Gerlich System and Software Engineering

**FAST** and **DCRTT**

# Capabilities of the FAST Test Process and the DCRTT Tool

the extended test automation process and tool for C and C++
to save effort in test preparation, execution and evaluation.

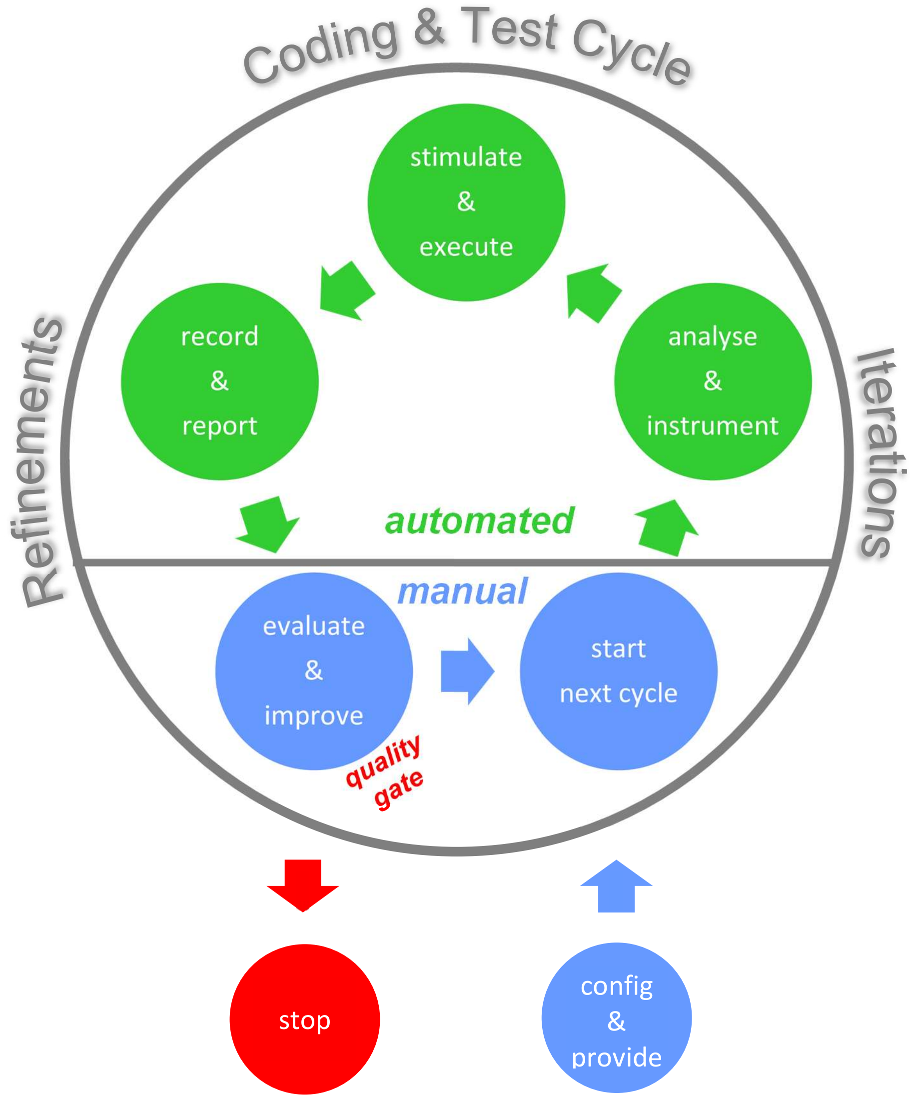
**Get coverage figures, information on anomalies and much more**
**without manual intervention**
**just by providing (compilable) source code.**

**Put your software under stress instead of yourself.**

# Capabilities of the FAST Process and the DCRTT Tool

**Take benefit from full test automation:**

- **let an automaton do the job of from test data generation to reporting including instrumentation,**
- **continuously apply the test cycle right from beginning of coding and improve your software by feedback,**
- **save time and costs by automation, but not by compromising the test goals,**
- **easily inject large numbers of diverse test stimuli into your software,**
- **apply random and grid-based test data generation,**
- **apply fault injection and robustness testing,**
- **perform unit testing and testing of the integrated software system,**
- **automatically embed functions into wrappers for monitoring and exception handling during testing and at run-time.**



The FAST process (***F**low-optimised **A**utomated, **S**ource-code-based **T**est*) together with the tool DCRTT (***D**ynamic **R**andom/**R**obustness **T**esting **T**ool*) allow you to start the test process by provision of source code, and to get a variety of reports at the end without any further manual intervention. The provided source code only needs to be compilable on host and target platform.

