Electronic Communications of the EASST
Volume 20 (2009)

Proceedings of the
Third International Workshop on
Foundations and Techniques for
Open Source Software Certification
(OpenCert 2009)

Open-DO: Open Framework for Critical Systems

José F. Ruiz and Cyrille Comar

12 pages

Guest Editors: Luis Barbosa, Antonio Cerone, Siraj Shaikh
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Open-DO: Open Framework for Critical Systems

José F. Ruiz and Cyrille Comar

AdaCore, 46 rue d'Amsterdam, 75009 Paris, France

Abstract: Critical systems development pushes software quality to the extreme. When human life depends on the correct operation of the software, strict processes are put in place to ensure, as much as possible, the absence of errors in the airborne system. These processes are very tool-demanding, and these tools also need to follow stringent and rigorous guidelines to provide the proper guarantees of quality. The Open-DO initiative aims at providing a framework federating open-source tools for safety-critical systems. A key point is that these tools will come with the material to ensure that industrial users can trust their output and use them to develop software compliant to the highest integrity levels.

Keywords: software engineering, free/open software, tools, certification, qualification

1 Introduction

In recent years, a significant trend has been a greater production and use of Free libre Open-Source Software (FlOSS). Improving and assessing quality of FlOSS products have become an strategic objective of FlOSS communities, with the goal of further increasing FlOSS adoption.

If FlOSS wants to expand in the high-integrity arena, new facets of software quality need to be adopted by developer communities. Safety-critical systems, in which human life depends on the correct operation of the software, require following strictly defined methods to ensure, as much as possible, the absence of errors. Existing standards for developing high-integrity software [Eur03a, Eur03b, RTC92, DEF97] assist developers in applying good software practices during the development, focusing on adequate definition and execution of software engineering processes and activities.

In the safety-critical world, the notion of certification has a very precise meaning which goes beyond what is typically covered by quality. In DO-178B [RTC92], certification is the process through which applications get official authorization to operate, which typically materializes as a strict guidance/audit of their development process plus the production of required verification artifacts.

Some of the required activities for certification are very tedious, human-labour intensive, and error prone, when no automatic tools are in place. For example, structural coverage analysis can be more efficiently and reliably performed by a tool than by a human, hence tools are extensively used to support this activity. These tools need to provide the proper guarantees of quality for the development of high-integrity systems.

The DO-178B [RTC92] certification standard defines the concept of qualified tools. Such tools have been verified to produce an output reliable enough to be used as evidence of a given certification task. Qualified tools can then eliminate, reduce or automate processes without its output being verified. To justify reliance on a tool, certification authorities require evidences that the tool meets precisely the operational requirements needed for the task it automates.

There are already some open-source initiatives targeting the development of tools for high-integrity systems, such as TOPCASED (Toolkit in OPen-source for Critical Application and SystEms Development) [top] for Model Based Development, and OPEES (Open Platform for the Engineering of Embedded Systems), which is a followup of TOPCASED focused on building an open-source ecosystem.

A new initiative, called Open-DO [ope], is being put in place to provide an infrastructure for DO-178 related activities. Beyond the production of FlOSS tools, a key point is the production and evolution of open qualification material so that these tools can be used in DO-178 processes.

There are also common software components, such as the operating system, where even competing companies can work together to share cost without compromising their competitive advantage. The objective in Open-DO is to develop certifiable components, which would come with the required artefacts to fit in the certification process.

Apart from tools and components, Open-DO aspires to provide more elements which would simplify critical development. Safety-critical standards require the production of many documents. The creation of templates to speed up document production and to make sure that the required elements are provided is one of the tasks in Open-DO. Another one is the development of education material to make training for safety-critical development easier and more widely available.

2 Certification process

Safety-critical standards focus on the adequate definition and execution of software engineering processes and activities. They define the objectives for the different processes, the activities that need to be performed for achieving these objectives, and the evidences that indicate that the objectives have been satisfied.

Software failures can affect differently system safety. The DO-178B certification standard defines five software levels according to the failure condition that can result from anomalous software behavior: failures in Level A software can cause catastrophic results, Level B could cause potential fatal injuries to a small number of occupants, Level C can impair crew efficiency or possible injuries to occupants, Level D corresponds to minor failures, and Level E would have no effect on aircraft operational capability or pilot workload.

