FAST

Assessment and Evaluation of Fully Automated Source-code-based Testing Strategies

Data Systems in Aerospace DASIA 2013

May 16th, 2013, Porto, Portugal

ESA Contract No. 4000102645

BSSE Team: Rainer Gerlich, Ralf Gerlich, Thomas Boll

DNV Team: Bengt Solheimdal Johansen, Kenneth Kvinnesland

ESA Technical Officer: Marek Prochazka

Dr. Rainer Gerlich BSSE
Rainer.Gerlich@bsse.biz
Ralf.Gerlich@bsse.biz

ESA/ESTEC
Marek.Prochazka@esa.int

Det Norske Veritas AS (DNV)
Kenneth.Kvinnesland@dnv.com
Bengt.Solheimdal.Johansen@dnv.com

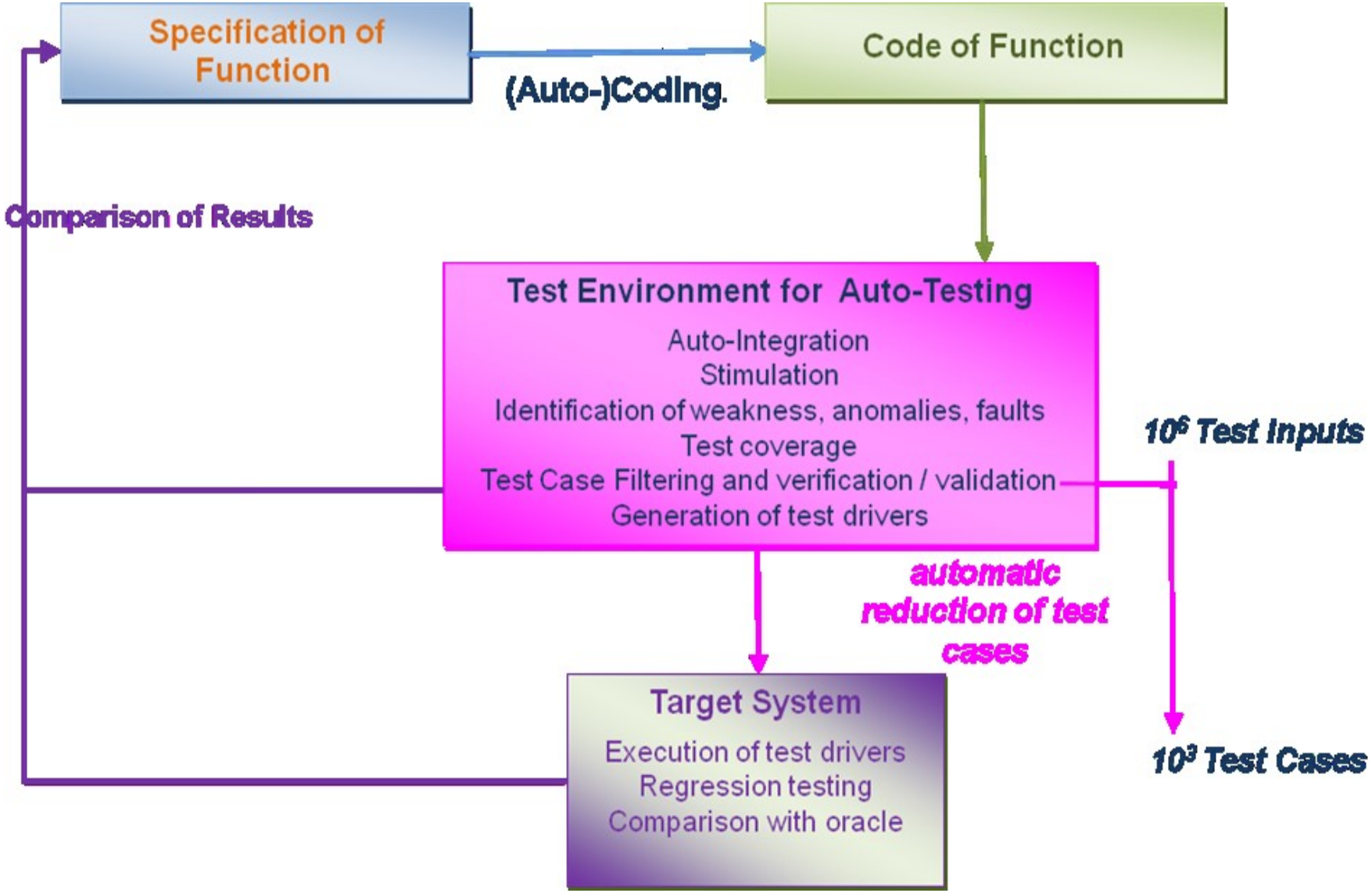
Contents

- The FAST Process

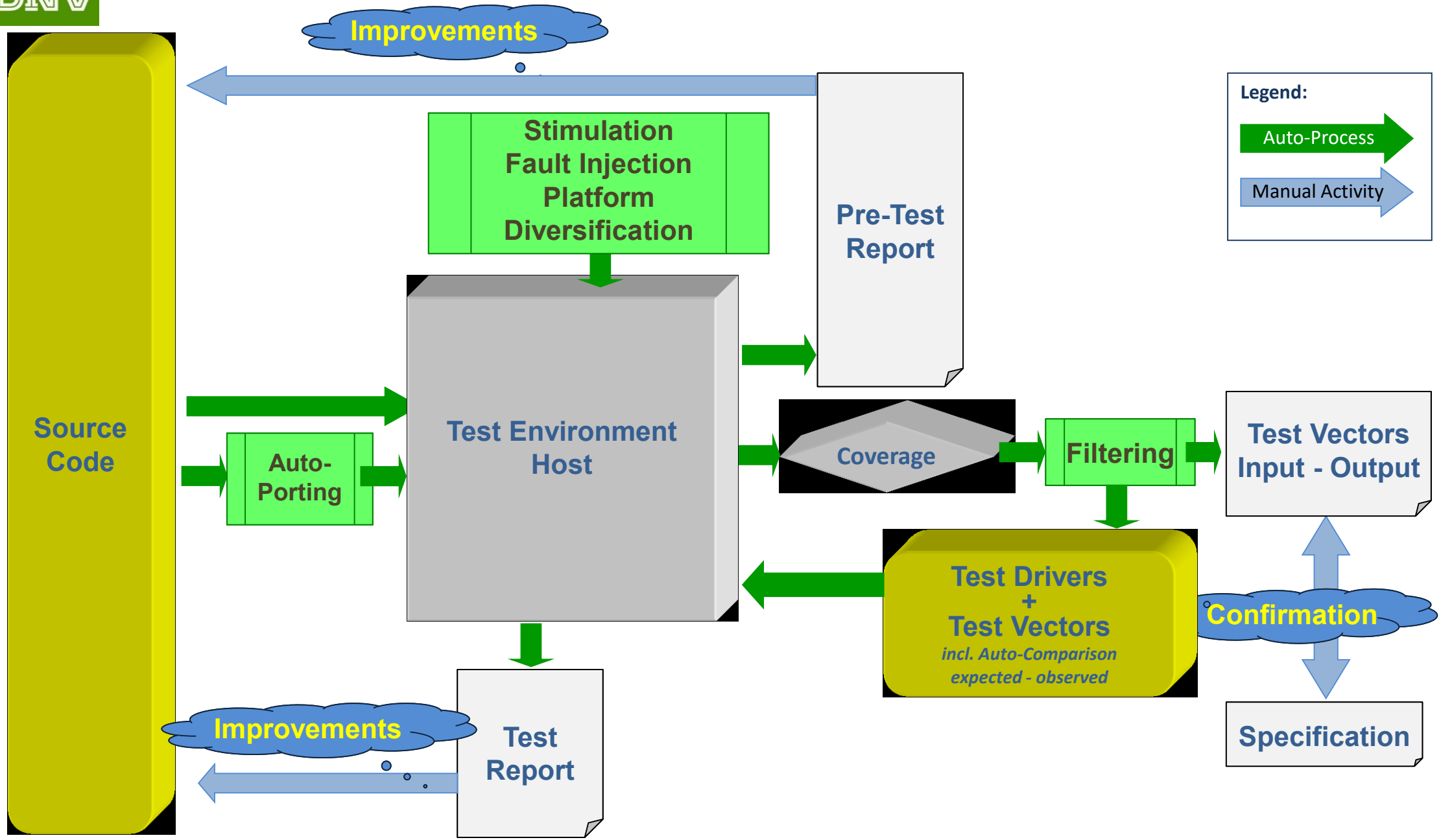
  Fully / Flow-optimised Automated, Source-code-based Testing
- Verification Challenges and Findings
- Conclusions