# Capabilities of the FAST Process and the DCRTT Tool

| Features of the FAST Test Process and DCRTT | | | |
|---|---|---|---|
| **Test Cycle** | 1. Provide the source code.<br>2. Start the automated process (with execution on host or a target).<br>3. Wait for the automatically generated reports. | | |
| **Languages** | C / C++ | | |
| **Lifecycle Support** | function testing | ✓ | |
| | integration testing | ✓ | |
| | requirements-based testing | ✓ | |
| | regression testing | ✓ | |
| | robustness testing | ✓ | |
| | tasking simulation, non-pre-emptive | ✓ | |
| | early/continuous testing over LC[1] | ✓ | |
| | data mutation | ✓ [2] | |
| **Test data generation** | automated | black-box | random-based |
| | | | grid-based |
| | | gray-box | genetic (research topic) |
| | | white-box | constraint-based |
| | | | source code heuristics |
| **Script generation** | automated for testing of the full set of functions included in the provided source code | | |
| **Test environment** | automated for testing on top-level / execution of threads (non-pre-emptive)[3] [4] | | |
| | missing symbols automatically added | function, stubs<br>data generation for return and out-parameters<br>nominal/non-nominal range | |
| | | global data<br>nominal/non-nominal range<br>overriding of const if desired for fault injection | |
| | type ranges | automatic identification of constraints on nominal range | |
| **Test Execution** | automated, based on massive stimulation | host | |
| | regression testing based on automatically generated test drivers, remote execution on target | host | |
| | | target | |
| | auto-resizing of data objects to reduce false positives | unconstrained arrays | |
| | | pointers | |
| | for C++<br>constructors / destructors | constructor calls in random sequences, random number of parameters (if variable) and with random data | |
| | Automated generation of wrapper-functions for mitigation (return value check, monitoring of execution time, function termination if deadline is exceed, asynchronous interruption of control flow, a number of reporting options) | | |
| | Support of file management on host: if target only supports one physical file stream (e.g. stdout, UART), tunnelling of separate logical files through single channel and re-separation on host | | |
| **Test modes** | test data in nominal range | | |
| | test data in nominal and non-nominal range (fault injection[5]) | | |
| | fault injection on return and out-parameters of called functions[5] (esp. malloc, …) | | |
| **Report generation** | Automated, based on test evaluation | csv | |
| | | rtf-document | tables |
| | | | graphics |
| | | xml | *on request* |
| | | other formats | *on request* |

---

[1] due to automation of test organisation and stubbing, tests can be executed as soon as source code is compilable

[2] return value, scalar output, condition value

[3] customer-specific notations may be converted into the DCRTT format

[4] stimulation of external system interfaces

[5] for functions-under-test: parameters and used global data; for (non-static) functions in general: return values and output parameters

# Capabilities of the FAST Process and the DCRTT Tool

| Test Evaluation | | |
|---|---|---|
| **Coverage** | block | ✓[6] |
| | decision, condition, MC/DC | ✓ |
| | differential coverage[7] | ✓ |
| | test input vector vs. coverage[8] | ✓ |
| **Fault detection** | assertions | ✓ |
| | concurrency | ✓[9] |
| | exceptions | ✓ |
| | file handling faults | ✓ |
| | file-descriptor leakage | ✓ |
| | index out-of-range | ✓[10] |
| | invalid addresses / pointers | ✓[11] |
| | malloc-memory corruption | ✓ |
| | malloc-memory leakage | ✓ |
| | NULL-pointer dereference | ✓ |
| | numerics (float) | ✓[12] |
| | recursion | ✓[13] |
| **Verification** | test input vector vs. test output vector, manually | ✓[14] |
| | numerical differences / float, host vs. target, automated | ✓[15] |
| | Oracles with pre- and post-condition, automated | ✓[16] |
| **Resource consumption** | measurement of execution time | host and target[17] |
| | heap and stack usage | ✓[18] |
| | malloc, heap and stack usage | ✓[19] |
| | file usage | ✓ |
| **Reports** | test data distribution | ✓ |
| | data ranges | ✓ |
| | anomaly reports (csv) | ✓ |
| | tables | ✓ |
| | graphics | ✓ |
| | compilation, link and execution times | ✓ |
| | test log-files | ✓ |
| | result comparison host-target | ✓[20] |
| | sensitivity analysis (1st derivative) | ✓[21] |

---

[6] full block coverage is equivalent to full statement coverage. Intermediate percentages need to be converted.

