BSSE System and Software Engineering

SCISYS

FAST II

Automatic Source-code-based Testing, Improvement

Final Presentation

Noordwijk, December 4th, 2019

ESA Contract No. 4000116014 (GSTP)

BSSE Team: Rainer Gerlich, Ralf Gerlich

SCISYS Team: Allan Pascoe, Glenn Johnson

ESA TO: Maria Hernek

Dr. Rainer Gerlich
Auf dem Ruhbuehl 181
88090 Immenstaad
Germany

Tel. +49/7545/91.12.58
Fax +49/7545/91.12.40
Mobile +49/171/80.20.659
email Rainer.Gerlich@bsse.biz

BSSE System and Software Engineering

Contents

SCISYS

- **The FAST Approach**
- **DCRTT Open Tool Interface**
  - VectorCAST Interface
  - Cantata Interface
- **Requirements-Based Testing**
- **Benchmarking**
- **Conclusions and Outlook**

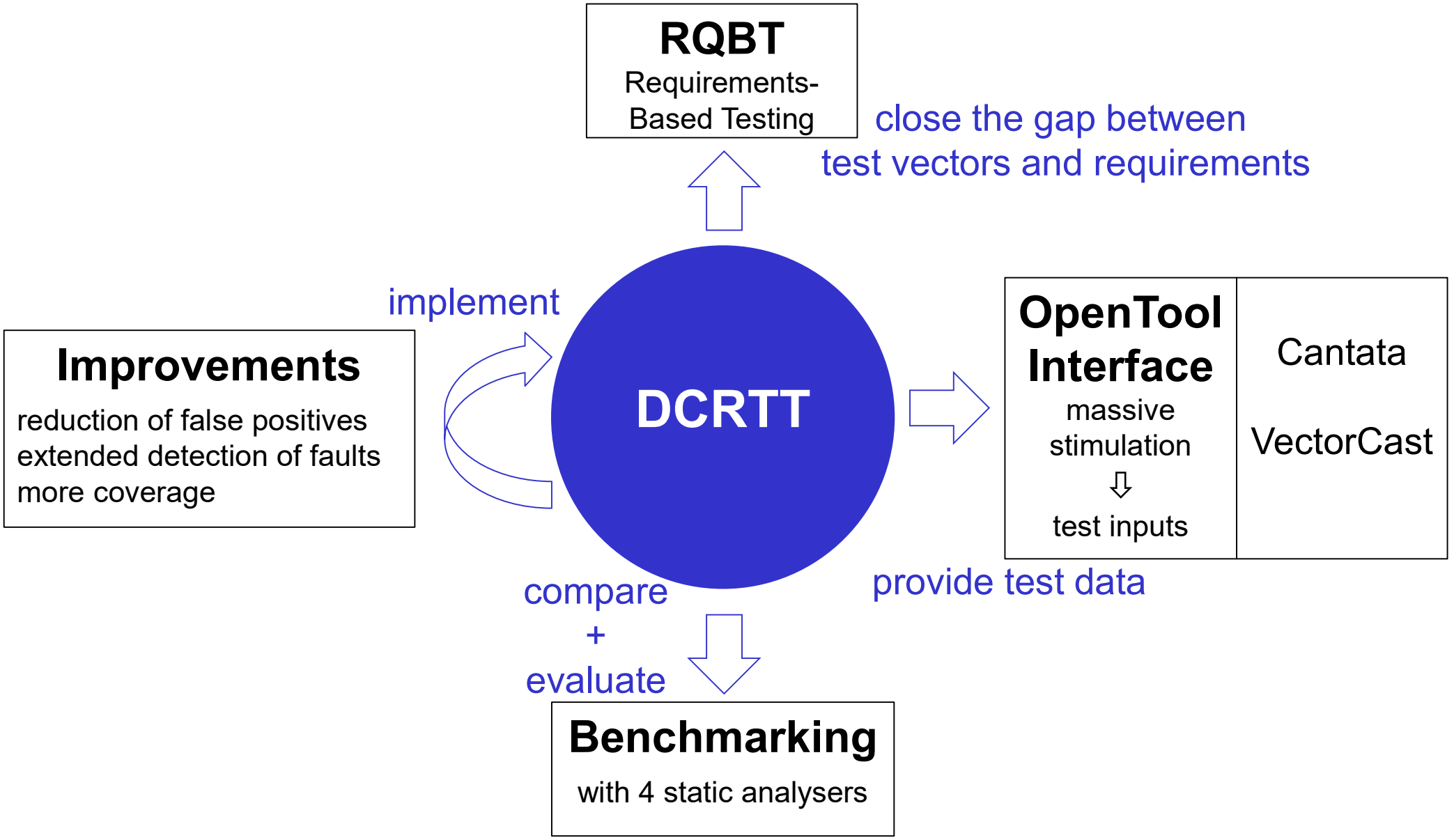
The FAST Approach

About FAST and DCRTT

- **FAST**
  - Flow-Optimised Automated Source-code-based Testing
  - automate the test process from test data generation to report generation

- **DCRTT**
  - Dynamic C Random Test Tool
  - following DARTT, Dynamic Ada Random Test Tool
  - tool supporting the FAST approach