The FAST Process

The FAST Test Process
Principal Flow



The FAST Test Process
Detailed Flow



The Application Software

The Software

- **Earth Observation Satellite**
  assumed not to contain any further critical defects
  tested according to ECSS E-40 and Q-80, version B and ISVV Guide
- **Characteristics**
  - Set 1: mission-critical (subset)

    anomalous behaviour would cause or contribute to a failure of the satellite system resulting in permanent and/or non-recoverable loss of the satellite's capability to perform its planned mission
    - >1500 functions
    - 65 KLOC
  - Set 2: full set
    - >3000 functions,
    - 165 KLOC

Verification Challenges and Findings

Findings

Category	Description		#Findings	Status
I	Findings directly related to observation of an anomaly / analysis required		2	update requested
			12	non-critical, either in the current version or in general
II	Findings identified during analysis of a reported anomaly		11	
III	Findings identified due to use of different utilities (diversification) / no analysis required	DCRTT support utilities	16	
		Compiler/Linker	11	
IV	Findings identified by comparison of expected and observed values		(n/a)	
Total			52	

Finding Examples

```
#define QUEUE_SIZE          10
#define BUFFER_SIZE         1024

typedef struct TyQueue {
  unsigned int start;
  unsigned int len;
} TyQueue;

unsigned int head=0;
unsigned int tail=0;
unsigned int next=0;
bool         full=FALSE;
TyQueue      queue[QUEUE];
char         *buffer[BUFFER_SIZE];

void storeQueue(char *buff, unsigned int len){
  if (head != tail || full==TRUE)
        next = queue[head-1].start +
               queue[head-1].len;
  if ((next + len) < BUFFER_SIZE){
    queue[head].start=next;
    queue[head].len  =len;
    memcpy(buffer+next,buff,len);
    head++;
    if (head >= QUEUE_SIZE)
      head=0;
    if (head == tail)
      full=TRUE;
  }
}
```

Initially:

head == tail == 0 && full == FALSE

Next steps:

head > 0

Wrap-around:

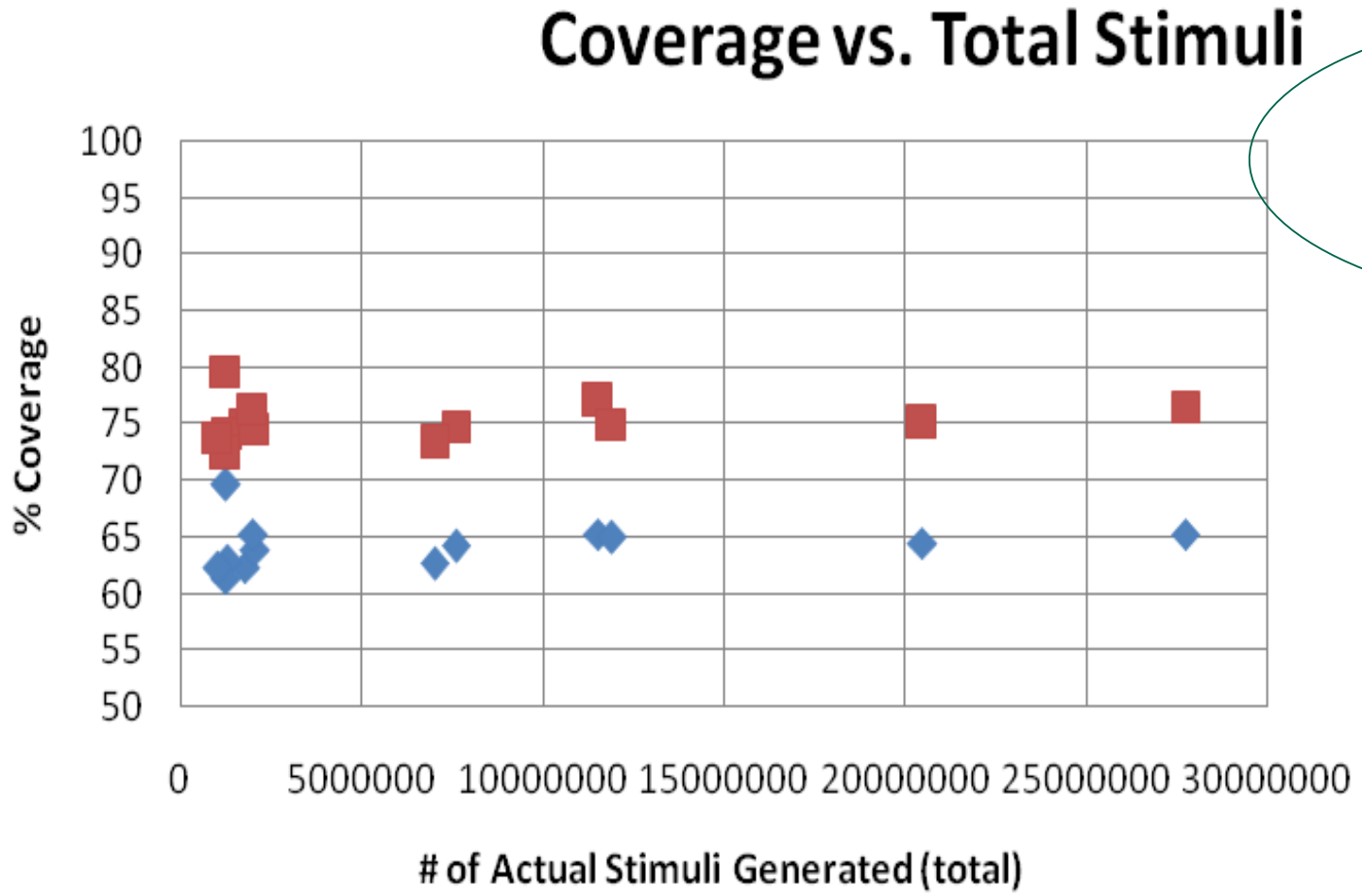
head == 0 && full == TRUE

while possibly still tail == 0

access intended to

`queue[QUEUE_SIZE-1]`

Coverage
Stimuli Dependency



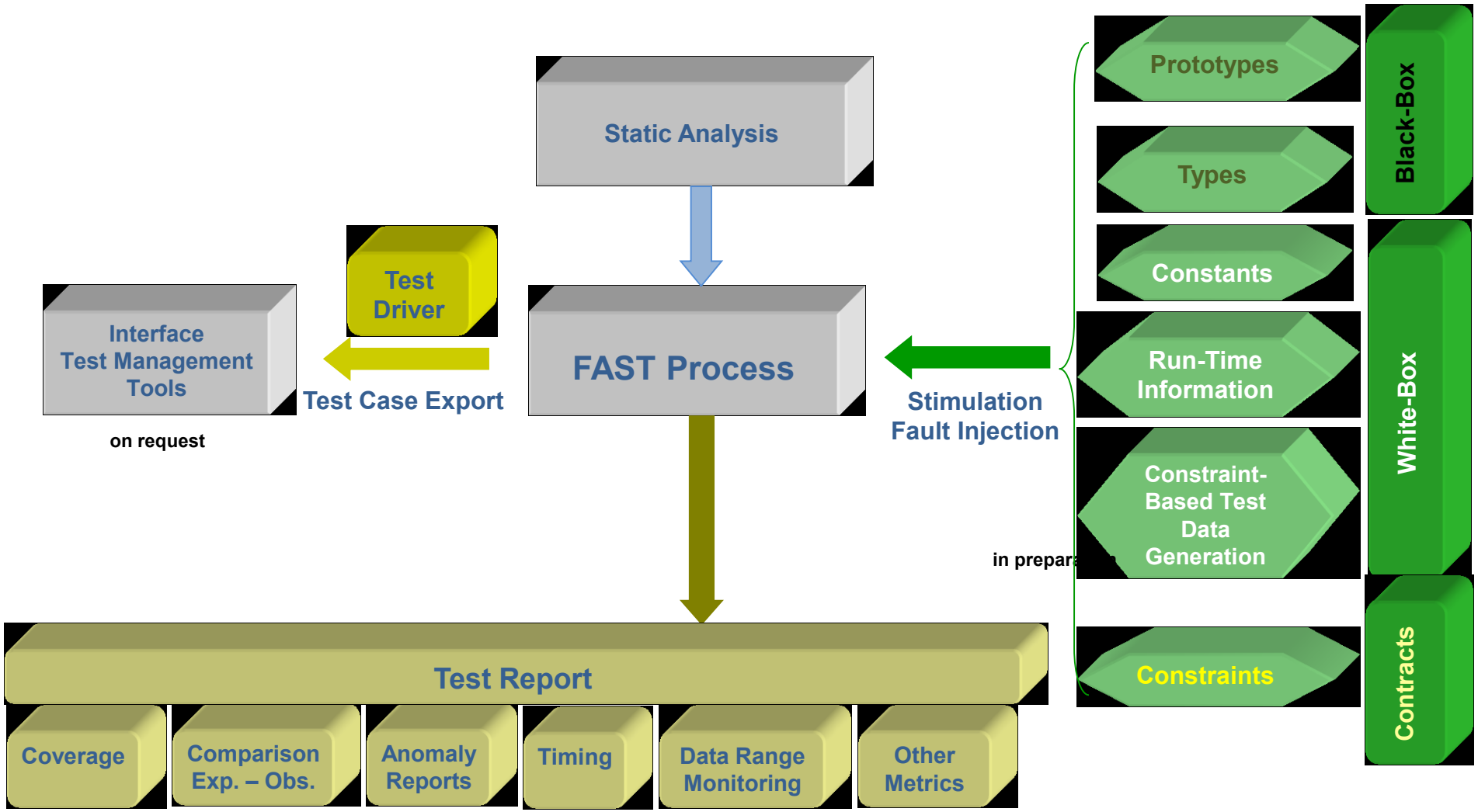
A block is a set of *statement sequences* which are executed one after the other once the first statement has been executed.

A small number is sufficient to start with anomaly analysis

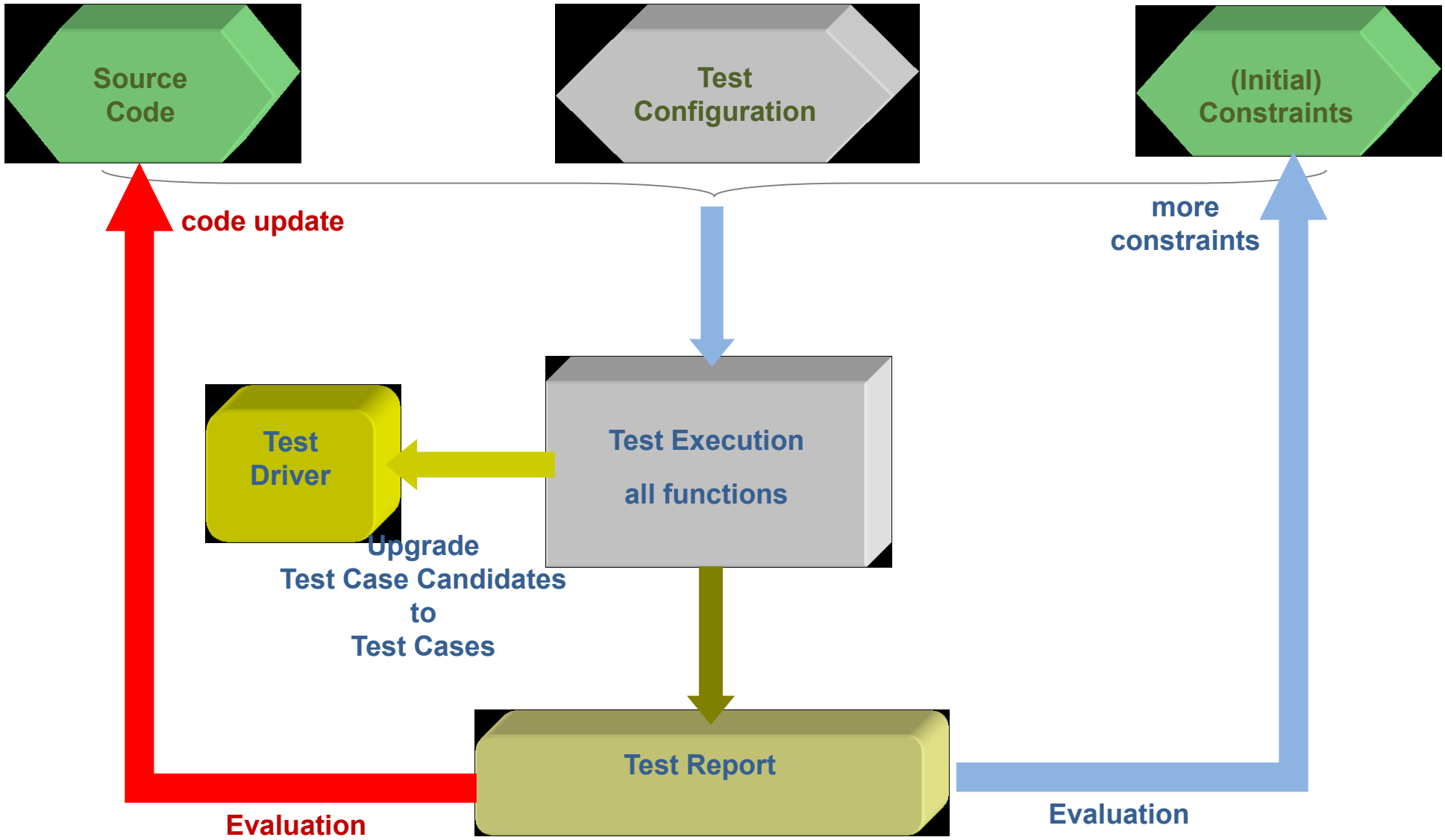
Moderate increase of coverage with number of stimuli

Tool Interfaces

Interfaces of the FAST Test Process



From Test Preparation to Result Evaluation



Conclusions

Limitations

- **L1**
  - Since the theoretical number of input combinations may be very large, the tool may not select the particular input combination necessary to exercise specific parts.
- **L2**
  - Coverage of requirements is not considered in test data generation
  - Additional test cases typically need to be added manually
- **L3**
  - Unknown “Design by Contract” may lead to false positives
  - Have to be manually removed by introduction of (few) constraints
- **L4**
  - Tool not yet qualified
  - Interface to qualified test management tools would remove this limitation

Advantages

- **A1**
  - Automatically generated unit test suite providing high structural coverage
  - Amount of coverage depends on code structure
- **A2**
  - Large amount of stimuli exposes S/W to inputs normally not considered
  - Reduced impact of engineer's bias, additional validation element
- **A3**
  - Assumed to increase path coverage over usual coverage requirements
  - Measurement of basis path coverage in tool available
- **A4**
  - Multiple levels of software are tested together instead of test in isolation
  - No classical integration test, but useful in finding problems
- **A5**
  - Can reveal complex programming style problems
  - Can find defects that cannot be found by rule-based static analysers

Use in the Software Suppliers Test Process

- Apply static analysis tool to remove as many poor coding practices as possible
- Apply DCRTT tool
- Investigate issues reported by DCRTT as potential errors
- Review auto-generated test suite using objectives from e.g. ECSS E-40 or DO178 regarding:
  - requirement coverage,
  - additional robustness issues.
  - This should give you the structural coverage, as well.
- Add test cases manually based on the output from the review

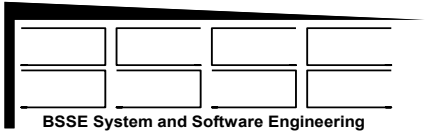
Use in ISVV

- Apply DCRTT tool as an alternative or in addition to static analysers.
- It's important to be as complementary to the supplier's process as possible.

