Verification of the C++ Operating System RODOS in Context of a Small Satellite

R. Gerlich[1], R. Gerlich[1],
Sergio Montenegro[2], Frank Flederer[2], Erik Dilger[2],
Merlin Barschke[3], Karsten Gordon[3]

The Second Workshop on Computer Architectures in Space (CompSpace'18)
April10, 2018, Braunschweig, Germany

[1] Dr. Rainer Gerlich BSSE System and Software Engineering
Immenstaad, Germany
E-Mail: Rainer.Gerlich@bsse.biz
Ralf.Gerlich@bsse.biz

[2] Julius-Maximilians-University, Informatik VII
Wuerzburg, Germany
E-Mail: sergio.montenegro@uni-wuerzburg.de
frank.flederer@uni-wuerzburg.de
erik.dilger@uni-wuerzburg.de

[3] Institute for Aeronautics and Astronautics
Technische Universität Berlin, Germany
E-Mail: Merlin.Barschke@tu-berlin.de
Kasten.Gordon@tu-berlin.de

Contents

- Verification Environment
- Verification Strategy
- Results
- Lessons Learned
- Conclusions
- Outlook

Verification Environment

3 Elements

- **TechnoSat:** **Satellite** *TU Berlin*
- **RODOS:** **OS and middleware** *Uni Würzburg*

  subject of verification
- **Verification Tool:** **DCRTT** *BSSE*

  Dynamic C Random Testing Tool

  Robustness Testing on function level

  Extension to communication monitoring

TechnoSat

Orbit	600 km SSO *sun synchronous orbit*
Launch date	July 14th, 2017, Fregat, Baikonur
Design lifetime	1 year
Spacecraft mass	20 kg
Spacecraft volume	465 x 465 x 305 mm
Attitude sensors	IC magnetometers, Sun sensors MEMS gyroscopes, Fiber optic rate sensors
Attitude actuators	Torque rods
Payloads	• a fluid dynamic actuator (FDA) • an S-band transmitter (HISPICO) • fourteen laser ranging retro reflectors • a particle detector (SOLID) • a star tracker (STELLA) • a reaction wheels system with four wheels • a CMOS camera
Processor	ARM® Cortex®-M4-based MCU

- In-orbit demonstration of novel nano-satellite technology
- based on TUBiX20 platform for LEO missions
- key design considerations: modularity, reuse, dependability