Goals of the Project

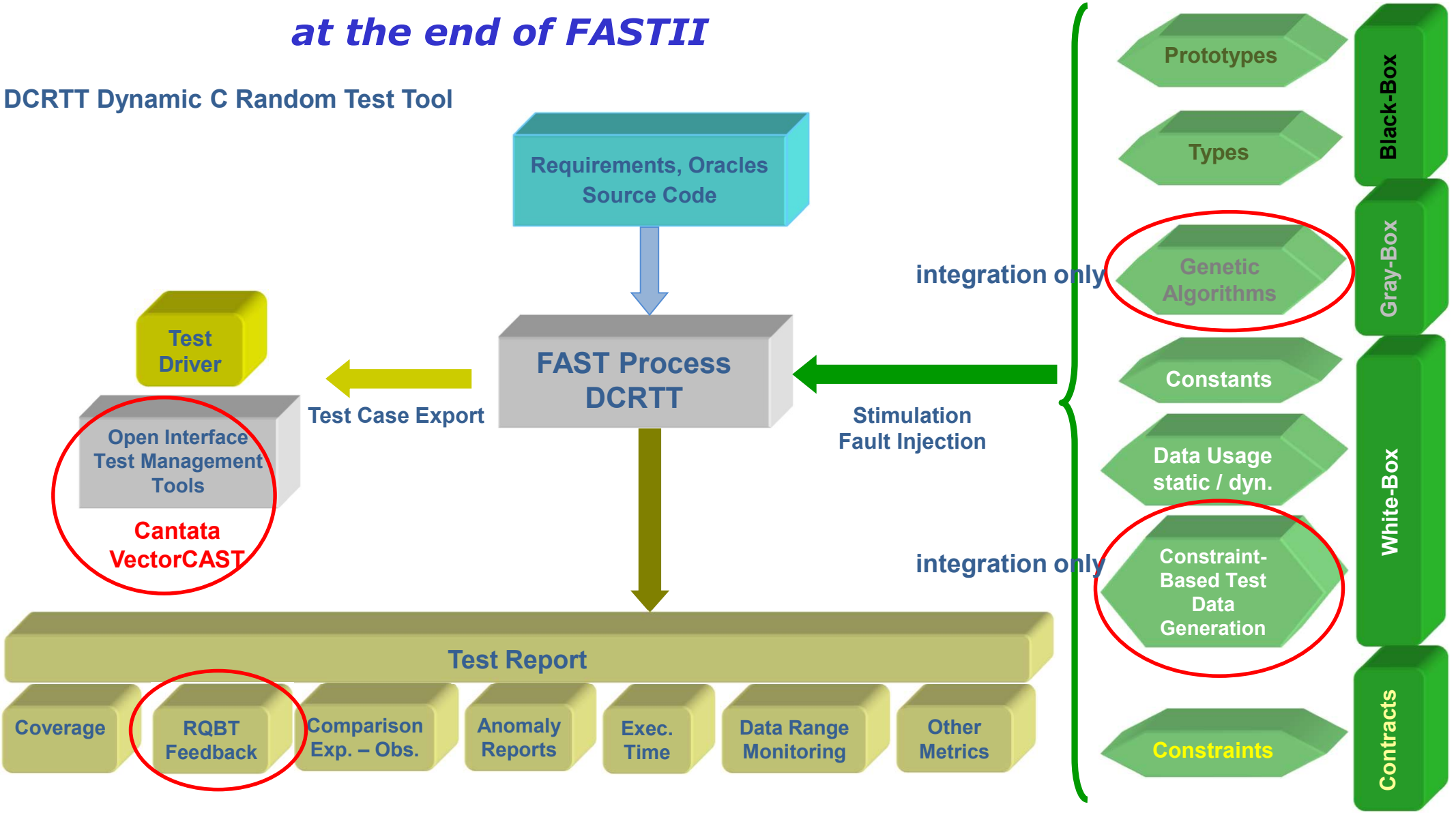


Interfaces of the FAST Test Process in DCRTT

at the end of FASTII

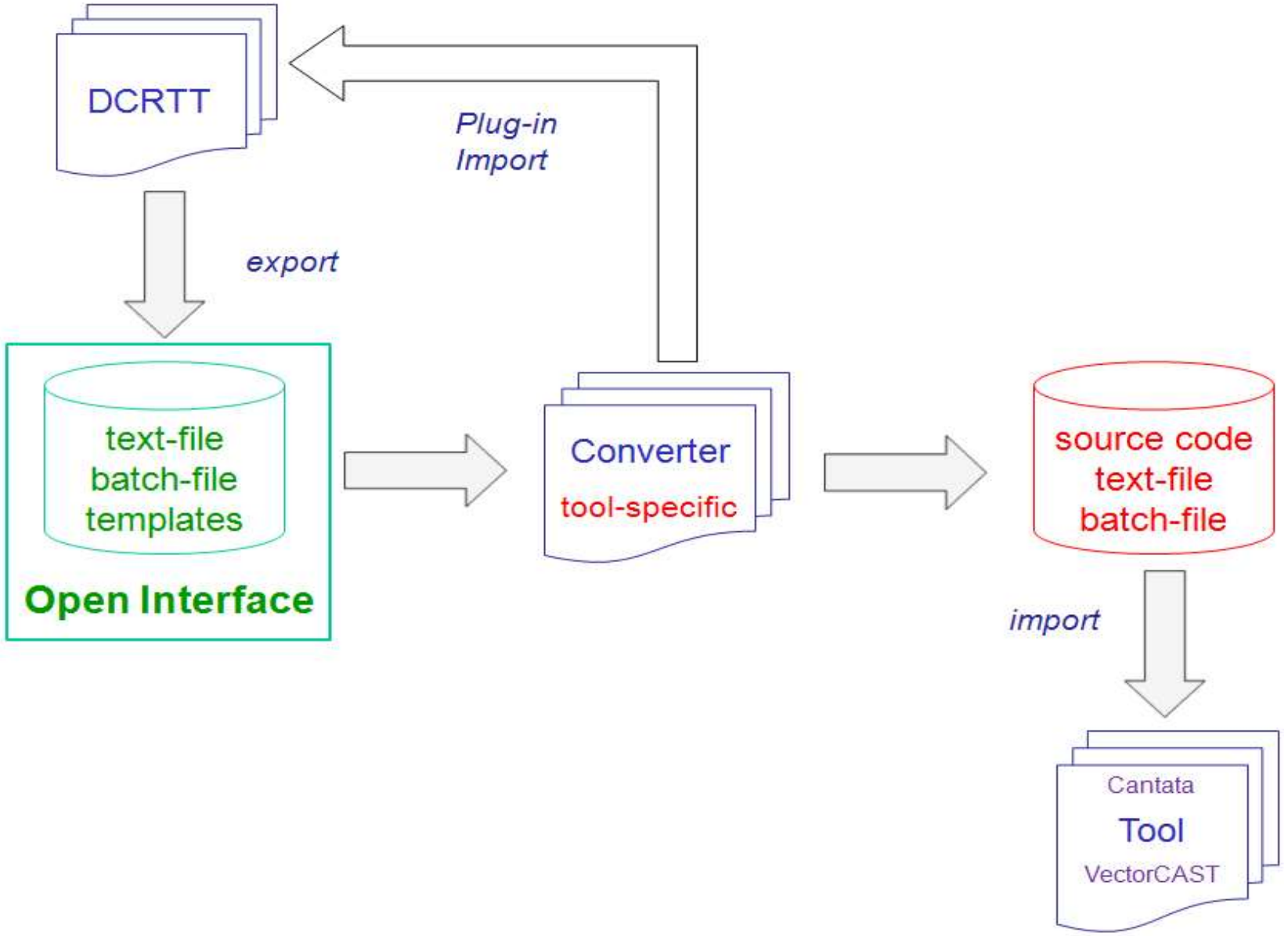
BSSE System and Software Engineering

SCISYS



Open Tool Interface

Open Tool Interface Principal Approach



Output VectorCAST Coverage Report

Aggregate Coverage Report

Configuration Data

Date of Report Creation: 25 MAY 2018

Time of Report Creation: 11:51:10 PM

Aggregate Coverage

Code Coverage for Unit: hello_vcast_1.c'1

```
-- Coverage Type: Statement+MC/DC
-- Unit: hello_vcast_1
-- Test Case: Aggregate
                #include "C:/PROJEKTE/DCRTT/MYTEST/TESTSRCDOIF_VC_ALL_1_5_ALL/DCRTT_test_addons.h"
                #include "C:/PROJEKTE/DCRTT/SRC/DCRTT_ENV/DCRTT_TS/inout_def.h"
                extern int DCRTT_stdout_fprintf(const char *form, ...);
                extern void *missFunc(int para1, int *ptr);
                extern int *missData;
                int *globArrNULL =((void *)0);
                int *globArrNonInit;
                int checkOORdown(int *arr)
                {
1 0      (T)    checkOORdown
1 1       *       int ii,ret=0;
1 2      (T)(F)   for (ii=50;
1 2.1    (T)(F)   ii>=-1;ii--)
1 3       *         ret+=arr[ii];
1 4       *       return ret;
                }
                int checkOORup(int *arr)
                {
2 0      (T)    checkOORup
2 1       *       int ii,ret=0;
2 2      (T)(F)   for (ii=0;
2 2.1    (T)(F)   ii<=15;ii++)
2 3       *         ret+=arr[ii];
2 4       *       return ret;
                }
                int *checkPtrReturn(int *arr)
                {
3 0      (T)    checkPtrReturn
3 1       *       int ii,ret=0;
3 2      (T)(F)   for (ii=0;
3 2.1    (T)(F)   ii<=25;ii++)
3 3       *         ret+=arr[ii];
3 4       *       return arr;
                }
```