Acknowledgement

The FAST team would like to thank

- the ESA project which provided sources of its flight software,
- SciSys for providing QERx,
- EADS Astrium for providing the RTEMS Product,
- the German and Norwegian national space agencies for funding the activity.



Thank you for your attention!

Questions?

Finding Examples

file1.h

```
typedef struct TyCmdDescr{
        unsigned int     cmd;
        unsigned int     minLen;
        unsigned int     maxLen;
} TyCmdDescr;
TyCmdDescr cmdDescr[ ];
```

file1.c

```
#include "file1.h"
#include "file2.h"
TyCmdDescr cmdDescr[ ]={{cmd1,3,10}, ... };
void recvCmd(char *cmdData, unsigned int offset, unsigned int entry) {
        unsigned int len;
        memcpy(&len,cmdData+offset,sizeof(unsigned int));
        if (len >= cmdDescr[entry].minLen)
                execCmd(cmdData,offset);
}
```

check on length

file2.h

```
void execCmd(char *cmdData, unsigned int offset);
```

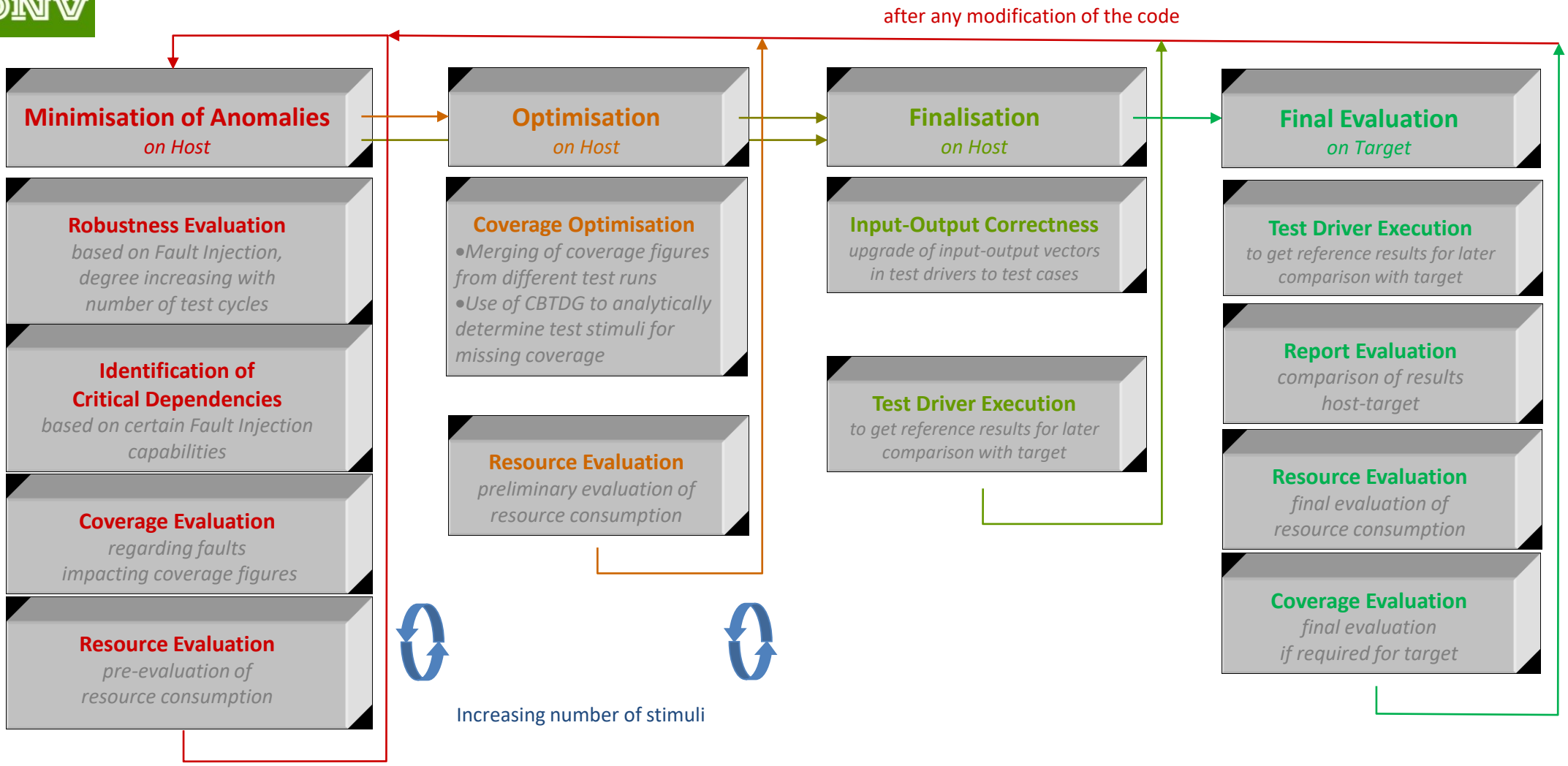
file2.c

```
#include "file2.h"
char buffer[10000];
void execCmd(char *cmdData, unsigned int offset){
        unsigned int len;
        memcpy(&len,cmdData+offset,sizeof(unsigned int));
        memcpy(buffer,cmdData+offset+ sizeof(unsigned int),len-3);
}
```

implicit dependency between two different, independent entities

4 GB corruption !

The FAST Test Process
Flow of Test Steps



Iterations over FI modes and modifications of the code

Iterations to improve coverage and resource consumption requiring modifications of the code

Iterations to achieve compliance between specification and implementation requiring modifications of the code

Iterations to achieve compliance between specification and implementation requiring modifications of the code

