# Fault-Based Testing

Alexander Pretschner, TU München
UCAAT, Munich, 17/9/2014

# Agenda

- ► Good tests
- ► Why coverage shouldn‘t be used a-priori
- ► Fault models
- ► Testing based on fault models
- ► Discussion

# Agenda

- Good tests
- Why coverage shouldn‘t be used a-priori
- Fault models
- Testing based on fault models
- Discussion

# What’s a good test case?

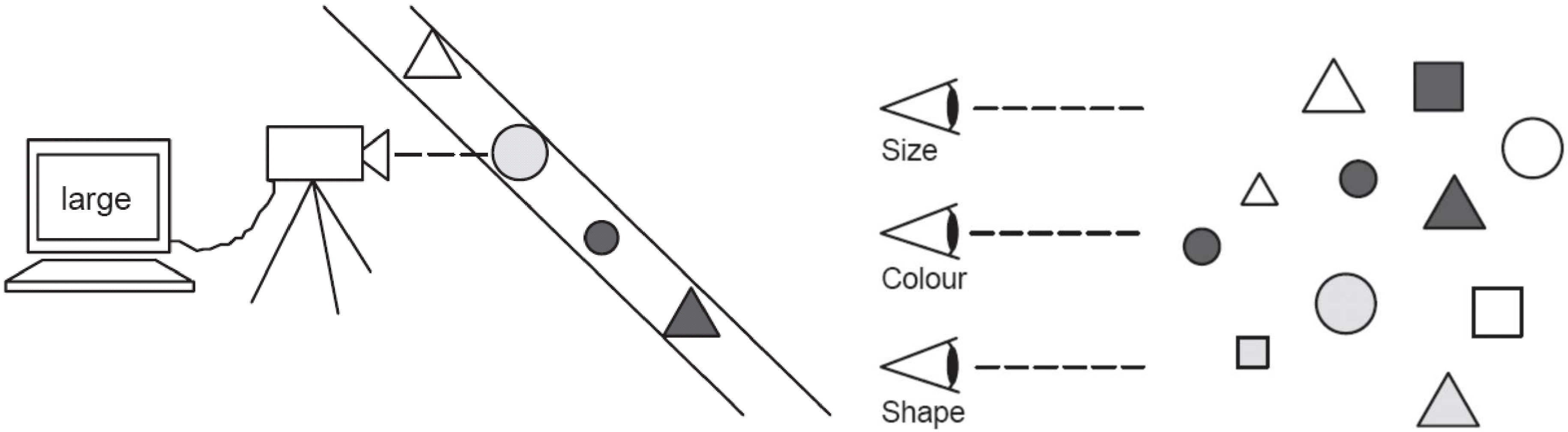
- “Ability to detect failures”
  - No good test cases for a perfect program!
- “Ability to detect potential failures”
  - “Potential”? Effort?
- “Ability to detect potential (or: likely) failures with good cost-effectiveness”
  - Writing/executing/evaluating/maintaining the test
  - Remaining failures in the field—severity
  - Going from failure to fault
- Perfect! And useless!

# Coverage-Based Testing

- Challenge: operational, measurable quality of tests
  - „Adequacy“: selection, stopping, assessment criteria
- Adequacy criteria induce partition of input domain
  - Requirements
  - Coverage criteria
  - [Faults]
- Coverage a good response?

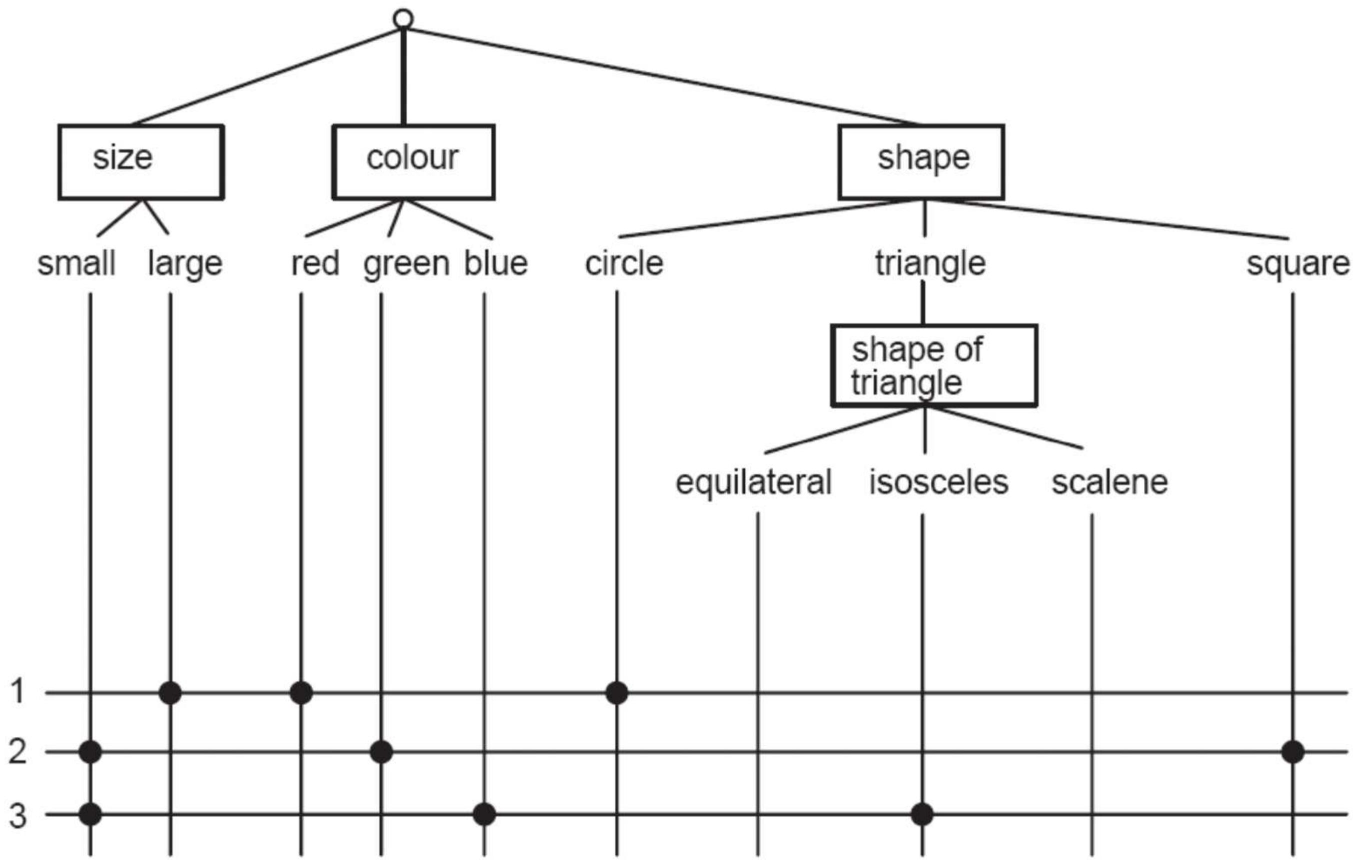
# Input space partition: category-partition method