Example VectorCast Test Script (2) unconstrained array

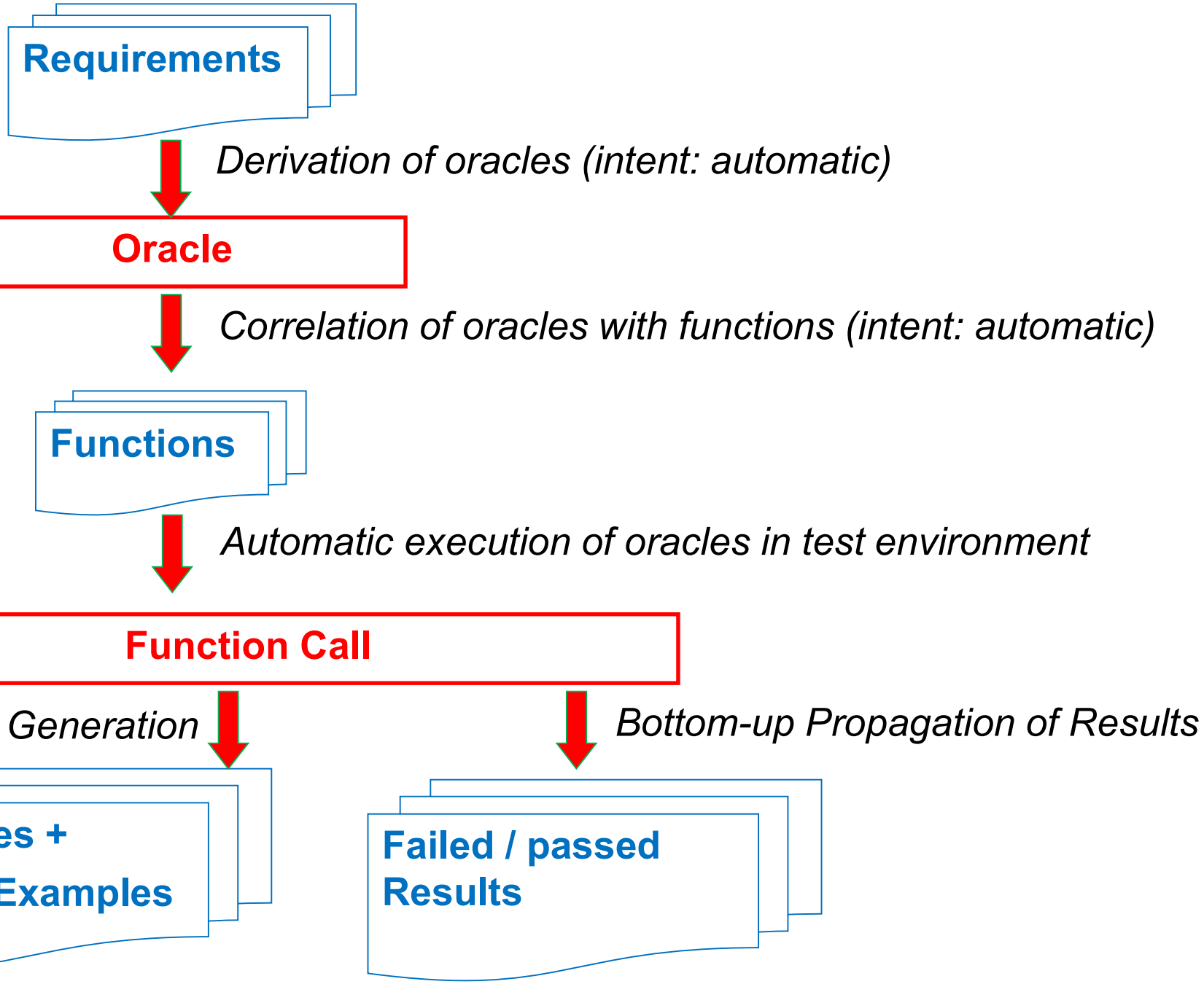
```
-- Test Case Script
-- Environment        :
vc_test_hello_vcast_1_BSSE_main_7
-- Function Under Test: hello_vcast_1 -
BSSE_main 7 of hello_vcast_1.c
-- Script Features
TEST.SCRIPT_FEATURE:C_DIRECT_ARRAY_INDEXING
TEST.SCRIPT_FEATURE:CPP_CLASS_OBJECT_REVISION
TEST.SCRIPT_FEATURE:MULTIPLE_UUT_SUPPORT
TEST.SCRIPT_FEATURE:STANDARD_SPACING_R2
TEST.SCRIPT_FEATURE:OVERLOADED_CONST_SUPPORT
-- End of header hello_vcast_1 - BSSE_main 7 of
hello_vcast_1.c
--  vc_test_7.tst generated by
dcrtt_open_if_cnv_vc.c for tool vc on <date>
-- Test File: hello_vcast_1.c
TEST.UNIT:hello_vcast_1
TEST.SUBPROGRAM:BSSE_main
-- mangled name BSSE_main
-- List of relevant data of function BSSE_main
-- #tot    para=  2
-- #func   para=  2
-- #glob   para=  0
-- #constr para=  0
-- Parameters
--  signed int  argc
--  char * argv[UC_LIT2]
-- Return
--  signed int _return_
TEST.NEW
TEST.NAME:(CL)BSSE_main.001
-- derived from DCRTT test case     1
TEST.NOTES:
No requirements provided
TEST.END_NOTES:
TEST.FLOATING_POINT_TOLERANCE: 9.999999747378751635555e-06
TEST.VALUE:hello_vcast_1.BSSE_main.argv[0]:<<malloc 26>>
TEST.VALUE:hello_vcast_1.BSSE_main.argv[1]:<<malloc 26>>
TEST.VALUE:hello_vcast_1.BSSE_main.argv[2]:<<malloc 26>>
TEST.VALUE:hello_vcast_1.BSSE_main.argv[3]:<<malloc 26>>
TEST.VALUE:hello_vcast_1.BSSE_main.argv[4]:<<malloc 26>>
TEST.VALUE:hello_vcast_1.BSSE_main.argc: -2147483648
TEST.VALUE:hello_vcast_1.BSSE_main.argv[0]: ""
TEST.VALUE:hello_vcast_1.BSSE_main.argv[1]: "lxivmf{lurnmkdzwlqrr
rjqg"
TEST.VALUE:hello_vcast_1.BSSE_main.argv[2]: "g
uxxclxgjnorgwhuqouzjmgi"
TEST.VALUE:hello_vcast_1.BSSE_main.argv[3]: "LQWT7WNaMA0K0HKJTMR
D1423"
TEST.VALUE:hello_vcast_1.BSSE_main.argv[4]: "1@i5}A6}7\\'%kB^I$2r
2)7m3"
TEST.VALUE:hello_vcast_1.BSSE_main.return: -2147483648
TEST.EXPECTED_USER_CODE:hello_vcast_1.BSSE_main.argc
{{ (signed long)<<hello_vcast_1.BSSE_main.argc>> == ( (signed
long)-2147483648) }}
TEST.END_EXPECTED_USER_CODE:
TEST.EXPECTED_USER_CODE:hello_vcast_1.BSSE_main.argv
{{ strcmp( <<hello_vcast_1.BSSE_main.argv>>[0] ,  "1234567890" ) }}
{{ strcmp( <<hello_vcast_1.BSSE_main.argv>>[1] ,  "1234567890" ) }}
{{ strcmp( <<hello_vcast_1.BSSE_main.argv>>[2] ,  "1234567890" ) }}
{{ strcmp( <<hello_vcast_1.BSSE_main.argv>>[3] ,  "1234567890" ) }}
{{ strcmp( <<hello_vcast_1.BSSE_main.argv>>[4] ,  "1234567890" ) }}
TEST.END_EXPECTED_USER_CODE:
TEST.EXPECTED_USER_CODE:hello_vcast_1.BSSE_main.return
{{ (signed long)<<hello_vcast_1.BSSE_main.return>> == ( (signed
long)0) }}
TEST.END_EXPECTED_USER_CODE:
TEST.END
```