The Project and the Application Software

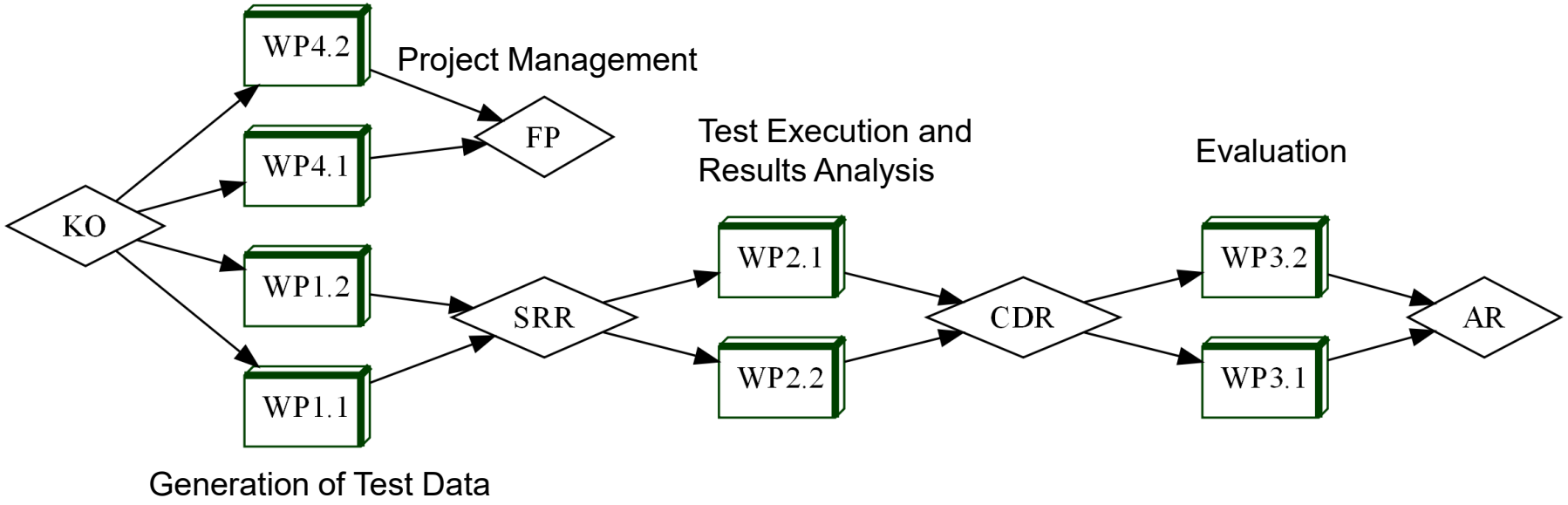
Project Goals

- Objective 1
  Perform an assessment of the fully automated, source-code-based testing (FAST) by applying it on realistic spacecraft flight software.
- Objective 2
  Evaluate applicability and scalability of the approach in the space domain, with focus on efficiency and effectiveness in spacecraft flight software as a whole or some of its subsystems.
- Objective 3
  Evaluate source-code-based testing for fault identification sensitivity, testing coverage, cost efficiency, and usability both in nominal validation and Independent Software Verification and Validation (ISVV).
- Objective 4
  Prepare and disseminate a set of guidelines and recommendations for the automated source-code-based testing process, and put it in the context of the ESA software development process.

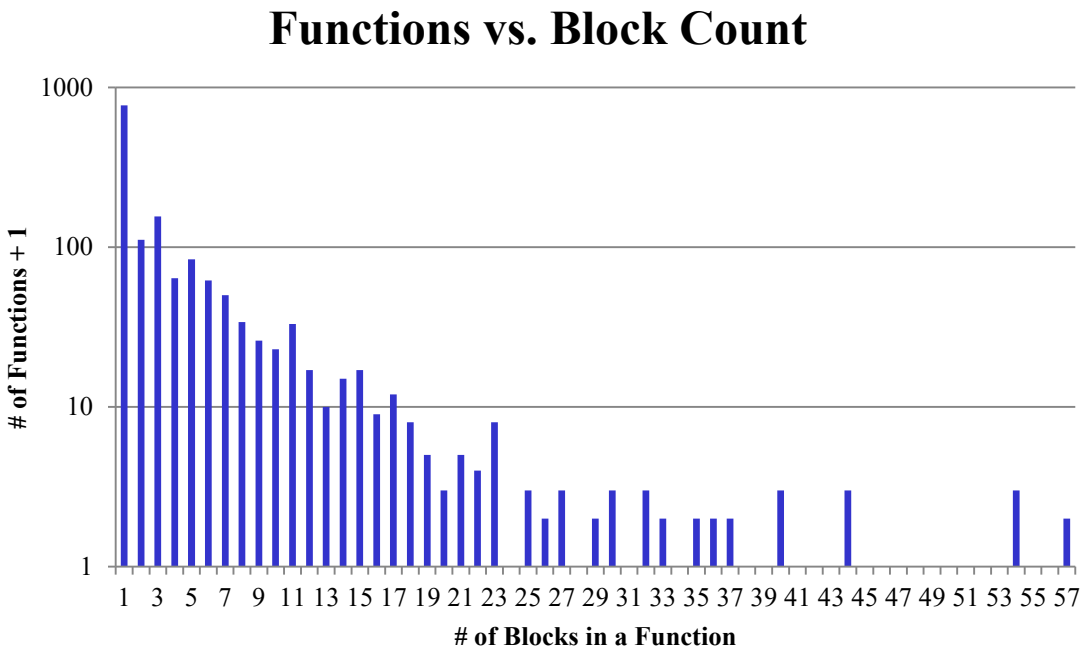
Project Organisation

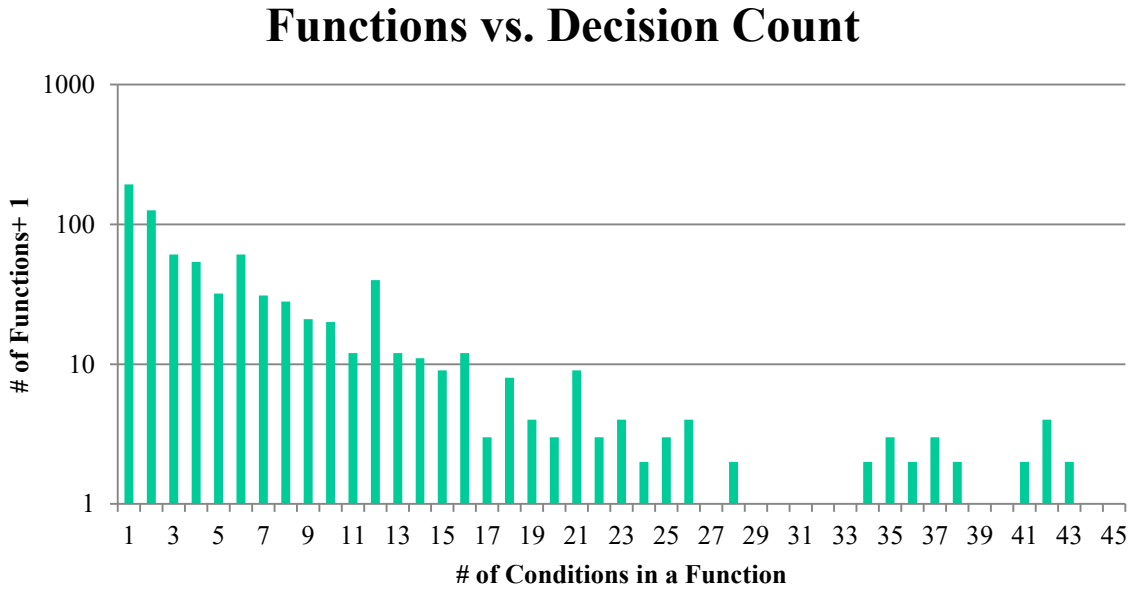
Team

- BSSE, prime
  - experience with the FAST process
  - demonstration of capabilities of BSSE tool “DCRTT”,
- DNV
  - experience in testing and verification
  - experience in ISVV



Histograms on Blocks and Conditions





Verification Issues

Verification Issue	Done	Comment
Robustness	✓	Injection of invalid inputs, modification of return values, stimulation of const-elements, overriding of initialisers
Resource Consumption		Figures were recorded, but not evaluated
Critical Dependencies	✓	
Internal Interfaces	✓	Injection of invalid inputs, modification of return values, stimulation of const-elements, overriding of initialisers
Code – Data Interfaces	✓	Injection of invalid inputs, modification of return values, stimulation of const-elements, overriding of initialisers
Platform Dependencies	✓	Different gcc-compiler versions
Reachability of Code	✓	Deadcode in context of anomaly analysis Coverage figures
Input – Output Correctness		No references available

Finding Examples (1 of 4)

```
memcpy(dest,src,para-3);
```

What if para < 3 ?