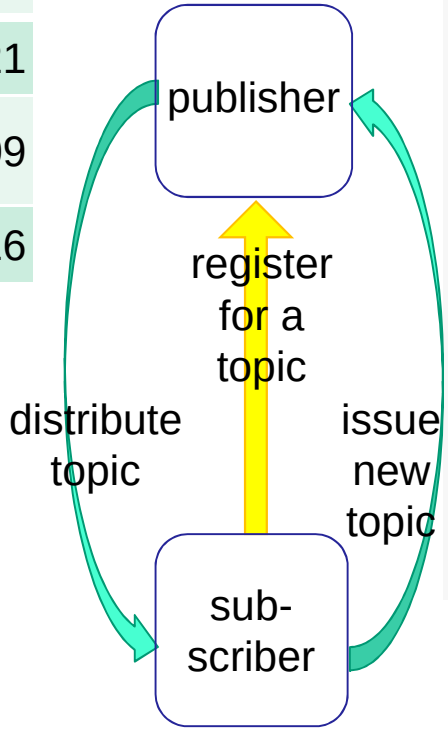
RODOS

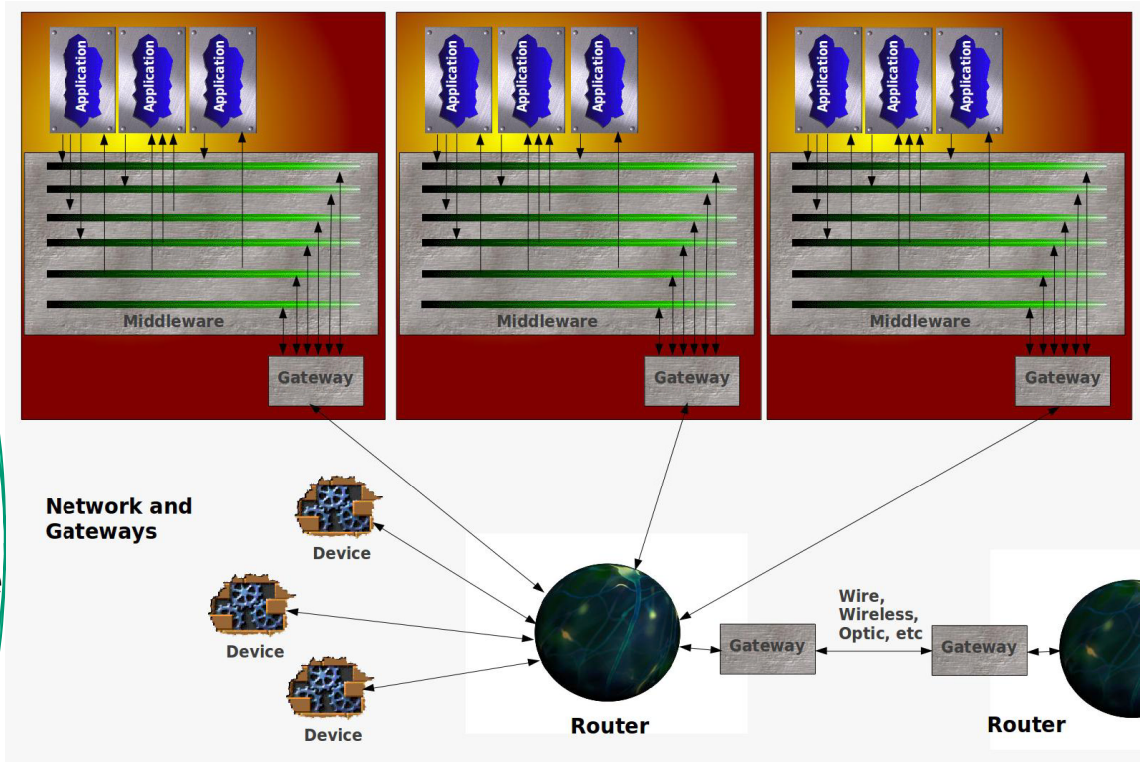
- Middleware and Real-Time OS, integrated
- Framework and Object-oriented Approach, C++
- Publisher – Subscriber Concept, Process distribution, network support
- Subscription based on topics (message types)
- Hardware Abstraction Layer, Support for a number of platforms

- verified-by-use

Subscriber	42
Telecmd-Destinations	21
Comb. Destinations / Telecommands	109
Topics	16

	h	cpp	Total
Files	247	102	349
KLOC	33	15	48
KLines	60	22	82
classes		142	
functions		658	