Output Cantata for Test Script / Decision Coverage

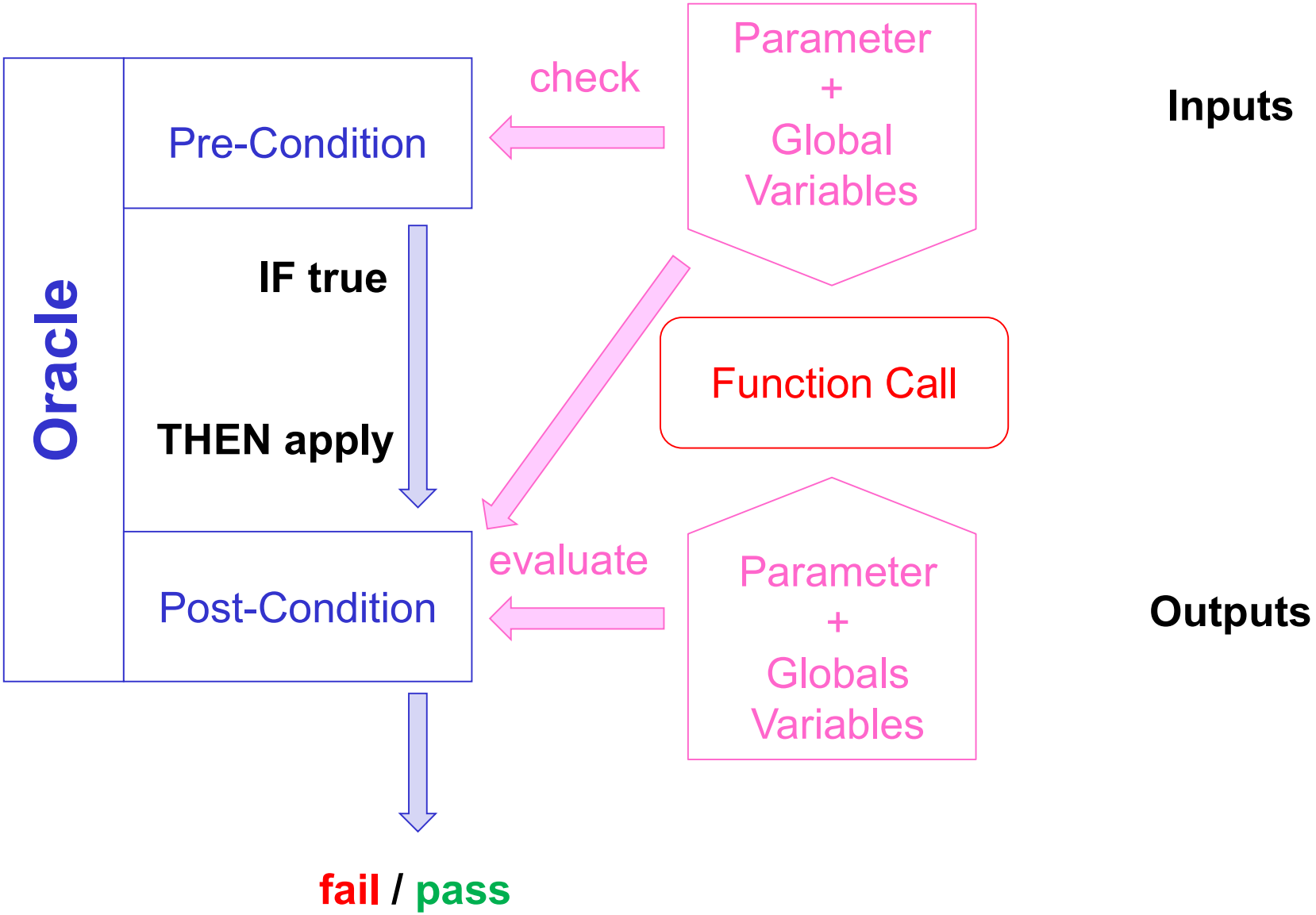
```
hello_cantpp_1.c(152):BSSE_main(int ,char **)
decision coverage details (with executed and un-executed cases)

hello_cantpp_1.c(156):       decn   1 (for)    branch TRUE              31744
hello_cantpp_1.c(156):       decn   1 (for)    branch FALSE              1024
hello_cantpp_1.c(158):       decn   2 (if)     branch TRUE               6144
hello_cantpp_1.c(160):       decn   2 (if)     branch FALSE             25600

"BSSE_main"                                        executed                4
"BSSE_main"                                      un-executed               0

hello_cantpp_1.c(166):BSSE_main2(int ,char **)
statement coverage details (with executed and un-executed cases)

hello_cantpp_1.c(169):      stmnt   1 (other)                            512
hello_cantpp_1.c(170):      stmnt   2 (loop)                             512
hello_cantpp_1.c(170):      stmnt   3 (loop)                           16384
hello_cantpp_1.c(172):      stmnt   4 (cond)                           16384
hello_cantpp_1.c(173):      stmnt   5 (other)                          13312
hello_cantpp_1.c(175):      stmnt   6 (other)                           3072
hello_cantpp_1.c(177):      stmnt   7 (return)                           512

"BSSE_main2"                                       executed                7
"BSSE_main2"                                     un-executed               0

hello_cantpp_1.c(166):BSSE_main2(int ,char **)
basic block coverage details (with executed and un-executed cases)

hello_cantpp_1.c(167):      block   1                                    512
hello_cantpp_1.c(171):      block   2                                  16384
hello_cantpp_1.c(173):      block   3                                  13312
hello_cantpp_1.c(175):      block   4                                   3072
```