- Consider input space "under various aspects"
- For each "aspect", form disjoint and complete set of classes
- (Iterate: build recursive classification)
- Instantiate classes so that the input domain is "covered"




[Grochtmann, Wegener, Grimm '95]

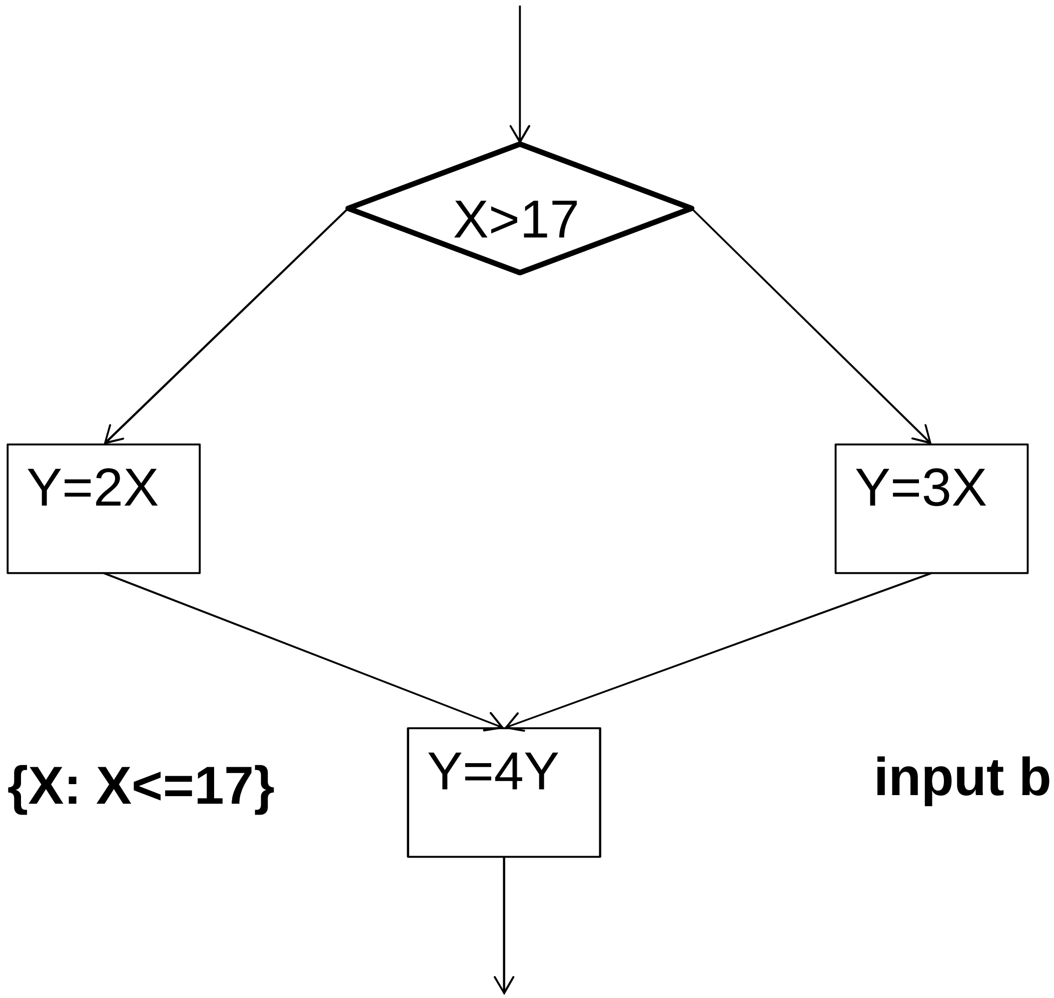
# Input space partition: category-partition method




[Grochtmann, Wegener, Grimm ’95]

# Input Space Partitioning: Coverage Criteria

```
if(X>17)
  Y=2*X;
else
  Y=3*X;
Y=4*Y;
```




**input block 1: {X: X<=17}**

**input block 2: {X: X>17}**

# Bottom line

- Coverage-based testing instance of partition-based testing

- [Coverage: statement/branch/condition/MCDC … coverage; also def-use pairs]

# Agenda

- Good tests?
- Why coverage shouldn‘t be used a-priori
- Fault models
- Testing based on fault models
- Discussion

# Simple decision

Pick two test cases for

```
if x==1
   f(g(h(i(j(k(l(m(x))))))))
else
   m(l(k(j(i(h(g(f(x))))))))
endif
```

[nondeterministic f..m]

# Simple decision

Now, pick two test cases for

```
if x==1
   f(g(h(i(j(k(l(m(x))))))))
else
   f(g(h(x)))
endif
```

# Simpler decision

And now, pick two test cases for

```
if x==1
   f(g(h(i(j(k(l(m(x))))))))
else
   print „Gott mit Dir, Du Land der Bayern“
endif
```

# So what?

- Structural criterion a good idea?

- Fault model matters!