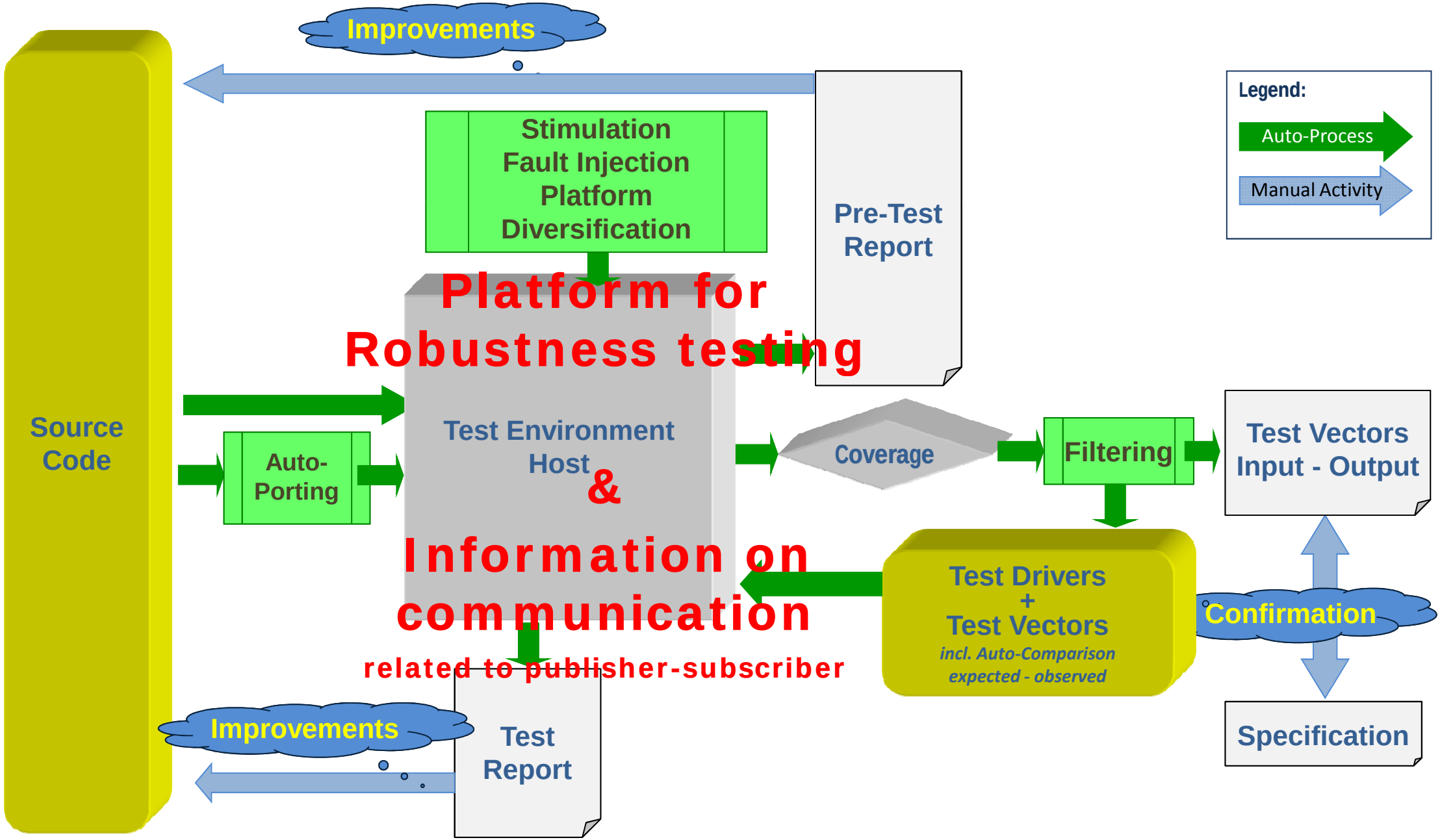




DCRTT

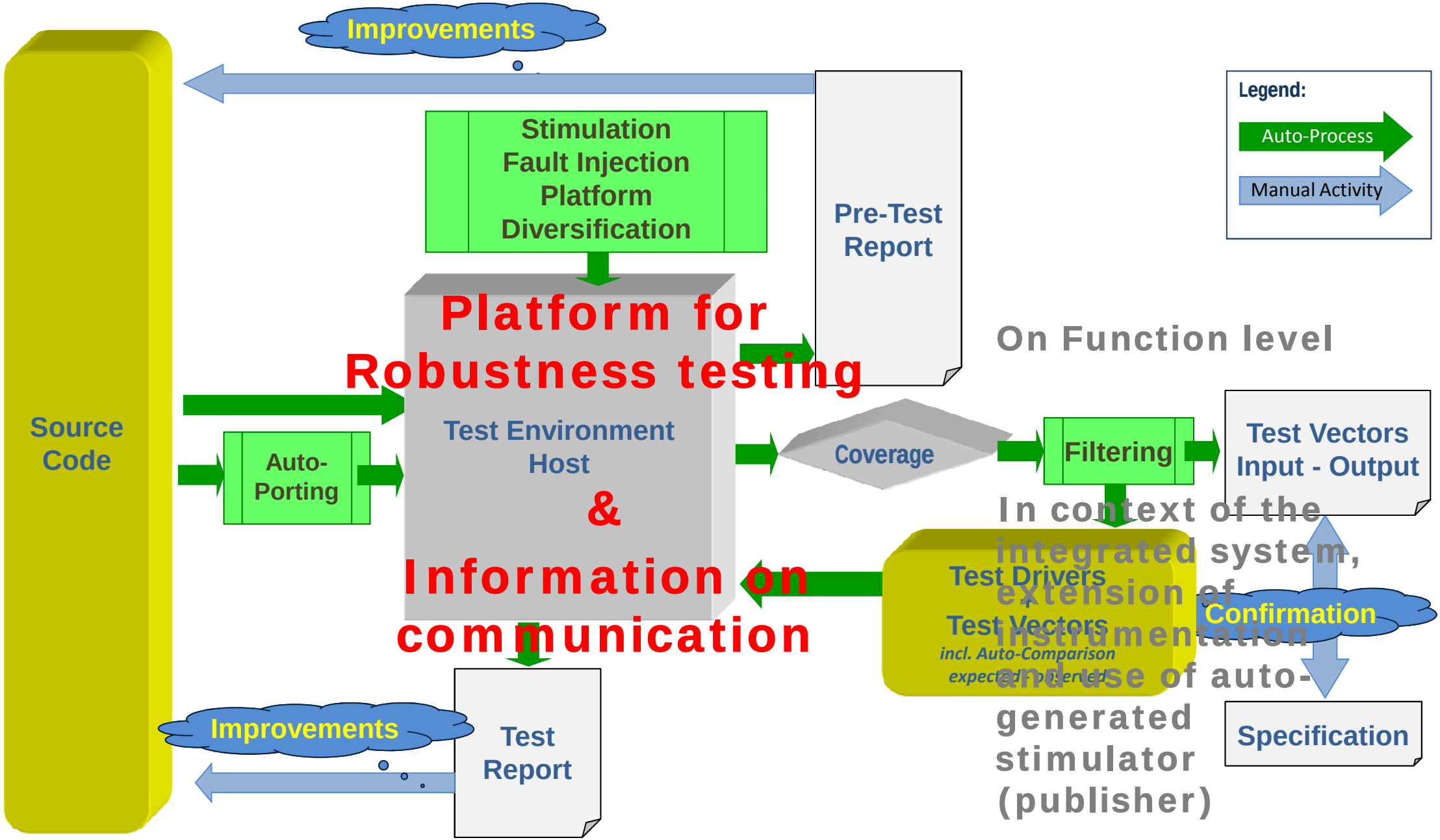
BSSE System and Software Engineering

Automating testing from test data generation to result evaluation



DCRTT

Feature used for verification



DCRTT Support and Interfaces

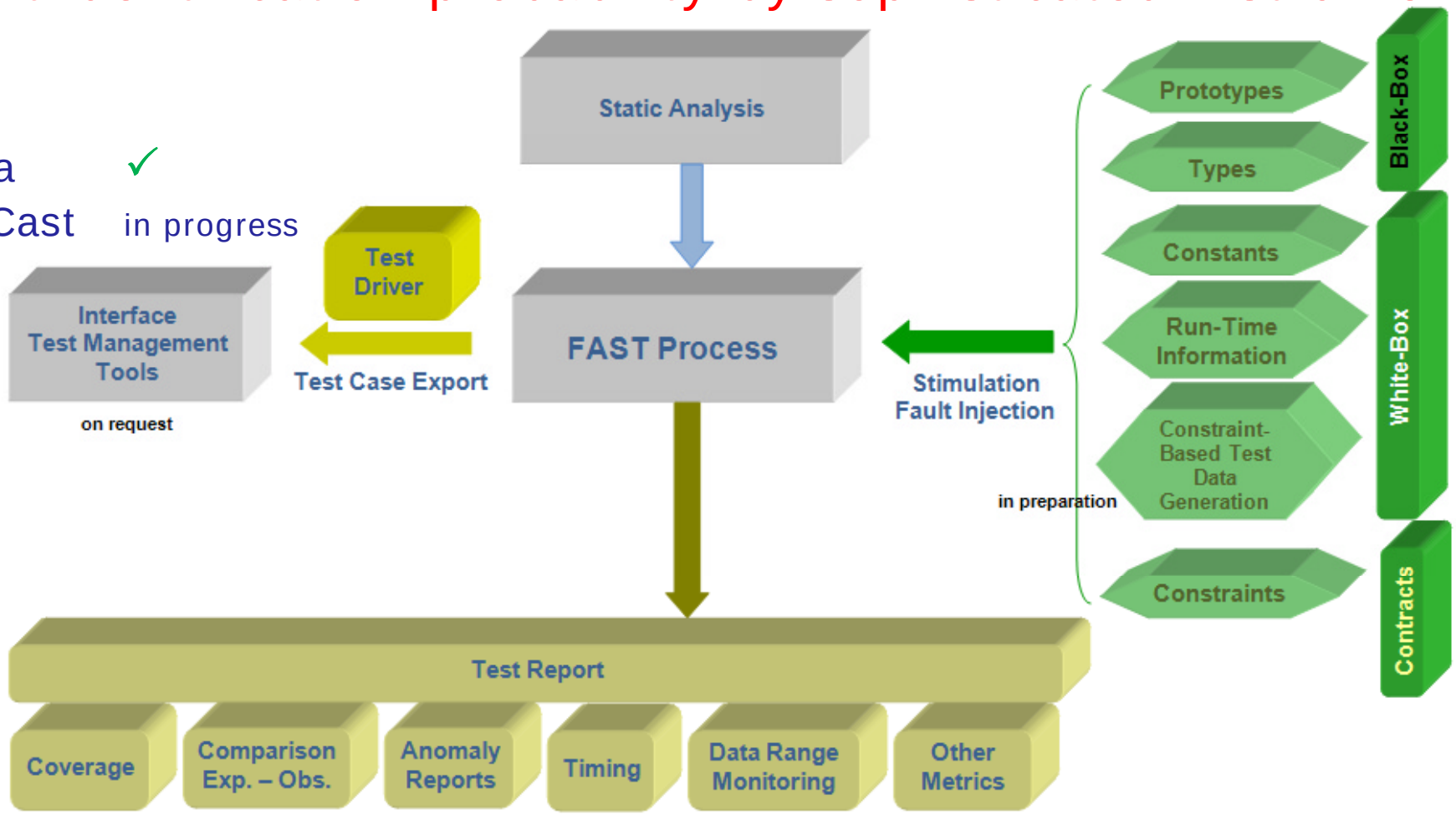
RAISE

fault activation probability by MASSIVE stimulation

fault identification probability by sophisticated instrumentation

Cantata ✓

VectorCast in progress



FAST = **F**low-optimized **A**utomated **S**ource-code-based **T**esting based on DCRTT

Verification Strategy

- **Identification of fault potential**

  robustness testing on function level

  verified-by-use: how many and which findings can we expect in addition?

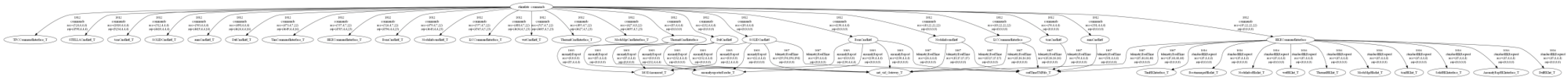
- **Publisher-Subscriber messaging scheme**
  - existing instrumentation extended towards recording of performance figures and communication topology
  - external stimulator (publisher),

    automatically generated from csv-file for telecommands
- **Two Iterations**
  - early version to give an immediate feedback for further development
  - late version for a final check

