How-to keep legacy simulation codes alive and continue developing them: A continuous integration process for a nuclear safety analysis code system

Joachim Herb, GRS
2019-05-14
Continuous Lifecycle London
Introduction to GRS

Gesellschaft für Anlagen- und Reaktorsicherheit (GRS) gGmbH is a non-profit organisation which deals with technical-scientific research and provides expertise in the fields of:

- reactor safety
- radioactive waste management
- radiation and environmental protection

GRS is Germany’s central technical safety organisation (TSO) in the field of nuclear safety and radioactive waste management.

Scientific staff: ~ 350
Annual turnover: ~ 53 million €

Development, validation, and application of computer codes for the simulation of thermal hydraulics, reactor physics, fuel behaviour, fission product chemistry, and structural mechanics.
Safety Architecture of Nuclear Power Plants

Fundamental safety objective
- to protect people and the environment from harmful effects of ionizing radiation

By fulfilling the fundamental safety functions
- reactivity control
- fuel cooling
- confinement of the radioactive materials

Deterministic safety analyses
Among other things, compare simulation results for postulated initiating events, e.g.
- Loss of coolant accidents
- Loss of off-site power
- …

with more specific criteria, e.g.
- temperature of fuel cladding below 1200 °C
Thermal Hydraulics Systemcodes

Purpose
- Used in **licensing process** (new builds, plant modifications, periodic safety reviews)
- by **constructors, operators, TSOs, licensing authorities** to demonstrate compliance with regulatory requirements
- Simulate the dynamics of the **coolant, fuel, structures, control systems**
  (temperature, pressure, phase state, …)
- Simulate different operational states of plants:
  Normal operation, accidents, core melt, …

History
- **Internationally**, different codes (RELAP, CATHARE, TRAC, ATHLET, …)
- Developed by different organisations since the **1970ies**
Properties
▪ 1D lumped parameter model
▪ Fast (but not real-time)
▪ Huge validation basis
▪ Simulation of complete plant/experimental facility (including all components and control systems)

Components
▪ Thermo fluid dynamics
▪ Heat transfer and heat conduction
▪ Neutron kinetics
▪ Instrumentation & Control
▪ 3D module (large scale 3D effects)
▪ Melt behaviour
▪ Fission products’ behaviour
▪ Containment phenomena
GRS System Code: ATHLET, ATHLET-CD, COCOSYS (2/2)

- **ATHLET, ATHLET-CD, COCOSYS** (and their ancestors) have been developed since several decades

- **Legacy codes**
  - oldest parts of the codes from the 1970ies
  - several generations of developers
  - different standard versions of FORTRAN
  - different software paradigms (e.g. encapsulation of data)
  - code is highly interwoven/dependent
  - different software engineering methods (e.g. source code management)

- **File format** of input data of test simulations changes over time
Ongoing Development

- Simulation of **new and innovative reactor concepts**
- Modelling of **phenomena of new interest**

⇒ **Coupling** of existing codes into code system AC² and further development

- Guidance and good practice set out in IAEA SSG-2
- Verification and validation of each new or modified code version
- Continuous integration is **standard** in software development

⇒ **Tools required to continuously check the quality and generate reports** about the current quality status

---

Example for Generation III+

Creative Commons license - Attribution 4.0 International (CC BY 4.0) [Xing et al., HPR1000: Advanced Pressurized Water Reactor with Active and Passive Safety, Engineering. 2 (1). doi:10.1016/J.ENG.2016.01.017]
**GRS Codes for Simulation of (Severe) Accident Analyses**

<table>
<thead>
<tr>
<th>Reactor Physics</th>
<th>Thermal-hydraulics / Severe Core Damage</th>
<th>Structural Mechanics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Containment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coolant Circuit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nuclear Fuel</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Diagram:**
- **Containment**
  - CoPoel
  - ANSYS CFX
  - OpenFOAM

- **Coolant Circuit**
  - DORT-TD
  - TORT-TD
  - QUABOX/CUBBOX
  - VENTINA
  - KMacs
  - SCALE
  - MCNP
  - DYN3D

- **Nuclear Fuel**
  - ATHLET-CD
  - COBRA-TF

- **Thermal-hydraulics / Severe Core Damage**
  - AC$^2$ Code Package
  - COCOSYS
  - ATHLET
  - MELCOR

- **Structural Mechanics**
  - Integral-Codes
  - ASTEC
  - FE-Codes
  - LS-DYNA
  - AUTODYN
  - PROST
  - WinLeck
  - ASTOR
  - code_aster
  - TESPA-ROD
  - ANSYS Mech.
  - FRAPCON