# Disclaimer

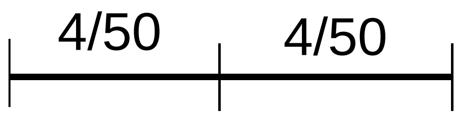
- Truth somewhat more complicated:
  coverage criteria usually applied to all function definitions, not just the main function

- General idea applicable nonetheless

- Plenty of empirical evidence that coverage is not helpful when used a-priori, mixed findings for a-posteriori usage
  most recent [Inozemtseva&Reid'14]

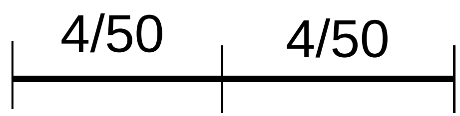
# Random and Partition Testing

- Partition testing can be better, worse, or the same as random testing
  - 8 in 100 inputs failure-causing, select n=2 tests
  - $P_r = 1-(1-\theta)^n = 1-(1-.08)^2 = .15$
- k=2 subdomains
  - $P_p = 1-\Pi_{1 \le i \le k}(1-\theta_i)^{n_i} = 1-(1-4/50)^2 = P_r$

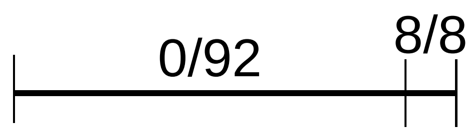


# Random and Partition Testing

- Partition testing can be better, worse, or the same as random testing
  - d=100, 8 inputs failure-causing, n=2 tests to be selected
  - $P_r=1-(1-.08)^2=.15$
- k=2 subdomains
  - $P_p=1-\Pi_{1\leq i\leq k}(1-\theta_i)^{ni}=1-(1-4/50)^2=P_r$

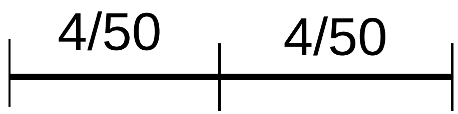



  - $P_p=1>P_r$

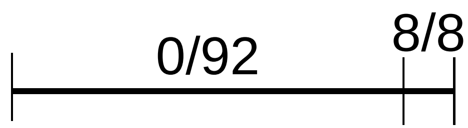


# Random and Partition Testing

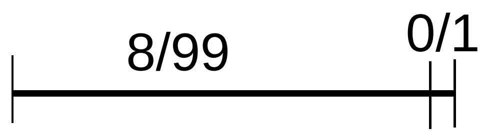
- Partition testing can be better, worse, or the same as random testing
  - d=100, 8 inputs failure-causing, n=2 tests to be selected
  - $P_r=1-(1-.08)^2=.15$
- k=2 subdomains
  - $P_p=1-\Pi_{1\leq i\leq k}(1-\theta_i)^{ni}=1-(1-4/50)^2 = P_r$




  - $P_p=1 > P_r$




  - $P_p=1-(1-0/1)*(1-8/99)=.08 < P_r$



# Results (Weyuker&Jeng 1991)

- In general, partition based can be as good as, better than, or worse than random testing
  - **Fault-prone blocks not known in advance**

- [yes several reasonable objections to this model]
- [Generalizations]

# Discussion

- If a-priori failure likelihoods are not known (or their characteristics or characteristics of their expectation), then partition-based testing **can be good or bad**!
- Yes, coverage is good from a management perspective.
Yes, MC/DC coverage is required by DO 178-B.
Yes, we can automate the derivation of tests.

- **But, we do it because we can and because one number is better than no number, not because it would, from a failure detection perspective, make sense!**

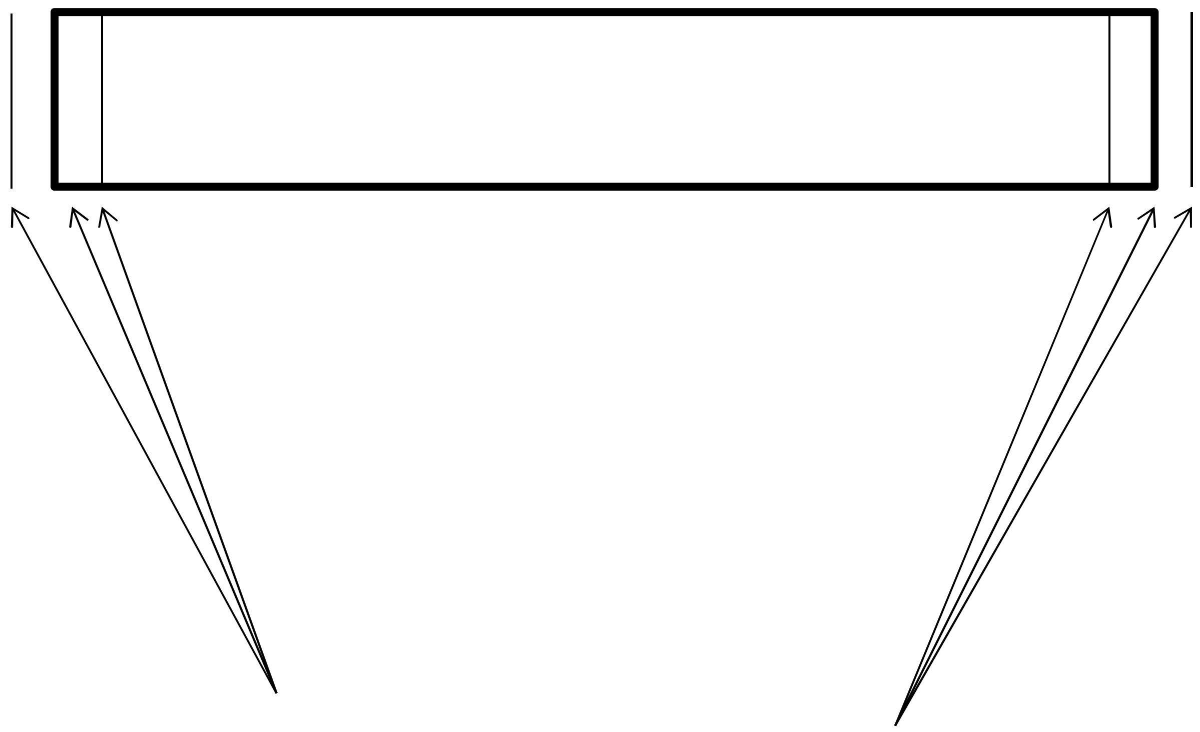
# Disclaimer II

- Random testing really such a good idea?