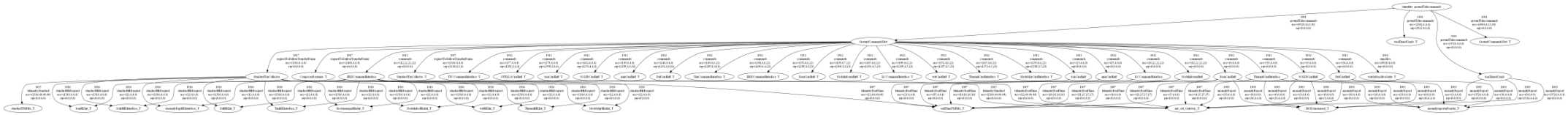
Complexity of Message Exchange

BSSE System and Software Engineering

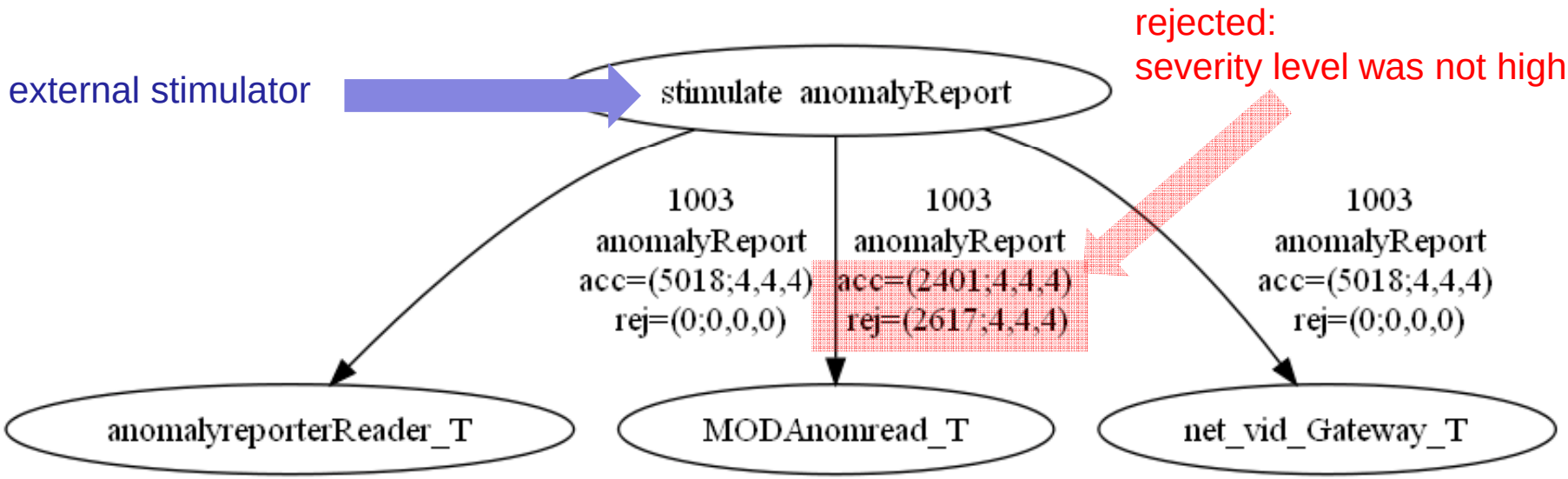
Group (topic) commands



Group (topic) ground telecommands



Group (topic) anomaly report



Example on Robustness Testing

BSSE System and Software Engineering

Fault Potential of Mix signed - unsigned

```
unsigned int len=MIN((int)maxLen,getLenDest(index));
             len=MIN(len,getLenSrc(source));


             memcpy(dest[index],source,len);
```

Edge case: index is invalid
⇒ getLenDest returns -1

len ⇐ (unsigned int) -1
= 2^{32} - 1
len ⇐ getLenSrc
impact by
maxLen and getLenDest is masked

valid only if

len(dest) ≤ len(source)

Guarantee

that minimum of source and destination is copied only

is no longer valid if a wrong *index* is passed to getLenDest

Intention ☺

on safe programming by computing the minimum of lengths is **compromised** by missing fault handling on return value of getLenDest ☹

(which is declared as *signed in*t and may return a negative value)

possibly insufficient knowledge on getLenDest interface

Lessons Learned (non-exhaustive)

- **Language Issue (C and C++)**
  - int abs(int)
  - edge case: -2^{31} abs returns negative value
  - must be known for complete (safe) fault handling

Fault Potential
(even if very low)

```
void myFunc(int val) {
  int valPos;
  valPos=abs(val);
  if (valPos >= limit)
  {<error handling>}
  else { 💣 }
}
```

edge case

Safe
(in any case)

```
void myFunc(int val) {
  int valPos;     not too complex
  if (val <= -limLow || val >= limHigh)
  { <error handling> }
  else {
    valPos=abs(val);
  }    safe in all cases by defensive programming
}
```

reduces number of issues and analysis effort ☺

Edge Cases

- **Incomplete fault handling**
- **Index-out-of range (±1), insufficient length**
- **Mix signed - unsigned**

Analysis

in given context *index* is always valid

or at least no destructive impact

However

if context is changed: reasonable probability that fault may be activated

Would you fix this issue?
Would you avoid it?