Requirements-Based Testing (RQBT)

Requirements and Coverage



Structure of an Oracle



Oracle Examples

Three Oracles

status==active && mode==mode1 ? moniFlag==true

status==active && mode==mode2 ? moniFlag==true

status==active && mode==mode3 ? moniFlag==false

Pre-condition:
if true check post-condition

Post-condition:
if true: pass
if false: fail

Function	Requirement	Oracle		Number of Tests	Oracle Output			RQ fully covered	RQ verified
		Pre-Condition	Post-Condition		Coverage	true	false		
x*x	∀x∈{double} $\sqrt{x^2}$ shall not differ from x more than ε	x≤-1.0 || x≥1.0	(fabs(fabs(x)-sqrt(retVal))/x)<eps	302	299	225	74	yes	no
		x>-1.0 || x<1.0	fabs(fabs(x)-sqrt(retVal))<eps		3	3	0		
abs(x)	∀x∈{sint} abs(x) shall be ≥0	RQBT_FORALL(x)	retVal>=0	302	302	301	1	yes	no

counter examples found

BSSE System and Software Engineering

Conclusion on Requirements Analysis

SCISYS

- **Readiness for Auto-Extraction of Information**
  - ***only some requirements found suitable for auto-generation of oracles***
  - ***formal models of requirements needed***
  - ***guidelines required***
- **Requirements Top-Down Tracking**
  - ***continuous tracking chain required***
  - ***all functions must track back to at least one requirement***
- **DCRTT Implementation**
  - infrastructure available supporting this notation
  - support for bottom-up propagation available
  - demonstrated bottom-up result propagation / requirements fulfilment

Benchmarking

Selected Tools

	Analysis Type	Analysis Approach	Soundness
DCRTT	dynamic	test, auto-stimulation and auto-test data generation	not sound
Astree	static	abstract interpretation	sound
CodeProver			sound
BugFinder			not sound
QA/C	static	symbolic execution, dataflow analysis	not sound

The Application

- **Two versions**
  - early version with potentially more defects
  - late version with potentially less defects
- **Intention of using two versions**
  - evaluate impact on reporting by number of reports
  - no significant difference found
- **Characteristics of Application**
  - ~190 KLOC
  - 60-70 tasks (periodic, synchronous, sporadic)
  - ~120 functions missing ⇒ stubbed

Analysis and Test Modes

Execution Mode	Description
EM 1	deterministic / sequential execution of the task bodies
EM 2	non-deterministic / random execution of the task bodies
EM 3	modelling of concurrent execution of task bodies with pre-emption
Unit testing	every function is subject to stimulation / testing
functionwise	every function is independently analysed