# Agenda

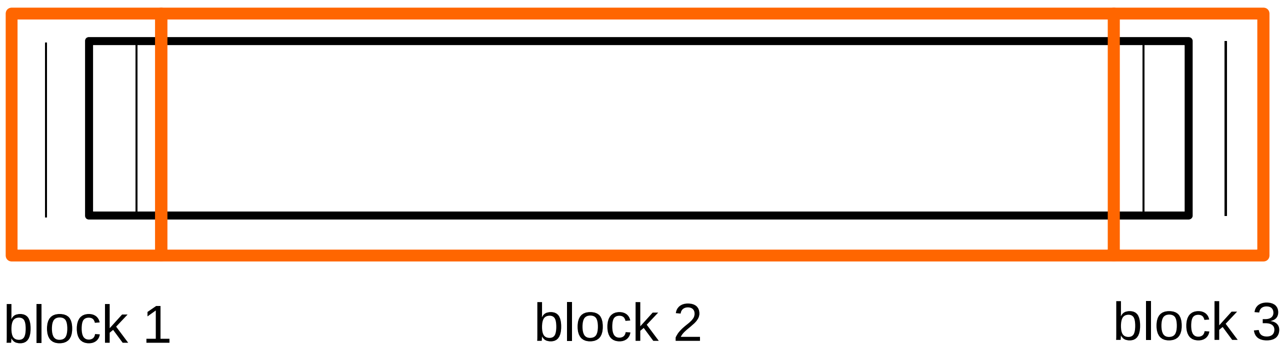
- ► Good tests?
- ► Why coverage shouldn‘t be used a-priori
- ► Fault models
- ► Testing based on fault models
- ► Discussion

# Limit testing?



Empirically speaking, trouble!

# Limit testing?




Blocks 1 and 3 with higher expected failure rates
Plus, comparably small w.r.t. block 2
Hence: can expect $E(P_p)>E(P_r)$

# What‘s this?

- … a fault model!

# Fault models

- Limit testing
- Deadlocks, order violations, atomicity violations
- Incorrect transition, sneak paths, trap doors, corrupt states …
- Invariant violations in subclass
- Syntactic problems as used in mutation testing
- Combinatorial testing

- Domain-specific faults

# Fault models [Morell 1991, Pretschner et al. 2013]

- Faults are delta with correct programs
- Fault models are descriptions of mappings from correct to incorrect programs and/or characterizations of hypothesized failure domains
  - Combinatorial testing special case
  - Limit testing easier to grasp by failure domain
- „Effective" fault models simple to define

# Fault models

- Limit testing
- Deadlocks, order violations, atomicity violations
- Incorrect transition, sneak paths, trap doors, corrupt states …
- Invariant violations in subclass
- Syntactic problems as used in mutation testing
- Combinatorial testing

- Domain-specific faults

# Agenda

- ► Good tests
- ► Partition-based testing: On „equivalence classes“
- ► Why coverage shouldn‘t be used a-priori
- ► Fault categories and models
- ► Testing based on fault models
- ► Methodology and Formalization
- ► Discussion

# Example I: Legacy Business IT

- Recurring faults

Project P1

**RPG**:

- System state management
  - Variables not re-initialized between workflows
  - State kept in temp DB tables
- Hard-coded values
- Incorrect data types
- Too loose or too restrictive checks
- Arithmetic bugs
- …

Project P2

**Cobol**:

- System state management
  - Global variable reuse
- Hard-coded values
- Arithmetic bugs
- Too loose or too restrictive checks
- Incorrect data types

**PowerBuilder**:

- Variables not re-initialized between workflows

**PL/SQL**:

- Too loose or too restrictive checks

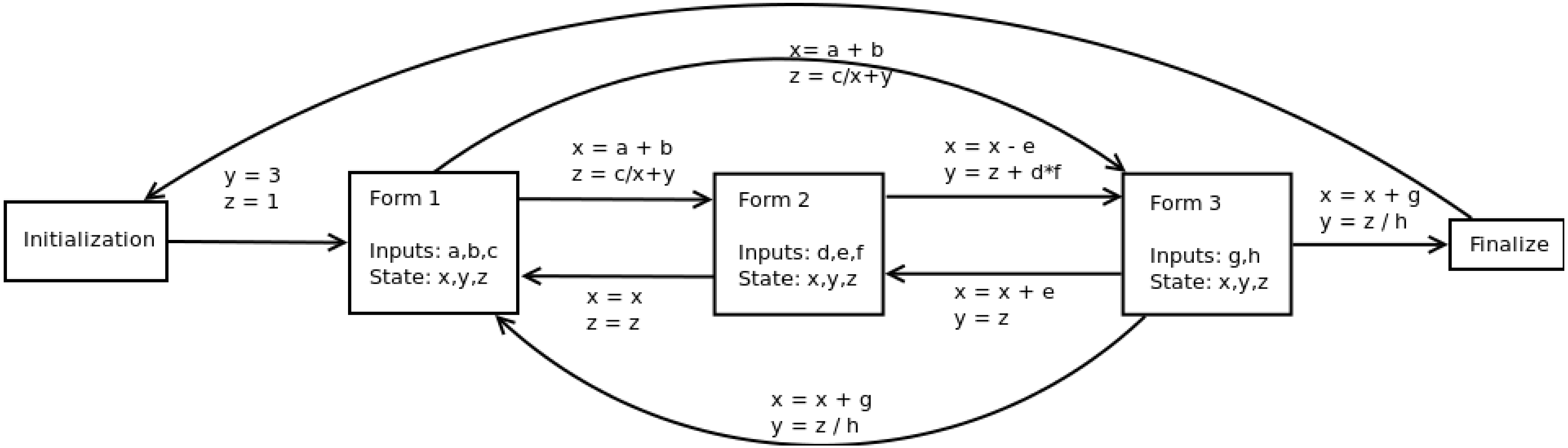
# Aggregated View: Examples

| Fault |
| --- |
| Too loose or too restrictive checks / conditions |
| System state management (has sub categories) |
| Variables not re-initialized between workflows |
| Global variable reuse |
| State kept in temporary DB table |
| Hard-coded values |
| Incorrect data types |
| Arithmetic bugs |