---

How to keep legacy simulation codes alive and continue developing them, Continuous Lifecycle London 2019-05-14
Requirements for a Continuous Integration Environment (1/2)

IAEA Specific Safety Guide No. SSG-2

Deterministic Safety Analysis for Nuclear Power Plants

- “all activities that affect the quality of computer codes should be managed”
- “best available software engineering practices that are applicable to the development and maintenance of software critical to safety should be applied”
- “procedures should be implemented to ensure that the code correctly performs all the intended functions and does not perform any unintended function”
- “the tracking of errors and reporting of their correction status should be a continuous process”
- “new versions of codes are developed, an established set of test cases should be simulated”
Requirements for a Continuous Integration Environment (2/2)

- **Efficient approach** to reduce manual actions by developers of all three codes ATHLET, ATHLET-CD and COCOSYS
- **Invest** more time in the **analysis** of simulation results
- **Define tests** (for modules, single effects, integral behaviour)
- **Build** of code triggered **automatically**
- **Execution** of **test cases**

- For all tests
  - Define **test target bands** or **limiting values**
  - Compare results of the test runs with targets/limits
- **Inform developers** if test deviations are detected

⇒ **Immediate actions** possible to resolve any problems
Tests to Support Verification (1/3)

“Verification of the code design should be performed to demonstrate that the code design conforms to the design requirements.” [SSG-2]

Unit Tests
- Method: Call **individual subroutines** or **functions** of the code
- Input
  - **Parameters** of the subroutine or function
  - Test setup (aka “**test fixture**”) to make sure that dependencies (e.g. module variables, files) are set up correctly
- Test criteria: **expected results** of subroutine/function
  
- Hard to implement for existing code due to dependencies
- Require complicated test setup
How to Test „Plots“ of Simulation Results

Different simulation runs can result in different time steps

**Algorithm**
- Select time steps falling into one segment of limit curve
- Calculate $z$-component of cross product
- Sign of cross product indicates if below/above limit curve

**Implementation**
Use **pytest** framework
- Read limits from JSON or HDF5 files
- Check limit
- Generate XML test reports and graphs
  ⇒ Report test failure (with time of violation)
Tests to Support Verification (2/3)

Single Effect Tests

- Method: Simulation using whole code
- Input: Input file for simulation
- Test criteria
  - **Limiting curves** for selected simulation results
  - Based on analytical calculations or single effect experiments
  - Need to be specified by developers

Test result shown in graph
Tests to Support Verification (3/3)

Regression Tests

- Method: Simulation using whole code
- Time steps of simulation
  - determined dynamically during run
  - sensitive to changes in physical modelling
- Input: Input file for simulation
- Parse output file of simulation
- Test criteria: **Number of internal time steps** needed by last test run

Test Case: sample1.Release.x64.5019.out

<table>
<thead>
<tr>
<th>Build number for current version</th>
<th>TOTAL NUMBER OF TIME STEPS</th>
<th>TOTAL NUMBER OF REDUCED TIME STEPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>5019</td>
<td>447</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build number for last version</td>
<td>TOTAL NUMBER OF TIME STEPS</td>
<td>TOTAL NUMBER OF REDUCED TIME STEPS</td>
</tr>
<tr>
<td>5005</td>
<td>469</td>
<td>98</td>
</tr>
</tbody>
</table>

TEST WITH DIFFERENT TIME STEPS
Experimental results of integral tests are used for validation. But they cannot be used directly to derive the limit curves:

- Boundary conditions and test procedures associated with uncertainties or might be incomplete.
- Problems with the measurements or unknown or undocumented effects of the experiments (e.g. local heat losses).
- It is not possible to fully replicate all measurement results by simulation.

⇒ Expert judgment is needed in the comparison of the experimental and simulation results.
Each previous release of the AC² component ATHLET was accompanied by a validation report.

At the time of the release, the code version had passed all internal quality assurance measures.

⇒ Simulation results of these releases can be considered as validated assuming no qualitative changes in the code models.
Selection of Test Criteria for Integral Tests (3/4)

- The data of **four integral tests** are used
- **Experimental** results not used for limit curves but **included** for **manual checks**
- 17 different simulations runs of past and validated ATHLET releases
Determine Enveloping Curves of Old Simulation Runs

- Segments of 1 s
- Convex hull of each segment
Selection of Test Criteria for Integral Tests (4/4)

- 17 different simulations runs of past and validated ATHLET releases
  - Upper/lower parts of the convex hulls are combined to build the upper/lower limit curve
- ~ 800 upper and lower limit curves generated automatically
- For comparisons with simulations limit curves are relaxed by 2 % (peak-to-peak/duration)
- Limit curves are included in the source code management system
Integral Tests to Support Validation

Analysis of integral tests’ results

- **Compare** simulation results with limit curves
- **Violations detected** by test software
- **Reported** to the software developers: are deviations intended and acceptable?