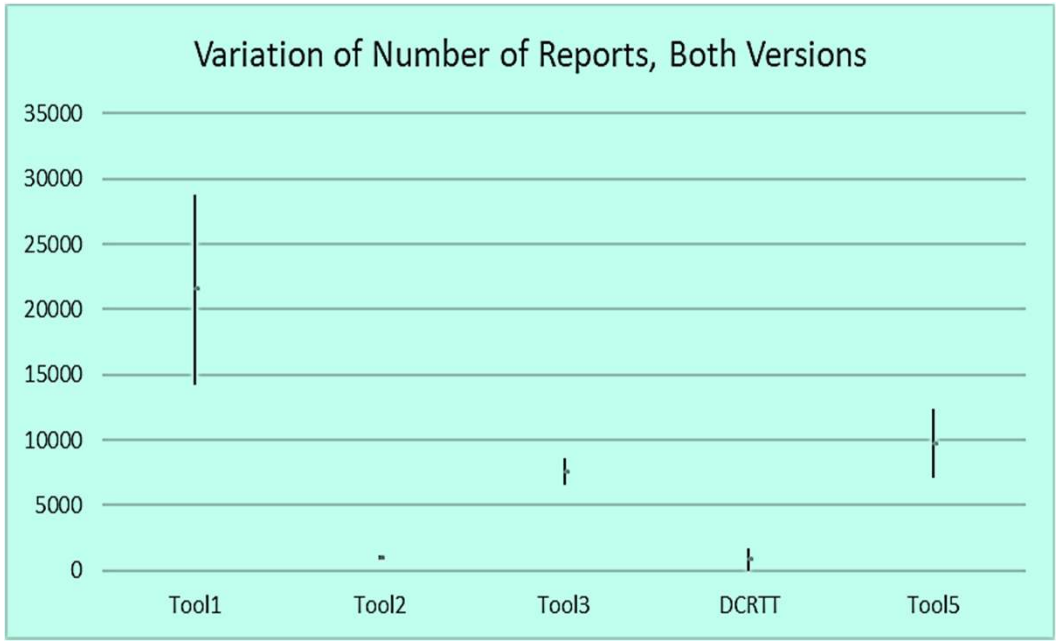
EM = Execution (analysis) mode

Analysis Mode	Tool				
	DCRTT	QAC	Astree	BugFinder	CodeProver
EM 1	x		x	x	x
EM 2	x		x	x	x
EM 3			x	x	x
Unit testing	x				
functionwise		x			

Remarks on Benchmarking Evaluation Results

- **Boundary Conditions**
  - Benchmarking was performed at the end of development
  - Many reports issued by the tools (up to ~30.000)
  - Unclear: Number of reports & effort in case of continuous integration
- **Application Impact**
  - number and type of reports depend on application defect profile
- **Configuration Impact**
  - number and type of reports strongly (tool-)configuration dependent
- **Report Selection Impact**
  - evaluation results strongly depend on selection process
  - number of reports issued may heavily differ between tools

Variation of Report Figures Both Versions



To be considered:

different reporting policies of tools

- reporting for every location?
- reporting for every path?
- duplication of reports?

Report Classification

		Code Status	
		Defect present	Defect not present
Reported	Defect present	True Positive / TP	False Positive / FP
	Defect not present	False Negative / FN	True Negative / TN

Classification Category	Criterion	Applied Evaluation Condition / Check
Validity	tool	Is the tool message formally correct?
	state	Can an undesired state really be reached?
Context	with context	The execution conditions may be constrained by the calling function (caller)
	without context	The execution conditions are not constrained

BSSE System and Software Engineering

Std. Defect Types vs. Tools

SCISYS

Results depend on application, tool configuration and defect profile

Id	Standard Defect Type	Astree			BugFinder			CodeProver			DCRTT			QAC		
		supp	330	450	supp	330	450	supp	330	450	supp	330	450	supp	330	450
1	Array Index Out-of-Bounds		x	x		x	x		x	x	x	x	x			
2	Assert failure		x	x		x	x		x	x	x	x	x			
3	Dangling Pointer					x										
4	Dereference of Invalid or NULL Pointer		x	x		x	x		x	x	x	x	x		x	x
5	File Access Error										x					
6	Format String Mismatch					x										
7	Invalid arithmetic operation		x	x											x	x
8	Invalid function pointer		x	x					x	x						
9	Invalid Return Statement															
10	Loss of Precision					x	x								x	x
11	Macro Use with Unintended Consequences															
12	Non-terminating Loop		x	x					x	x	x					
13	Possible Invalid Use of Function		x	x		x	x		x	x						
14	Possible Recursion		x	x							x					
15	Resource Leak										x					
16	Undefined Result								x	x					x	x
17	Uninitialized Variable		x	x		x	x		x	x					x	x
18	Unintended Use of Implicit Member Function															
19	Arithmetic Operation on NULL Pointer		x	x												
20	Arithmetic Overflow		x	x		x	x		x	x	o				x	x
21	Cast to pointer of incompatible types		x	x											x	x
22	Comparison of floating-point values														x	
23	Concurrency Issues		x	x		x	x		x	x						
24	Conflicting Declarations															
25	Incomplete List of Cases for enum-Type without default		x													
26	Intended Change of Invariant Data		x			x	x									

	critical		uncritical
	warning		to be ignored

No contribution for late version

Variation of Report Figures (Excerpt)

Late Version						
Defect Type	Tool1	Tool2	Tool3	DCRTT	Tool5	Comment
Assert	363	6	343	227	0	compromised by stubbing
Concurrency Issues	10755	807	2633	n/a	n/a	non-relevant due to non-representative scheduling
Unused Result	4909	732	0	n/a	0	
Uninitialized Variable	1140	26	1215	n/a	4	