- And so on …

# Example: Unintended Workflows

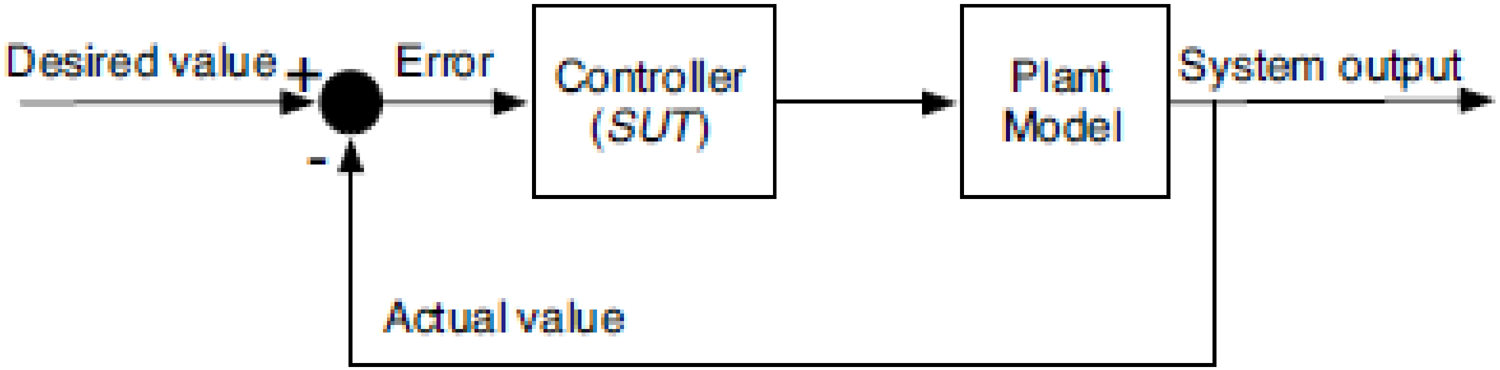
- **Problem**: navigating between forms in different ways leads to different results (failures)
- **Idea**:
  - Compare operations performed between forms (states) in different workflows
  - Use only “Next” button in GUI to determine intended or correct workflow
  - Test <u>un-intended workflows</u> dynamically to find high severity failures



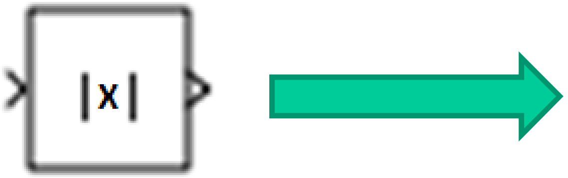
# Example II: Continuous Systems

- Implementation of controllers in Matlab/Simulink
- Example 1
  over/underflows; division by zero (or close-to-zero)
  … using smells
  A fault model.
- Example 2
  problems if intended value smaller than current value –
  usually, tests only for larger values
  Rather a failure model.

# Continuous controllers and plants



# Overflowing Abs – A Typical Fault Model




```
if x < 0:
    return -x
if x > 0:
    return x
```

Example: 8-bit signed integer

-128 = 10000000
(two complement)

→

Multiply with (-1) :
- Invert all bits -> 01111111
- Add 00000001

→

Result:
10000000 = -128
Once again!

# Division by Small Value

- 8-bit unsigned fixed point value with 4 bits before and 4 after the comma.

Max value before comma: 15

| 0 | 0 | 0 | 0 | . | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|

Max value after comma: .9375

Total value range: $15\frac{15}{16}, \ldots, 15\frac{1}{16}, 15, \ldots, \frac{15}{16}, \ldots, \frac{1}{16}, 0$

- Example:

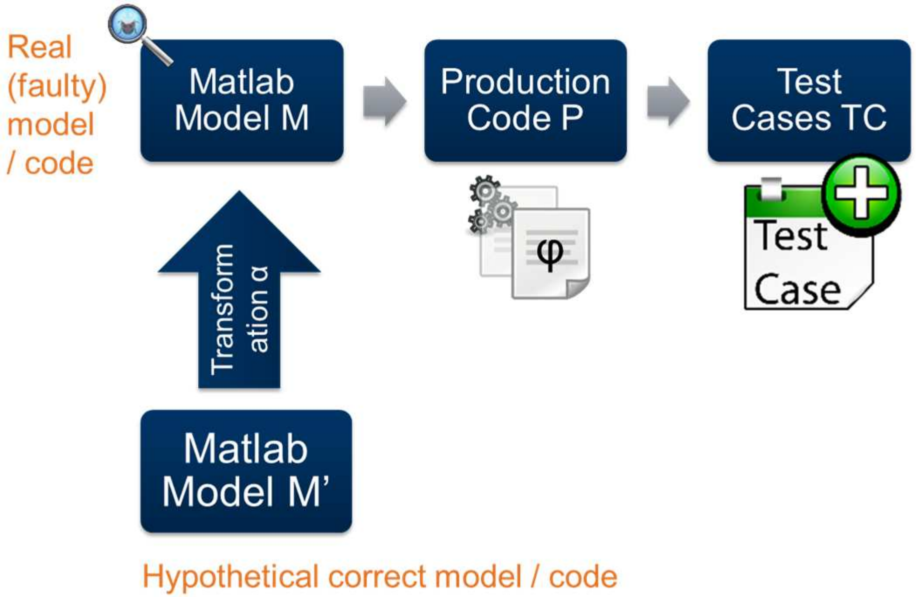
$$\frac{8}{\frac{1}{16}} = 8 \cdot 16 = 128$$

128 is far greater than the highest number (15.9375) that we could store in a 4/4-bit fixed point value

8 (decimal) as fixed point binary: 1000.0000, 1/16 as fix.p.bin: 0000.0001

# 8Cage [Holling et. al 2014]

- Analyze models for potential faults (smells)
- Derive and execute test as evidence for actual fault:
  Use potential faults to provoke failures
- Dynamic addition of further fault models