Verified-by-use

feeling that it is ok

Robustness testing

highlights fault potential

analysis provides feedback that / when it is ok

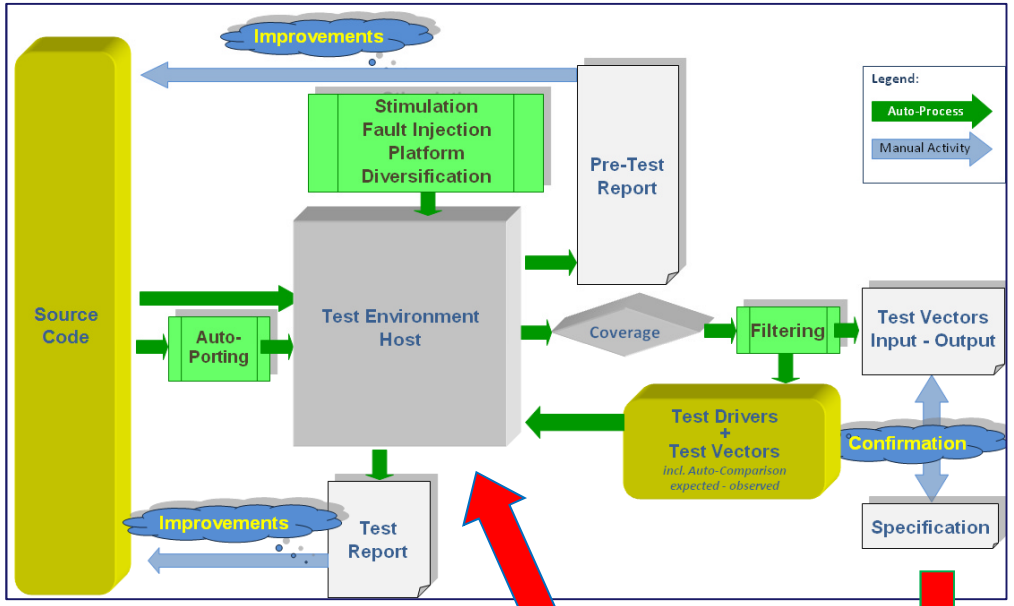
Analysis required to identify the fault potential!

Analysis effort can be reduced by feedback on critical constructs!

Conclusions

- **Robustness Testing**
  - information on fault potential and fault activation conditions
  - Information on how constructs with fault potential can be made **fully safe** instead of being **sporadically unsafe**
  - analysis effort is driven by number of raised issues on fault potential
- **Publisher-Subscriber messaging scheme**
  - Information on topology of message exchange provided
  - correctness and completeness confirmed
  - analysis: no big overhead by publisher-subscriber broadcasting
- **Recommendation**
  - early feedback on development by code analysis driven by tools
  - development must meet verification constraints, if not effort increases
  - early trade-off required: verified-by-use is sufficient or more is required: *think about in advance*

Outlook: Adding Requirements-based Testing (RQBT)



Test Cases + Counter Examples

Functions

Requirements

auto-correlation

Oracle

Pre-Condition | Post-Condition

Function Call

Oracle for unchanged check

```
p1==c? pInput==p
```

Oracle for int abs(int xAbs)

```
FORALL(xAbs)? ret>=0
```

Will fail! Why?

Oracle for inverse function of $f(xSq)=xSq^2$

```
FORALL(xSq)? (sqrt(ret)-fabs(xSq))<eps
```

Wrong! Why?

Check on relative deviation needed!

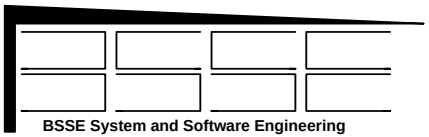
Two checks needed due to division by xSq

Even then check will fail, when and why?