[7] identification of coverage provided per generated input test vector

[8] identification of input vector(s) providing the coverage

[9] scheduling issues, non-pre-emptive, checks on semaphores

[10] for explicit index expressions and for dedicated C-library functions (memcpy, memset, strcpy, …)

[11] For pointers in source code and provided to dedicated C-library functions (memcpy, memset, strcpy, …)

[12] detection of inf, NaN and overflow conditions

[13] call-stack analysis during execution, limit-based

[14] manual comparison of values recorded in test-driver code

[15] automated comparison of test output vectors

[16] requirements-based testing, oracles derived from suitable requirements

[17] depending on I/O support on the target platform

[18] by static analysis

[19] by measurement

[20] comparison of exceptions and differences in numerical results (floating-point arithmetics)

[21] metric "change of output vs. previous vs. change of input" for every parameter, graphics, grid-based data generation only

# Capabilities of the FAST Process and the DCRTT Tool

| Interfaces | | |
|---|---|---|
| **User** | configuration of text execution | |
| | definition of range constraints | |
| | definition of initialisation calls | |
| **Tools** | Cantata | conversion of test drivers to tool notation, execution of test drivers in tool environment |
| | VectorCAST | |

| General Information | | |
|---|---|---|
| **Supplier** | Dr. Rainer Gerlich System and Software Engineering, GSSE<br>Auf dem Ruhbuehl 181, 88090 Immenstaad, Germany<br>Phone: +49 7545 911258  Mobile: +49 171 8020659<br>E-Mail: contact@gsse.biz | |
| **Customer Service** | Tool Customizing | ✓ |
| | Training/Coaching | ✓ |
| | User Support | German |
| | | English |
| **Host OS** | Windows | |
| **Target-OS** | bare machine | |
| | Linux | |
| | Rodos | |
| | RTEMS | used in context of 4.11 |
| | VxWorks | used in context of 5.3 |
| **Compiler** | gcc | up to version 8, also cross-compiler |
| | VC++ | *on request* |
| **Installation** | desktop / stand-alone | |
| **Licensing** | commercial / service-based | |

### Robustness Testing

A major feature of DCRTT is the capability for robustness testing. The goal is to expose the software-under-test to non-nominal conditions and to evaluate whether it can protect against them.

In case of safety issues the software shall tolerate these invalid conditions and shall not show unexpected behaviour, e.g. in case of memory access violation, which may cause a crash or degradation of operation.

In case of security issues the concern is, e.g. that unprotected access of memory may allow penetration by which unauthorized control over a system or unauthorized access of confidential data may be possible

Standards requesting robustness testing for safety issues are (non-exhaustive list)

- ISO 26262 - Road vehicles – Functional safety
- DO178 - Software Considerations in Airborne Systems and Equipment Certification

and for security issues (non-exhaustive list):

- ISA/IEC 62443-4-1 - Security for industrial automation and control systems – Part 4-1: Secure product development lifecycle requirements
- ISO/SAE 21434 - Road vehicles — Cybersecurity engineering
- ISO 27001 - Information technology – Security techniques – Information security management systems – Requirements
- ISO 22301 - Security and resilience – Business continuity management systems – Requirements

## Capabilities of the FAST Process and the DCRTT Tool

### Results from a Space Project

The following tables provide information on the software-under-test and the achieved coverage results. The space application contains 3400+ functions of criticality level Cat. B and Cat. C (Tab. 1) according to the ECSS norms (Cat. A is the highest criticality category).

| Application | | | |
|---|---|---|---|
| **Item** | # | KLOC | Comment |
| h-files | 170 | 29 | |
| c-files | 150 | 167 | |
| all files | 320 | 196 | |
| functions, all | 3400+ | | Cat. B: mission critical<br>Cat. A: highest criticality level in this terminology |
| functions, Cat. B | 1900 | | |
| functions, Cat. C | 1500 | | |
| tasks | 70 | | |
| operating system | RTEMS | | |

*Tab. 1: Source Code Statistics of a Space Project*

Test coverage is one of the criteria required by software standards for assessment of software quality. It provides feedback about the extent to which the source code lines were reached and executed. Execution of every source code line is a necessary – although not sufficient – condition for detecting faulty behaviour.