=> Early fault detection and direct localization in the model

- Demo video offline if you wish

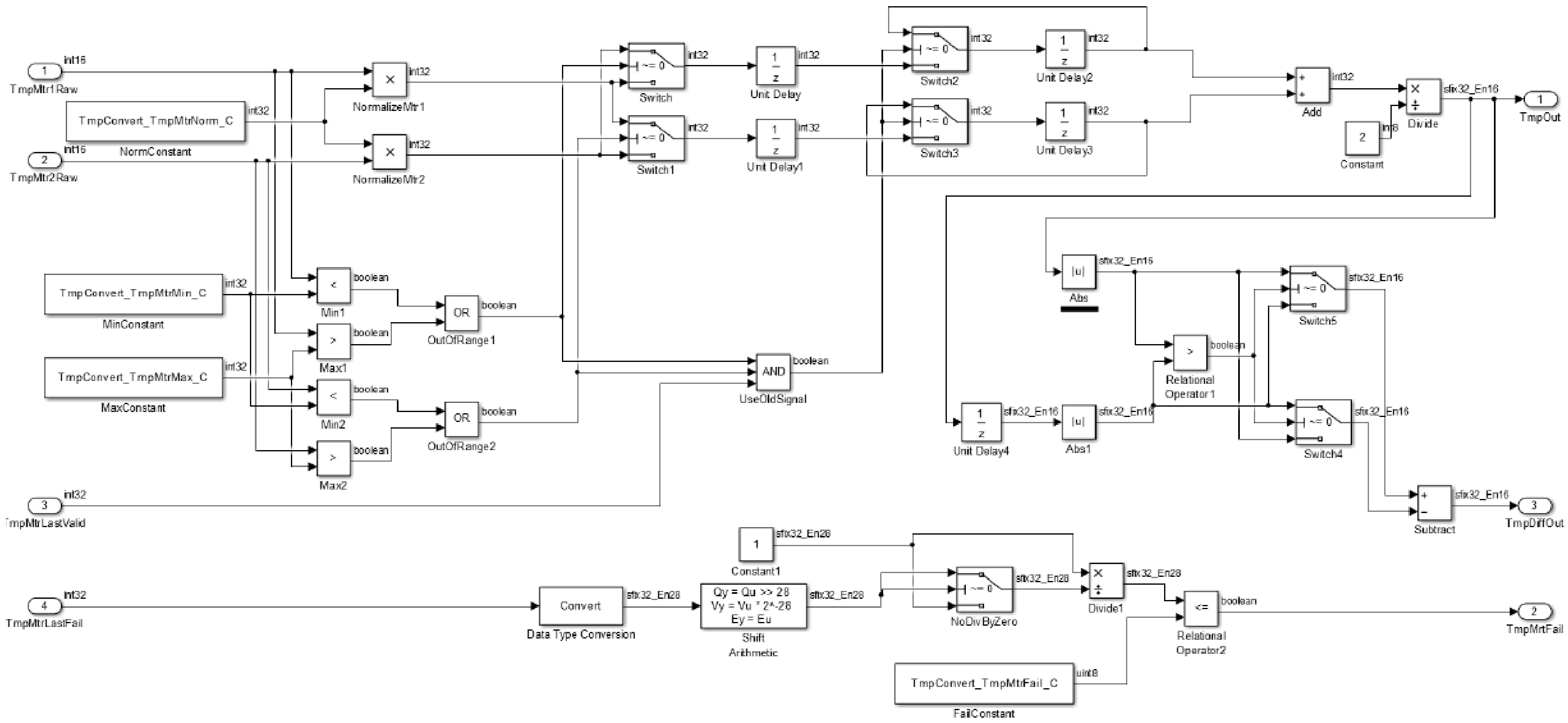
# Can‘t we use static tools instead?

Yes we can.

But they are costly, both in terms of licenses and man power, and „trivial“ faults are annoying to the analyst – and expensive.

Similar reasoning for check lists.

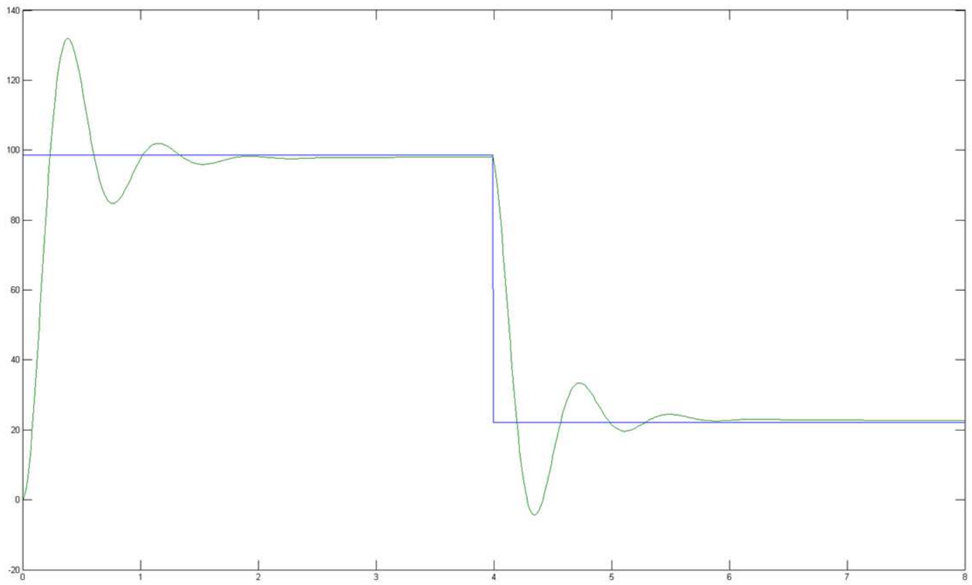
# Example Controller



# Fault Model for Continuous Systems (Failures)

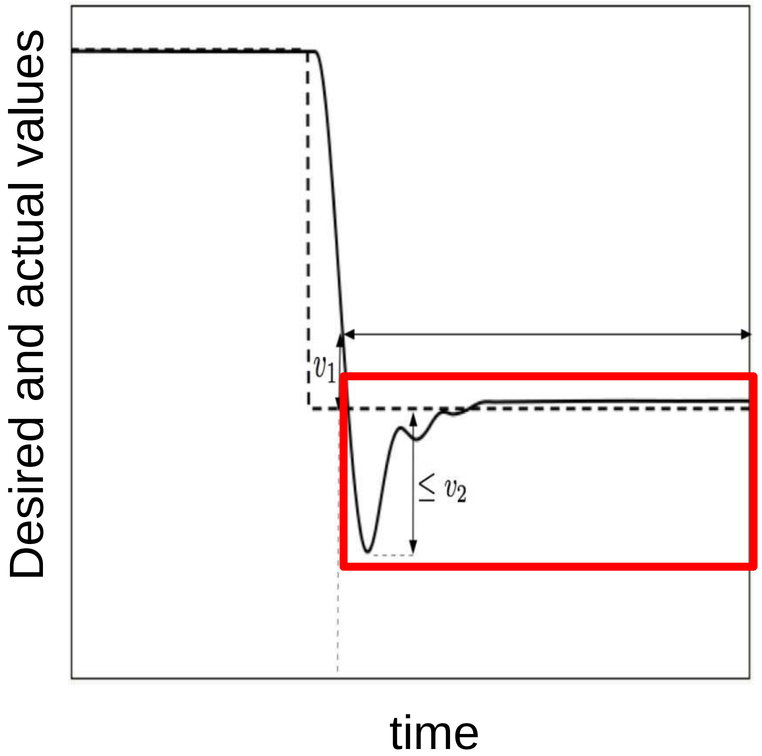
- Complete test even more impossible than usual …
- Experts write representative tests
- Frequent assumption: controller is in initial state (that is, 0)
- Hence only „positive“ computations starting at 0

⇒ Sufficient to test requirements such as stability, responsiveness etc.?