Oracle for Status Monitoring

```
status==active && mode==mode1 ? moniFlag==true
status==active && mode==mode2 ? moniFlag==true
status==active && mode==mode3 ? moniFlag==false
```

The Team



Rainer Gerlich **Ralf Gerlich**

Sergio Montenegro **Erik Dilger** **Frank Flederer**

Merlin Barschke **Karsten Gordon**

The TechnoSat project was funded by DLR Space Administration on behalf of the German Ministry of Economics and Energy, BMWi under Contract No. 50 RM 1219

Thank you for your attention!

Questions?

Backup

Analysis Approach

BSSE System and Software Engineering

- **Robustness Testing**
  - issue-driven analysis / review, issues as resported by DCRTT
  - guided by issued reports on fault potential
- **Publisher-subscriber message exchange scheme**
  - driven by recorded figures and derived topology
  - check of the telecommand chain
  - check of message distribution and processing
  - built-in cross-checks on verification issues (non-exhaustive list)

Edge Case


```
unsigned int len=MIN((int)maxLen,getLenDest(index));

             len=MIN(len,getLenSrc(source));

            memcpy(dest[index],source,len);
```

Analysis

in given context *index* is always valid

However

if context is changed: reasonable probability that fault may be activated

Verified-by-use

feeling that it is ok

Robustness testing

highlights fault potential

analysis provides feedback that / when it is ok

Would you fix this issue?
Would you avoid it?

Analysis required to identify the fault potential!

Analysis effort can be reduced by feedback on critical constructs!

Lessons Learned (non-exhaustive) 2/2

- **Language Issue (C++)**
  - Initialization sequence of objects is undefined amongst compilation units
  - RODOS (or application) needs to manage it itself from main
  - undeterminism enforces deviation from O-O best practices
  - compromises role of constructors
  - objects are in an undefined state after creation
  - Issue for auto-testing, auto-identification of init-functions

- **Examples**
  - hardware drivers: initialization needed before objects using them
  - similarly for memory allocation, not released

Clarification of Terms

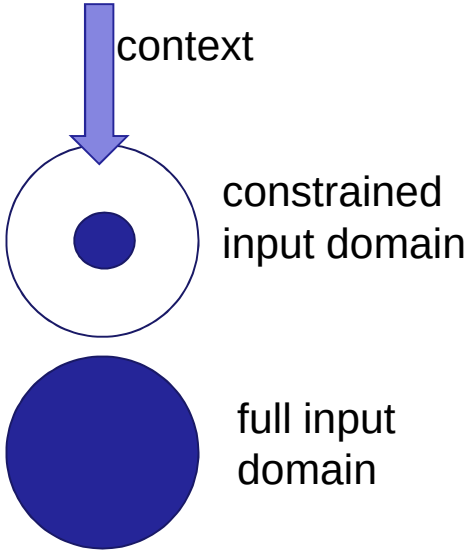
- **(Code) Coverage**
  - block coverage: sequence of statements, all executed together, except: exceptions
  - decision coverage: logical expression, true or false, both are considered
- **Fault Injection**
  - exposure to invalid conditions to check behaviour under these conditions
  - decision cover
- **Fault Assessment**
  - classification of a fault
- **Context**
  - Contraining condition for fault assessment

Fault Assessment and Context

Term	Scope
Fault	Mistake in code
⇓	⇓
Error	Bad state of a system
⇓	⇓
Failure	Unexpected observed behaviour

		Code	
		Defect present	*Defect NOT present*
Result	*Defect Reported*	true positive	false positive
	Defect NOT reported	false negative	true negative

with context	Input domain may be constrained by callers
without context	Maximum input domain can be used



Analysis Approach

BSSE System and Software Engineering

- **Robustness Testing**
  - issue-driven analysis / review, issues as resported by DCRTT
  - guided by issued reports on fault potential
- **Publisher-subscriber message exchange scheme**
  - driven by recorded figures and derived topology
  - check of the telecommand chain
  - check of message distribution and processing
  - built-in cross-checks on verification issues (non-exhaustive list)
- **Examples**
  - publisher-subscriber: broadcasting to all registered subscribers
  - What is the overhead due to broadcasting?
  - accepted / rejected messages, reason for rejection
  - every message is accepted once at least
  - every telecommand is accepted by one subscriber at least
  - no exception is raised