Nothing copied at all ?

Nearly 4 GB copied !

High fault potential, if activated !

Check if relevant

```
for (i=0;i<limit;i+=para)
```

What if para == 0 ?

Endless loop !

High fault potential, if activated !

Check whether of relevance !

file1.c

```
#define MYLIT 999
```

file2.c

```
typedef enum{ myLit=999} TyLit;
```

What if value is changed ?

Maintenance !

Inconsistency !

Dependency known ?

Which part needs to be re-tested ?

Finding Examples (3 of 4)

```
int myCopyFunc(const int * const src, const int * const dest, unsigned int len);
library function: body is hidden
int myCopyFunc(const int * const src, const int * const dest, unsigned int len) {
        /* something else */
        memcpy(dest,src,len);
        /* something else */
        return <value>;
}
```

Not issued:passing arg 1 of `memcpy' discards qualifiers from pointer target type

Found by analysis of an anomaly

```
common.h
typedef enum { ... } TyEnum1;
typedef enum { ... } TyEnum2;

prototypes.h:
#include "common.h"
extern void func1(TyEnum1 para1);
extern void func2(TyEnum2 para1);

bodies.c:
#include "common.h"
#include "prototypes.h"

void func1(TyEnum1 para1) { ... }
void func2(TyEnum1 para1) { ... }
```

gcc 3.2.3: no message issued

gcc 3.4.5: message issued

Found by platform diversification / porting

False Alarms Issues

- **What is a false alarm?**
  - violation of a rule without having fault impact?
  - identification of fault potential without having fault impact in the current version?
  - request for confirmation of fulfilment of a pre- or post-condition?
- **How are alarms raised in case of FAST / DCRTT?**
  - by analysis of required conditions for building an executable
  - by raising an anomaly at run-time (exception, abort, timeout)
  - by missing coverage identified at post-run-time
  - by code analysis in context of an observed anomaly
- **Required: clear rules for assessment**
  - How to deal with “design-by-contract”?
  - What are the demands on robustness?

Constraints and Initialisation

```
int myFunc(int ind, int len, char *buffer);
```

```
PARA const 7 =< elem len =< size buffer  ; const 5 =< elem ind < elem len ;
```

Constraint Set **Atomic Constraint** **Atomic Constraint**

```
STRUCT TyMyStruct const 7 =< elem len <= size buffer; const 5 =< elem ind < elem len ;
```

```
typedef struct TyMyStruct {
  int ind;
  int len;
  char *buffer;
} TyMyStruct;

int myFunc(TyMyStruct para1);
```

Constraint Sets	Atomic Constraints	Instances	Reduction
28	52	1328	25.5

Initialisation Patterns

```
cond case   filesToLook   funcsToLook       void     void     filesToTest    funcsToTest
                                           return   paraList
u     c     file1.c       func1initialise    *      void     error_manager.c     *
u     c     file2.c       func2_initialise   *      void     start_up.c          *
c     n        *.c        *init*             *      void       *.c               *
```

Anomaly Identification

Anom. Type	Checkpt. Type	Critical Function	Id Critical Function	Block Id	Cond. Id	Test Id FUT Id	#Events
Excp	Cond	Func1	428	13	1	428	1
Excp	Block	Func2	464	1	none reached	464	51
Excp	Block	Func3	605	0	none reached	607	26
						608	26
						609	26
						610	26
						611	26
						612	26

6 anomalies reported,

but all have the same source 605

Anomaly Identification / Extended – Index Monitoring

Anom. Type	Critical Function	Index Id	Dim. Index	Min/Max Observed	Index Expr.	Test Id FUT Id	#Events
OutOfRange Low	Func1	2523	0	min=-1<0	idx1	232	2284
						236	2284
OutOfRange Low	Func2	2524	0	min=-1<0	idx2-1	232	2284
OutOfRange High	Func3	2844	2	0	idx3	281	2339
						275	2339

2 anomalies reported each for functions with test id 232 and 281

```
Expr     Type    Violation          Function     File
idx1     low      min= -1   < 0     func1        file1.c
idx2-1   low      min= -1   < 0     func1        file1.c
idx3     high     max=965 > 27      func2        file2.c
```

Compressed information:
List on Criticial Indices

Results

Test Runs

- **Host and target platform**

  FAST: most activities on host, test drivers / regression tests on target

- **Sensitivity Analysis for the Process**
  - stimuli per function: 50, 300, 3000, 10000

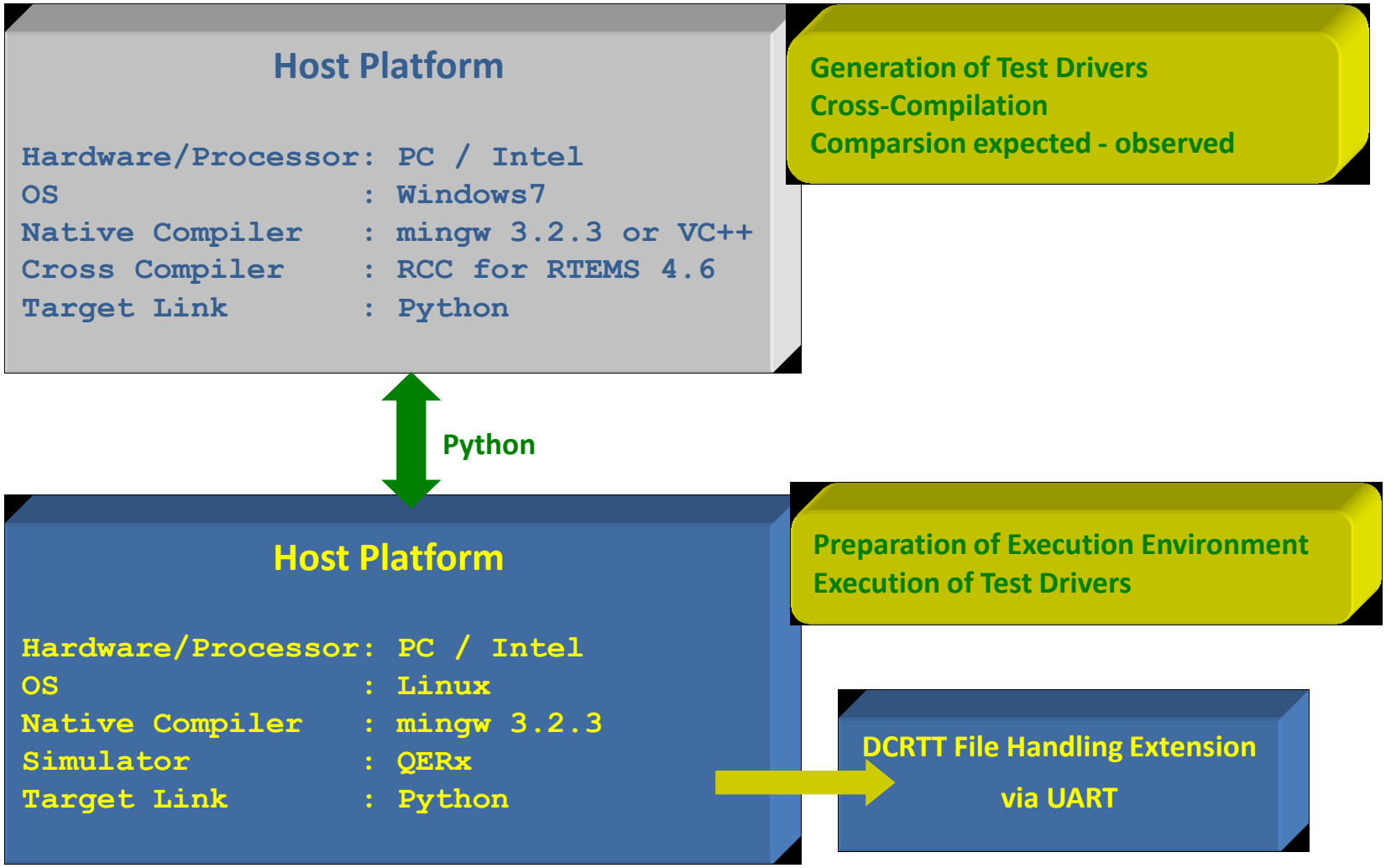
    resulting in a total number of stimuli for all functions