⇒ Results by Matinnejad et al. 2013, 2014

# Controller requirement – Smoothness



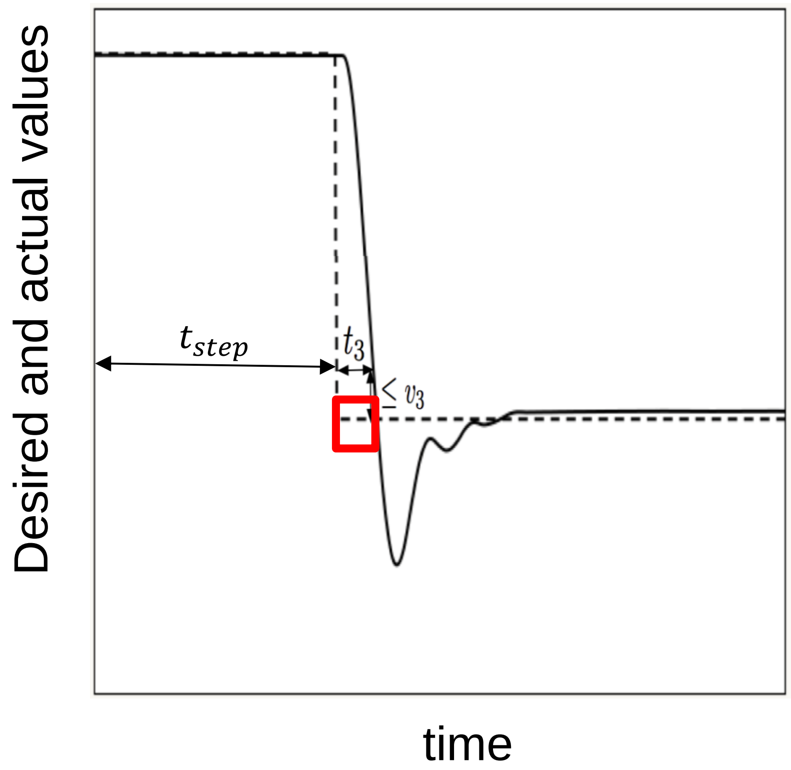

**Intuition:** No large over/undershoots once close to desired value

**Measurement:** *max(|Actual(t) – Desired(t)|)*

$where\ t_{v_1} \leq t \leq t_{end}$ and $t_{v_1}$ time when |*Actual(t)* – *Desired(t)*| < $v_1$ for the first time

**Goal**: maximum error ≤ $v_2$ *after* $t_{v_1}$

# Controller requirement – Responsiveness




**Computation:** First time until error less or equal to $v_3$
**Measurement:** *time (=* $t_3$*)* from $t_{step}$ until |*Actual(t) – Desired(t)*|<= $v_3$ for the first time
**Goal:** Check if $t_3$ is within required bounds

# Credits

- Text book properties
- Ideas borrowed from Matinnejad et al.
- Our definitions slightly different
- Close relationship with standard controller quality criteria: $L^1$, $L^2$, ITAE, max norms

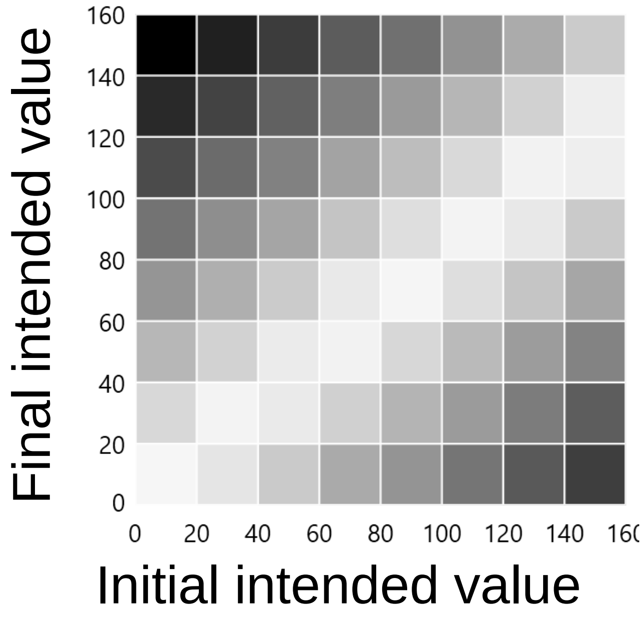
- Many more: removal of opposing force, oscillation, discretization, …

# Approach

► Simulation with two intended values (fault model)
  - First half: get system to initial intended value
  - Second half: get system to final intended value

Step 1 [Matinnejad et al. 2013]:
  - Partition input space into blocks
  - Randomly select N points per block
  - Assess requirement satisfaction per point
  - Create heatmap (brighter block = better satisfaction)

Step 2 [Matinnejad et al. 2013]:
  - Use more fine-grained AI search methods for selected blocks
  - Find global maximum of deviation for blocks



Stability heat map

Further fault models, e.g. oscillation of plant after reaching intended value.
[Identifying these fault models is the crucial part!]