⇒ If yes by expert judgment, update limit curves

- **Procedure** does not by itself ensure consistency with experimental results
- **It cannot replace the ongoing validation** of AC² but supports the quality management during the code development process
- Simulation results can be used as input for the next validation report
Architecture of Continuous Integration Environment

The Jenkins Platform

- Describe job in a domain specific language
- Job specification becomes a part of the source code
- Wide range of plugins available, e.g. checking-out of source code, collecting test results, generating reports, notifying developers
Build Pipeline

Each job consists of a **pipeline** with **stages**

- Checking out source code
- Compilation
- Testing
- Documentation
- Integral Testing
- Notification

Jobs are parameterized, e.g. revision of the source code, build parameters

Two servers for Linux/Windows using same code base for pipelines and builds
Test Stages

- Use Python handler scripts
- Can be run locally by developers
- Use JSON configs to specify tests for different build configurations
- Run solver
- Execute tests
- Produce
  - XML xUnit outputs
  - HTML test reports
  - Graphs

```json
{
  "daily": {
    "run": [
      {
        "FC": "ifort",
        "CFG": "Release",
        "ARCH": "64",
        "test_script": "run_mini_tests.py",
        "script_args":
          "-E PWR -t . -s test_specs -r <RID> -i <INPUT> -e <BINARY> -t <TEND>"
      },
      "run_cmd": "<PYTHON> <test_script> <script_args>"
    ],
    "test_setting": {
      "WORKDIR": "Samples/PWR",
      "INPUT": "PWR.in",
      "TEND": "50"
    }
  },
  "nightly": {
    "validation": [
      {
        "FC": "*",
        "CFG": "ompRelease",
        "ARCH": "*",
        "test_script": "run_integral_tests.py",
        "script_args": "-e <BINARY> -t . --RID <RID> --RID-compare <RID_compare>"
      },
      "run_cmd": "<PYTHON> <test_script> <script_args>"
    ],
    "test_setting": {
      "WORKDIR": "validation"
    }
  }
}
```
Reporting of the Test Results (1/2)

Results of unit, single effect, regression and integral tests displayed by Jenkins in build reports

<table>
<thead>
<tr>
<th>Package</th>
<th>Duration</th>
<th>Fail</th>
<th>Skip</th>
<th>Pass</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>AthletRestart.apollo.integration_tests.athlet_restart</td>
<td>62 ms</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>iso2.apollo.integration_tests.athlet_run</td>
<td>48 ms</td>
<td>3</td>
<td>+3</td>
<td>68</td>
<td>+68</td>
</tr>
<tr>
<td>bolt_lpb1.apollo.integration_tests.athlet_run</td>
<td>0.46 sec</td>
<td>16</td>
<td>+16</td>
<td>760</td>
<td>+760</td>
</tr>
<tr>
<td>lsf_sbl18.apollo.integration_tests.athlet_run</td>
<td>61 ms</td>
<td>4</td>
<td>+4</td>
<td>142</td>
<td>+142</td>
</tr>
<tr>
<td>rosa_r916.apollo.integration_tests.athlet_run</td>
<td>0.17 sec</td>
<td>16</td>
<td>+16</td>
<td>241</td>
<td>+241</td>
</tr>
<tr>
<td>sample1.apollo.integration_tests.athlet_run</td>
<td>0 ms</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>+1</td>
</tr>
</tbody>
</table>
Reporting of the Test Results (2/2)

Build reports also include a HTML reports with graphs of all/failed tests
Conclusions

- It immediately **improved the productivity** of the software development process
  - detecting **build problems** (Linux/Windows)
  - synchronize the **development** of the different components of AC²
- **Testing tied** to everyday work of the developers
- Generate installation package for all components of AC²
- **Integral tests** run every **night** (instead of once in every few weeks)

⇒ **Longest feedback cycle one night**

- Significant **discipline required** to **react** to **failed tests** in **short time** and not to build up **technical debt**, otherwise
  - **additional failures** by further check-ins can go unnoticed
  - test system loses significantly in value

⇒ Continuous integration platform automates significant amount of work, but **developers must maintain** the **quality** of the code

- For more than three years the **continuous integration platform in everyday use**

**Future applications**

- More coupled test cases
- Use method to qualify full plant simulators
Acknowledgments

Main parts of the work described here have been supported by the German Federal Ministry for Economic Affairs and Energy in the course of the project RS1538

Supported by:

on the basis of a decision by the German Bundestag