*The more often a line is executed under different conditions while behaving as expected, the more confidence in the proper function of the software is warranted. Due to automated test data generation, DCRTT can expose the software to a huge number of test vectors implying that a line is executed more than once under different conditions, while standards usually require only one execution per line, due to the limitations of manual test data preparation. For Cat. C software the ECSS require 80% statement coverage (with an option for adaptation by negotiation with the customer).*

Figures obtained from 4 different configurations with random or random/grid-based test data generation are shown in Tab. 2:

- 3 configurations for function (unit) testing with different stimulation modes (configurations 1-3), and
- 1 configuration where the integrated system was stimulated by its external interfaces (Configuration 4).

| Stimulation Mode | | | | | | Configuration Id | Total Test Steps |
|---|---|---|---|---|---|---|---|
| **test goal** | data generation mode | parameter | globData | fault injection | tasking | | |
| **function test** | random + grid | ✓ | | | | 1 | 370221 |
| | | ✓ | ✓ | | | 2 | 347446 |
| | | ✓ | | ✓ | | 3 | 429306 |
| **integration test / top-level** | random | tele-commands | external data | | ✓ | 4 | 500000 |

*Tab. 2: Test Configurations*

In the latter case (Configuration 4) telecommands and external data were injected. The telecommands were generated according to a pre-existing XML-specification. For the data interface no specification on the nominal data ranges and valid data sequences did exist. Therefore, most of the injected data were rejected, which resulted in a rather low coverage. However, the merges with function tests show that a combination of different test configurations can result in a reasonably high coverage.

## Capabilities of the FAST Process and the DCRTT Tool

The combinations of the different test runs are provided in Tab. 3. They show

- that different test configurations provide complementary coverage, and
- that higher total coverage figures can be achieved by running different test configurations.

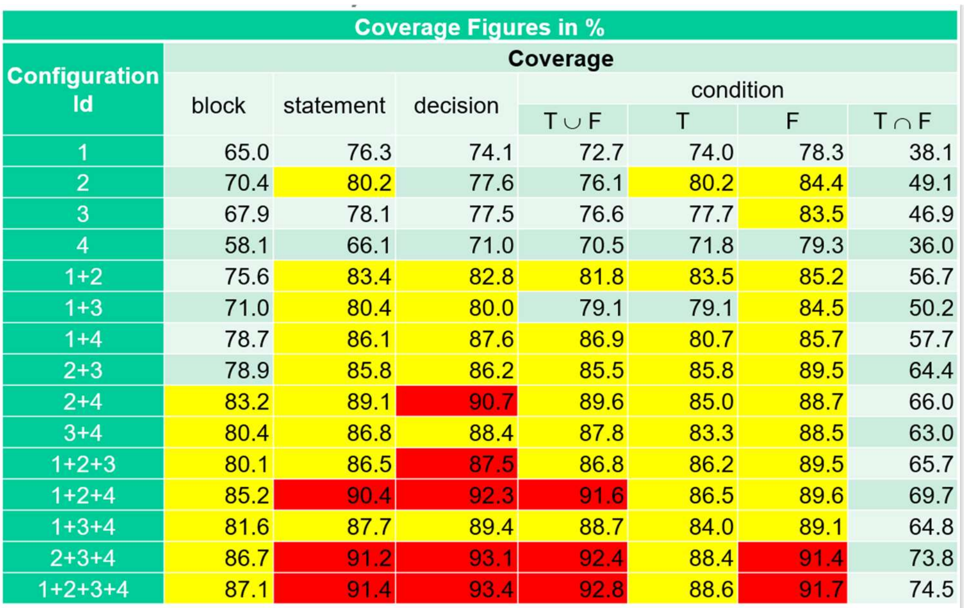
The results were mainly obtained by black-box testing as well as random- and grid-based test data generation. Based on the maximum coverage which is obtained by merging the coverage figures from all 4 test runs, the constraint-based test data generation could be applied to the non-covered blocks and conditions.

The sequence – random-/grid-based first, then constraint-based – is recommended because constraint-based generation is rather time-consuming compared to random/grid-based generation.

For the execution of all configurations no manual effort is required except for definition / modification of the test configuration and extraction of the figures from the report, while rather high coverage figures can be achieved at little effort – compared to the effort which would have to be spent for manual testing.