- Demo video offline if you wish

# Discussion

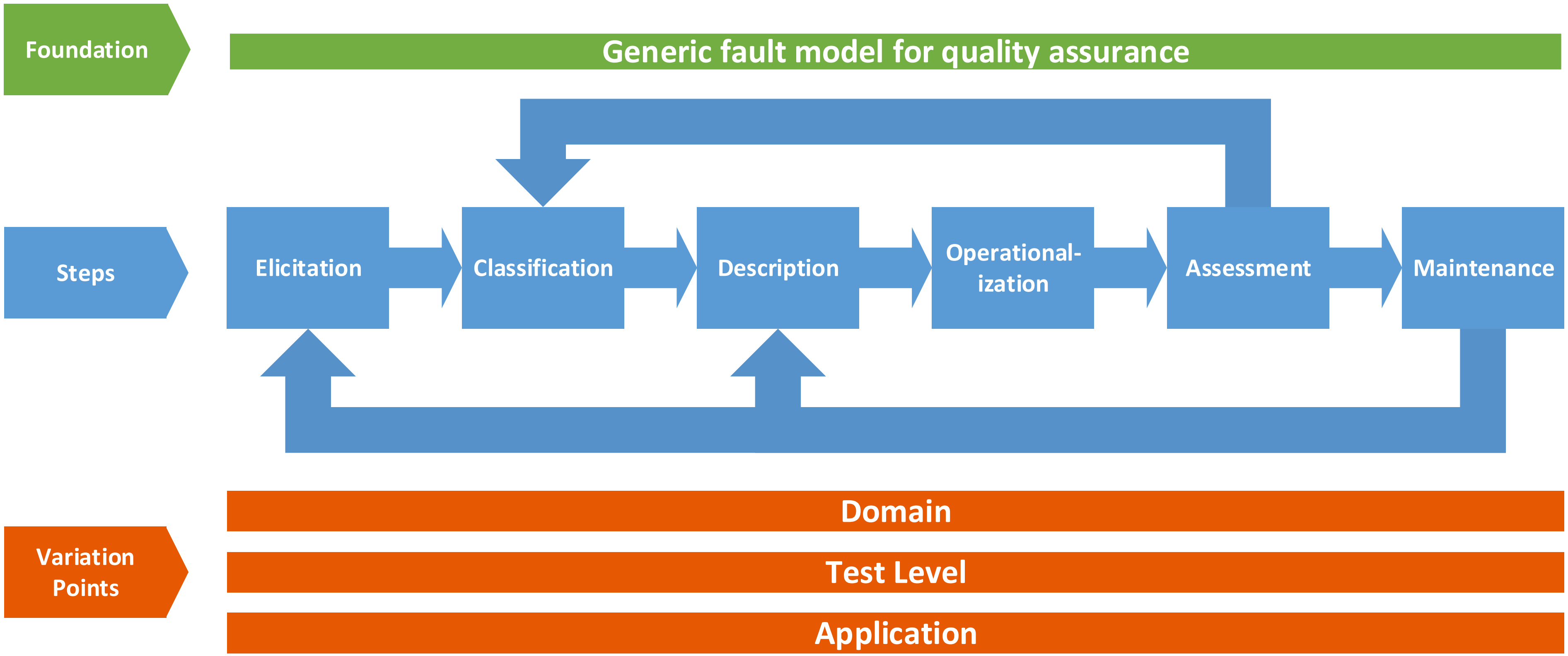
- Controller designers know what they are doing
- Various industry partners report they don‘t have these problems
- More interesting situation for cascading controllers

- TUM open source implementation
  Source: https://github.com/AlvinStanescu/ControllerTester
  Installer: http://sourceforge.net/projects/controllertester/

# How to Describe Fault Models

- Ad-hoc implementations
- Currently working on generic description

# Process: Big Picture



# Agenda

- Good tests?
- Partition-based testing
- Why coverage shouldn‘t be used a-priori
- Fault models
- Testing based on fault models
- Discussion

# Discussion

- Various tools do similar things – but for general faults
  - Test case derivation helps rule out false positives
- Fault injection not a new idea
- Fault models available –
  code reading the more efficient approach?
- How much process, how much technology?
- How to build and maintain a good fault data base? Agility?
- Fault-based testing needs to be complemented

# (Deliberate) Limitations

```
. . .
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(...);
. . .
```

# Wrap-Up and What‘s in it for you?

- „Good“ test cases require fault models
- Coverage not based on fault model
- „Fault models“ non-trivial
  - But everybody uses them all the time!
- Fault model needs to be applicable …
- … but not finding a problem doesn‘t make tests bad!!
- Operationalization: tests and check lists

- Continue to build a culture of faults!

# References

[Andrews et al. 2005] James H. Andrews, Lionel C. Briand, Yvan Labiche: Is mutation an appropriate tool for testing experiments? ICSE 2005: 402-41

[Binder 1999] Binder, Testing OO Systems, Addison Wesley, 1999

[Büchler et al. 2012] Büchler, Oudinet, Pretschner: Semi-Automatic Security Testing of Web Applications from a Secure Model. Proc. 6th IEEE Intl. Conf. on Software Security and Reliability, pp. 253-262, June 2012

[Gutjahr99] ] Gutjahr: Partition Testing vs. Random Testing: The Influence of Uncertainty. IEEE TSE 25(5):661-674, 1999

[Inozemtseva&Reid'14] Inozemtseva, Holmes: Coverage is not strongly correlated with test suite effectiveness. To appear in Proc. ICSE, 2014

[Lu et al.08] Shan Lu, Soyeon Park, Eunsoo Seo and Yuanyuan Zhou : Learning from Mistakes – A Comprehensive Study on Real World Concurrency Bug Characteristics. Proceedings of the 13th international conference on Architectural support for programming languages and operating systems, pp. 329-339, 2008

[Ma et al. 2002] Yu-Seung Ma, Yong Rae Kwon, Jeff Offutt: Inter-Class Mutation Operators for Java. ISSRE 2002: 352-366

[Matinnejad et al. 2013] Reza Matinnejad, Shiva Nejati, Lionel C. Briand, Thomas Bruckmann, Claude Poull: Automated Model-in-the-Loop Testing of Continuous Controllers Using Search. SSBSE 2013: 141-157

[Morell 90] Morell: A Theory of Fault-Based Testing. IEEE TSE 16(8):844-857, 1990

[Offutt 1992] A. Jefferson Offutt: Investigations of the Software Testing Coupling Effect. ACM Trans. Softw. Eng. Methodol. 1(1): 5-20 (1992)

[Pretschner et al. 2013] Pretschner, Holling, Eschbach, Gemmar: A Generic Fault Model for Quality Assurance. Proc. MODELS, pp. 87-103, 2013. Contains a few references to empirical studies on coverage.

[Weyuker&Jeng91] Weyuker, Jeng:Analyzing Partition Testing Strategies. IEEE TSE 17(7):703-711, 1991