQLI and 107 XSS input strings.

Evaluation



Application	Type of vulnerability	Security Level	Average execution time (s)	# of successful injections	% coverage
DVWA	SQLI	low	8.47	8	24.24
		medium	10.55	2	6.06
		high	-	-	-
	RXSS	low	23.00	15	14.02
		medium	-	-	-
	SXSS	low	26.60	15	14.02
		medium	-	-	-
Mutillidae	SQLI	low	15.69	5	15.15
		medium	17.94	5	15.15
		high	-	-	-
	RXSS	low	42.20	40	37.38
		medium	52.60	40	37.38
		high	-	-	-
	SXSS	low	53.30	17	15.89
		medium	78.10	17	15.89
		high	-	-	-
BodgeIt	SQLI	-	8.50	3	9.09
	RXSS	-	27.20	13	12.15
	SXSS	-	26.30	26	24.30
Wordpress	SXSS	-	33.5	7	6.54
Anchor	SXSS	-	30	8	7.48

Evaluation

- Both attack patterns have been slightly adapted.
- Wordpress was tested while our application was authenticated so all inputs were submitted after that step.
- Anchor CMS is similar to Wordpress with the difference that all posts have to be approved by the administrator.
- It was impossible to detect vulnerability on the hardest security level of the first three apps, which means that a more sophisticated test case generation strategy has to be adapted for this purpose.
- In Mutillidae, HttpClient enables communication on medium and hard level.

What’s next?

- Modeling of attacker
- Idea:
  - attack = sequence of actions = plan
  - use A.I. planning for attack generation
  - more flexible

Conclusion

- Model-based testing finds faults that have been previously undetected (using manual tests)
- Completely automated generation
- Requires model (+ test purposes)
- Complementary to manual testing
- Can be used for security testing too!

Save the date!

ICST 2015

8th IEEE International Conference on
Software Testing, Verification and Validation

13 - 17 April 2015, Graz, Austria

Franz Wotawa (General Chair)
Gordon Fraser and Darko Marinov (PC-Chairs)

Keynotes, Technical tracks, Workshops, PhD Symposium, Exhibitions, Panel

icst2015.ist.tugraz.at

General Chair:

- **Franz Wotawa**
  (TU Graz, Austria)

PC Chairs:

- **Gordon Fraser**
  (Univ. Sheffield, UK)
- **Darko Marinov**
  (Univ. of Illinois, Urbana-Champaign, USA)

Thank you for your attention!

“What I cannot create, I do not understand.”

Richard Feynman
(1918-1988)