Report Summary

Number of Reports								
Tool	Early Version				Late Version			
	entry-point-function			unit test	entry-point-function			unit test
	determ.	non-det.	multi-task.		determ.	non-det.	multi-task.	
DCRTT	31	31	n/a	1590	16	21	n/a	1638
Other Tools	One tool supplier denied publishing of data, therefore no figures are published with reference to a tool **range 800 – 29000 *(entries in std. csv-file)*** ***(entries extracted from tool-specific report files)***							

more functions, more reports

Evaluation Results for TP Based on Entry-Point Function

Tool	Version	Execution Mode	#Reports	TP		FP				
						with ctxt				w/o ctxt
				with ctxt	w/o ctxt	**tot**	data	task	stub	
DCRTT	330	deterministic	31	4	31	**27**	20	3	4	0
		non-deter.								
	450	deterministic	16	1	16	**15**	8	3	4	0
		non-deter.	21		21	**20**	13			

source of FP
non-representative analysis environment

4 + 1 from DCRTT entry-point execution
+ 1 from DCRTT unit testing

Vers.	Function	Location	Id	Reported by		Confirm.
				DCRTT		
				Entry-point-version	Module Testing	Other Tools
early	*func1*	*X*	1	*determ. + non-determ.*	x	*2x*
		X+2	2			*1x*
		X+5	3			*1x*
	func2	*Y*	4			*2x*
late	*func3*	*Z*	5	*determ. + non-determ.*	none	*1x*
	func4	*U*	6	none	x	*2x*

The two issues for the late version – highlighted in the analysis – have no impact on the current operational concept and mission performance.

Final Evaluation Results (late)

Report Assessment	Tool w/o ctxt	State w/o ctxt	Tool with ctxt	State with ctxt
TP	48	46	11	**9**
assert	7	7	1	**1**
out-of-bounds	34	33	9	**8**
dereference	6	6	0	**0**
uninitialised	1	0	1	**0**
FP	8	10	45	**47**
Total	56	56	56	56

Consolidated reports 56 < 57 manually evaluated reports

Relevance	Assessment	Number of TP Reports	
		In 450	Overlap with 330
should be fixed	one-off-index fault (1x), invalid index (1x)	2	0
not relevant	hidden check	3	0
not relevant	Assertion failure due to stub	1	1
not relevant	Supposing that telecommand contents is checked on-ground or in another task, check not visible	3	3

Context Approximation Example 1/2: interval approx.

```
typedef enum {
  lit0=0,lit1=1,lit2=2,
  litInvalid=255
} TySet;

TySet map2set(uint8_t para){
  if (para==0) return lit0;
  else if (para==1) return lit1;
  else if (para==2) return lit2;
  else return litInvalid;
}

int myArr[lit2+1];

void myFunc(TySet para) {
  idx=map2set(para);

  if (idx != litInvalid)
       myArr[idx]=0;
  return;
}
```

```
setEnum  =0,1,2,255 exact

setReturn=0,1,2,255 exact
setApprox=[0,255]    interval approx.

setApprox =[0,255]

setApprox2=[0,254], 255 removed
idx may be > 2: FP will be reported
```

Context Approximation Example 2/2: min/max

```
int myMapping[6]=
{1,3,5,54,7,78};

char mySrc [100];
char myDest[  5];

void myFunc(int idx) {
  if (idx>0 && idx<6) {
    memcpy(dest,
           src,
           myMapping[idx]);
  return;
}

int main(int argc, char* argv)
{
  myFunc(2);
  return 0;
}
```

array contents is approximated / squashed
by min/max: [1,78]

min/max are considered here:**[1,78]**

myFunc is called with
idx=2 ⇨ size=5 which is **valid**

but taking the maximum 78 the following report is issued:
78 out-of-bounds [0,5]

Sources of False Positives (1/2)

- **Mismatch of Verification Criteria and Programming Style**
  - if verification criteria are not considered during development ⇨ high number of FP
  - verification tool(s) should be considered continuously over the development cycle
- **Non-representativity of the analysis environment**
  - just exposing the source code to the analysis may not be sufficient
  - e.g. stubbing, scheduling scheme, dynamic changes of object structure (telecmd.)
  - mockups may be required to represent environment
- **Non-representativity of the analysis method**
  - context vs. robustness trade-off and approach of chosen tool
  - provision of required context information by tool and user

Sources of False Positives (2/2)

- **Approximation of the context, static analysers, abstract interpretation**
  - exact representation of the context vs. memory consumption and runtime
  - context information may be lost due to approximation
  - benefits of using context information may be lost ⇨ increased number of FP

- **Missing or non-visible checks**
  - checks not present to ensure valid conditions
  - checks present but not visible for the verification tool, e.g. task boundaries or ground checks

Recommendations

- Ensure a representative context to the degree possible
- Choose the right tool approach for the envisaged verification goal

  robustness testing vs. pure unit testing, context-sensitive or not

  provide as much context-information as possible
- Consider the feedback from the verification tool(s) as early as possible during coding
- Fix the defects according to the tool feedback
- Discuss a trade-off on protection against invalid data

  check or do not check?

  Checking will reduce the amount of false positives

BSSE System and Software Engineering

SCISYS

Conclusions and Outlook

FAST / DCRTT Potential and Test Strategies