    1 .. 27 millions
  - stimulation of parameters

    function parameters, optional global data directly accessed by a function
  - fault injection
    - invalid stimuli (input), NULL pointer
    - invalid return (modification of return), NULL pointer
    - assign to const-elements
    - blanking (0's) of initialised data

- **Complementing the application software**
  - generation of stubs for missing function bodies

Platforms



Performance / Build

Step	Duration / s	Comment	
Preparation phase	~4000	It covers all analysis activities performed before the first test of a FUT is started	for all 84 files and 1530 functions under test
	150		for two files under test, 21 functions
Test Build Phase	a few seconds	Generation of all files required to run a test for a FUT	
Test of a FUT	45	50 stimuli, one function, including execution of the test drivers on host and target and generation of part of inputs for documentation, of course, this figure depends on a FUT	50 stimuli
	980		1 Mio. stimuli
Complete test run	1900	stimulation, identification of test drivers, test driver execution on host and target, test report generation	1.1 Mio. stimuli, 2 files, 21 functions one of the 21 FUTs took 980 seconds (1 Mio. Stimuli), i.e. 50% of the overall time
	170		200 stimuli, 1 file, 1 function

Performance / Evaluation

Step	Duration / s	Comment
Test Driver Generation	~10 - 60	Time to generate a test driver derived from the total test duration (col. 1 in Tab. 5-6) and the number of suggested TC (col. 5). The time to create the source code is significantly smaller and is in the range of a few seconds only.
Generation of png-files	~1860	8163 png-files for documentation (in preview and full size, 16326 in total) to document structural coverage of 1530 functions (per function and every filtered TC, i.e. differential structural coverage for every test driver)
Test report	~5400	~1½ hour for all supported evaluation facilities, 84 files and 1530 functions under test
	130	2 files, 21 functions

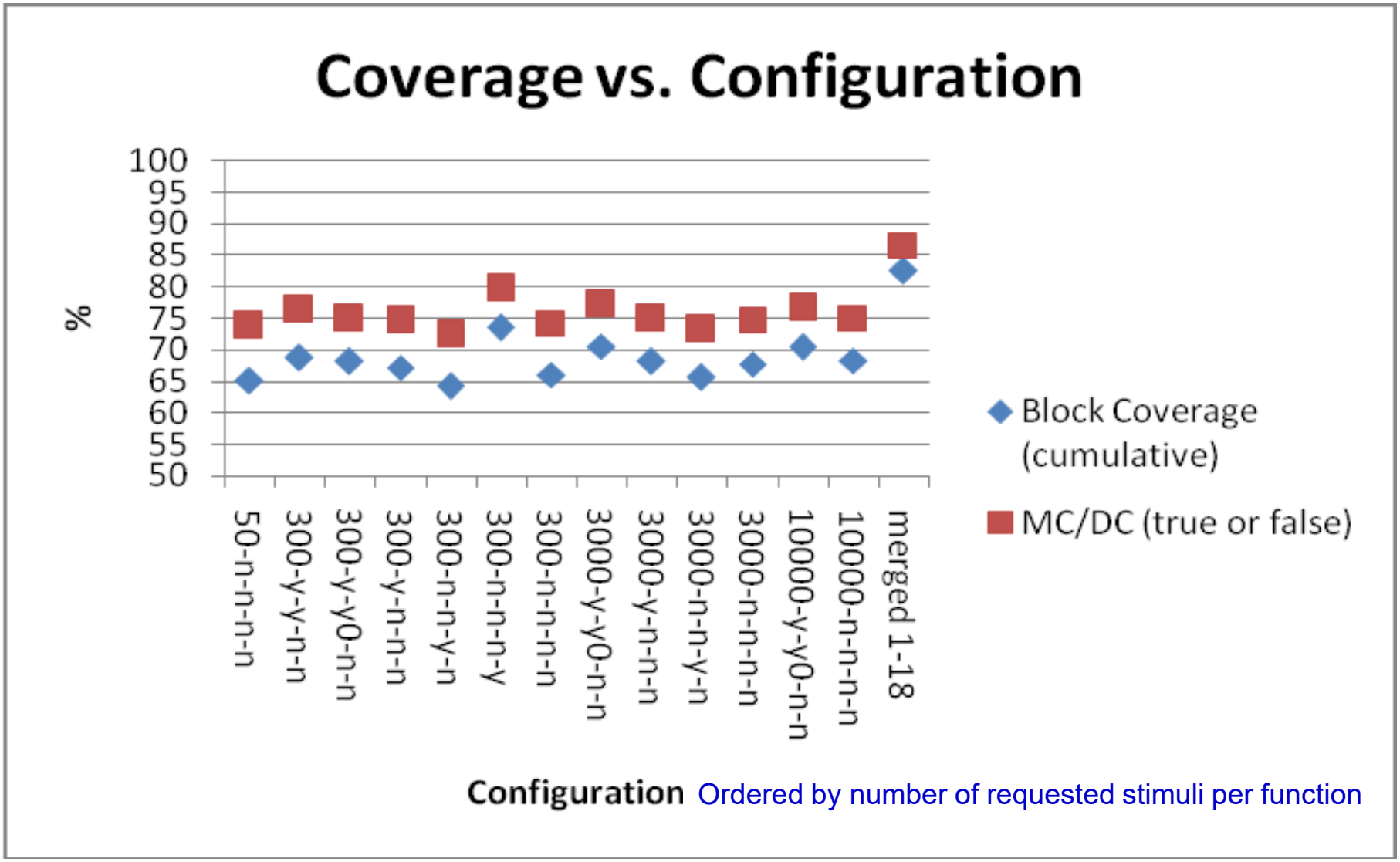
Performance (1530 Functions)

#Injected Stimuli per FUT	# Injected Stimuli in total	Duration / h	Comment
700	1 Mio.	17	without target execution
1200	1.8 Mio	26	
5000	7.6 Mio	41	
13500	20.5 Mio	88	
18000	27.8 Mio	129	
700	1 Mio.	32	with target execution index checking and basic path coverage