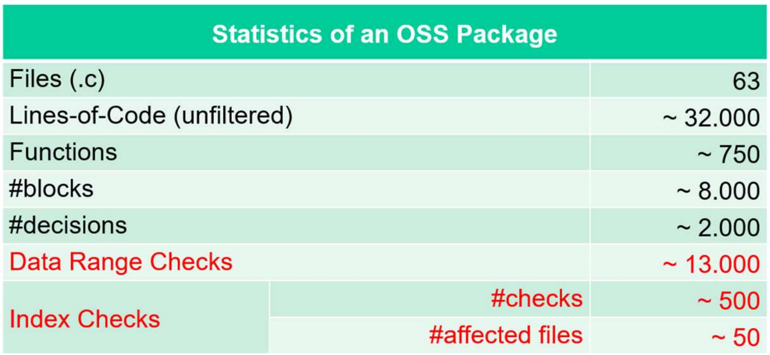
Apart from the coverage figures, a lot of other information, e.g., on anomalies, is provided in addition – with added effort.

The number of test steps applies to massive stimulation in the first step. The number of test drivers (test case candidates) is much smaller. Depending on the complexity of the software the observed average number of test drivers per function is in the range of 2 – 10.



| Coverage Figures in % | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Configuration Id** | **Coverage** | | | | | | |
| | block | statement | decision | condition | | | |
| | | | | T ∪ F | T | F | T ∩ F |
| 1 | 65.0 | 76.3 | 74.1 | 72.7 | 74.0 | 78.3 | 38.1 |
| 2 | 70.4 | 80.2 | 77.6 | 76.1 | 80.2 | 84.4 | 49.1 |
| 3 | 67.9 | 78.1 | 77.5 | 76.6 | 77.7 | 83.5 | 46.9 |
| 4 | 58.1 | 66.1 | 71.0 | 70.5 | 71.8 | 79.3 | 36.0 |
| 1+2 | 75.6 | 83.4 | 82.8 | 81.8 | 83.5 | 85.2 | 56.7 |
| 1+3 | 71.0 | 80.4 | 80.0 | 79.1 | 79.1 | 84.5 | 50.2 |
| 1+4 | 78.7 | 86.1 | 87.6 | 86.9 | 80.7 | 85.7 | 57.7 |
| 2+3 | 78.9 | 85.8 | 86.2 | 85.5 | 85.8 | 89.5 | 64.4 |
| 2+4 | 83.2 | 89.1 | 90.7 | 89.6 | 85.0 | 88.7 | 66.0 |
| 3+4 | 80.4 | 86.8 | 88.4 | 87.8 | 83.3 | 88.5 | 63.0 |
| 1+2+3 | 80.1 | 86.5 | 87.5 | 86.8 | 86.2 | 89.5 | 65.7 |
| 1+2+4 | 85.2 | 90.4 | 92.3 | 91.6 | 86.5 | 89.6 | 69.7 |
| 1+3+4 | 81.6 | 87.7 | 89.4 | 88.7 | 84.0 | 89.1 | 64.8 |
| 2+3+4 | 86.7 | 91.2 | 93.1 | 92.4 | 88.4 | 91.4 | 73.8 |
| 1+2+3+4 | 87.1 | 91.4 | 93.4 | 92.8 | 88.6 | 91.7 | 74.5 |

*Tab. 3: Coverage Figures for Test Configurations and Their Combinations*



| Statistics of an OSS Package | | |
|---|---|---|
| Files (.c) | | 63 |
| Lines-of-Code (unfiltered) | | ~ 32.000 |
| Functions | | ~ 750 |
| #blocks | | ~ 8.000 |
| #decisions | | ~ 2.000 |
| Data Range Checks | | ~ 13.000 |
| Index Checks | #checks | ~ 500 |
| | #affected files | ~ 50 |

*Tab. 4: Statistics of an OSS Package with Data Range Checks*

Tab. 4 provides figures of an OSS package for which the data range and index checks were activated to demonstrate the high number of check points which were inserted. In addition to recording of data ranges, NaN and Inf events are recorded.

In case of an index-out-of-range event the index may be set to a valid value (default 0) or may be left unmodified by the instrumentation code. If such an event occurs for an dynamically allocated array or a pointer, the size of the array may be extended accordingly in case of unit testing, and the observed maximum will be reported.