Host:
i5-2400@3.1GHz
4 GB RAM
Windows7

Target:
Pentium4@3GHz
3 GB RAM
Linux 2.6

Coverage Configuration Dependency



Coverage, Summary

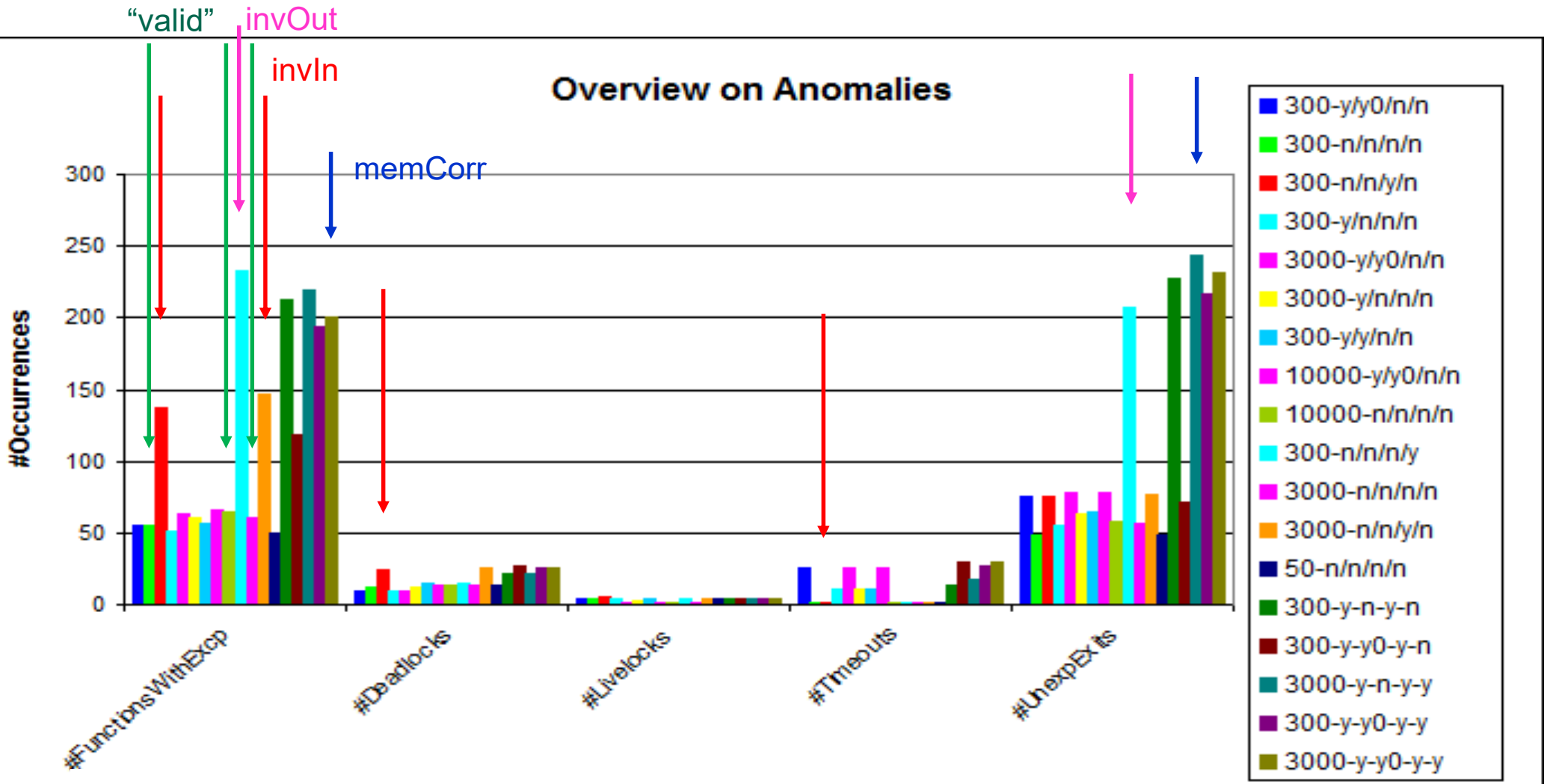
Coverage Type	Coverage / %					Comment
	merged 1 – 13	merged 1-18	merged 14-18	best of 1 – 13	best of 14 – 18	
Block	80.4	82.6	79.4	73.5	77.2	
MC/DC						
true OR false	84.8	86.6	84.7	79.7	82.6	At least one of the boolean values occurred
true AND false	67.8	70.4	62.4	*not available*	*not available*	Both boolean values occurred
Configuration				300-n-n-n-y	3000-y-y0-y-y	

Runs 1 – 13 : no combination of fault injection modes “invalid stimuli” and “modified return”

Runs 14 – 18: combination of “invalid stimuli” and “modified return” at varying number of stimuli

Robustness
Sensitivity on Invalid Values

This figure shows that invalid input (*-n-n-y-n) and invalid output (*-n-n-n-y) may cause problems. Therefore confirmation is required that such values will not occur during system operations



Memory corruption (corruption of tables): *-y-*-n-n or *-*-y-n-n or *-*-y0-n-n

Host – Target Comparison

- **Types of comparison**
  - general statistics on host and target execution
  - comparison of output vectors observed – expected on host and target
    expected values should be the confirmed reference vectors
    confirmation not a matter in the project
  - comparison of changes of parameters before and after test on host and target
    depending on parameter mode IN, OUT, INOUT, RETURN
  - comparison of results host vs. target
- **How are comparisons performed?**
  - comparisons are done automatically
  - built-in code as part of the generated test environment, for every user-defined type
  - results are printed to a log-file
  - evaluation of the log-file contents for the test report
  - diversification of evaluation algorithms to support correctness checks

Host – Target Execution Global Figures

Subject	Native	Host	Target
#Tests started	1530	1511	1488
#Tests completed	1529	1504	1186
#Tests not launched on target	n/a	n/a	1
#Tests killed on target due to timeout	n/a	n/a	59
#TC generated	4014	n/a	n/a
#TC executed	n/a	3972	3341

not all test drivers could be compiled and linked due to anomalies

not all test drivers could be compiled and linked for target

not all test drivers could complete

Host – Target Comparison
Input – Output, summary

Plat form	#Test Drvs.	#Parameters					#Fully Identical Parameters				
		Total	IN	OUT	INOUT	RET	Total	IN	OUT	INOUT	RET
Host	3972	8336	5815	0	1627	894	6491	5813 99.97%	0	581 35.71%	97 10.85%
Target	3341	6845	4645	0	1482	718	5623	4644 99.98%	0	637 42.98%	342 47.63%

Host – Target Comparison
Input – Output, detailed

File	Function	Test ID	Parameter Name	Parameter Type	#Elements	#TC total	#TC					
							#Identical		#Partly different		#Different	
							H	T	H	T	H	T
..\appl_main.c	Func1	0	Para1	IN	1	1	1	-	0	-	0	-
..\file1.c	Func2	1	Para2	IN	61	2	2	2	0	0	0	0
			Para3	INOUT	1		0	0	0	0	2	2
..\file1.c	Func3	2	Para4	IN	61	6	6	6	0	0	0	0
			Para5	INOUT	1		0	0	0	0	6	6
..\file1.c	Func4	3	Para6	INOUT	39	4	0	0	4	4	0	0
			Para7	INOUT	9		0	0	0	0	4	4
..\file1.c	Func5	4	Para8	INOUT	51	1	0	0	1	1	0	0
..\file1.c	Func6	5	Para9	INOUT	161	1	0	0	0	0	1	1
..\file1.c	Func7	6	Para10	INOUT	61	1	0	0	0	0	1	1
..\file1.c	Func8	7	Para11	INOUT	1	1	0	0	0	0	1	1
..\file1.c	Func9	8	Para12	IN	1	4	4	4	0	0	0	0
..\file1.c	Func10	9	Para13	INOUT	1	1	0	0	0	0	1	1
..\file1.c	Func11	10	Para14	INOUT	36	1	0	0	0	0	1	1
			Para15	INOUT	1		0	0	0	0	1	1

Host – Target Comparison
Expected – Observed, summary

Result Type	Plat form	Parameter Mode									
		IN		OUT		INOUT		RETURN		Total	
		#	%	#	%	#	%	#	%	#	%
identical	host	2855751	37.09	0	0.00	1664818	21.62	3173639	41.22	7694208	99.93
	target	145033	44.14	0	0.00	167814	51.07	12635	3.85	325482	99.05
different	host	16	0.00	0	0.00	2051	0.03	3565	0.05	5632	0.07
	target	135	0.04	0	0.00	2755	0.84	221	0.07	3111	0.95
total	**host**	**2855767**	**37.09**	**0**	**0.00**	**1666869**	**21.65**	**3177204**	**41.26**	**7699840**	**100.00**
	target	**145168**	**44.18**	**0**	**0.00**	**170569**	**51.91**	**12856**	**3.91**	**328593**	**100.00**

```
host.lgExecDCRTT18:175:   check_double_DCRTT -0.000000!=0.000000
                                                diff=1.0456925312107449000e-309
```