Once a system safety assessment is done and the safety impact of software being developed is known, then the software level is defined. The software level has an impact on the effort required to show compliance with certification requirements. Level A software needs to fulfil 66 DO-178B objectives, Level B 65 objectives, Level C 57 objectives, Level D 28 objectives, and none for Level E.

The rationale behind certification standards is that the use of standard processes and compliance with pre-determined objectives help avoid the common pitfalls of software development. DO-178B defines the following main processes to be implemented:

- Software Planning. Determines what will be done to produce safe software conforming to the system requirements.

- Software Development. This process is broken down into four sub-processes:
  - Requirements. Defines the high-level requirements describing the functionality, performance, interface, and safety.
  - Design. Defines the software architecture and the low-level requirements from which source code can be implemented.
  - Coding. Implements the source code from the low-level requirements.
  - Integration. Develops the integrated airborne system by loading the executable into the target hardware.
- Software Verification. Detects and reports errors that may have been introduced during the software development process. Verification is not simply testing (it cannot, in general, show the absence of errors). It is typically a combination of reviews, analyses and tests.
- Software Configuration Management. Establishes the mechanisms to identify, control, and regenerate all certification artifacts. Appropriate problem tracking mechanisms are also part of this activity.
- Software Quality Assurance. Ensures the quality of the software by assessing the software life-cycle processes and their outputs. Ensures that all objectives for a given level are satisfied, and detected deficiencies are evaluated, tracked, and resolved.

Each process has inputs, outputs and transition criteria. The transition between processes are defined as the minimum conditions to be satisfied to enter a process. All these software life-cycle processes are linked and must be traceable.

There is an initial phase (Software Planning) where the rest of the processes are properly defined. Then, there is the Software Development Process that creates the final system. As we can see, the three other processes are transversal (integral) to software development: verification, configuration management, and quality assurance.

3 Tool qualification

Critical software development requires many tools to handle requirements, design, code generation, tests, and structural coverage, among others. Repetitive and human-labor intensive processes can result in errors as well as high costs. The DO-178B standards acknowledges the need for tools, but keeping safety in mind: tools can be used, but they need to provide the proper guarantees of quality.

The DO-178B certification standard defines the concept of qualified tools, which are those providing the required guarantees allowing to trust their output. Qualified tools can then eliminate, reduce or automate processes without its output being verified. Therefore, qualified tools must provide confidence at least equivalent to that of the processes which are eliminated, reduced or automated.

Tools are classified as either development tools or verification tools. Development tools produce outputs that become part of the final airborne system and thus can potentially introduce errors in the final system. Therefore, the rules for qualifying development tools are close to those for the airborne system. Verification tools are those that cannot introduce errors but may fail to detect them or mask their presence. Qualification criteria for verification tools are much simpler than for development tools, and they are roughly based on demonstrating that the tool fulfils its requirements under normal operational conditions.

Tools need to comply with very stringent quality criteria to ensure their usability when being integrated in a DO-178B development process.

4 The FlOSS advantage for critical systems

Free licensing guarantees complete freedom to inspect, modify, and maintain tools. This is really important because the life-cycle of safety-critical applications can typically be decades. It is hard to ensure that the tool provider will be in business for all that time, and to guarantee that they will keep their know-how is even harder. FlOSS can ensure that a new company may continue supporting an existing toolset.

Free licensing also guarantees that components embedded in the final critical system, such as the run-time systems provided by compilers, are available in source form, and can be inspected and modified as necessary for certification purposes. Openness means also that many eyes are potentially looking at it, increasing the likelihood of detecting problems and malicious code.

From an industrial community viewpoint, open-source components allow for a shared infrastructure which ensures long term viability, no dependence on project life-cycles, no dependence on company policies, and cost reduction by sharing their development and evolution among users with similar needs. It also allows some level of cooperation with competitors, and reduces training costs (specially for subcontractors).

Tool providers can provide a complete supported solution more easily when tools are shared, and open software tools offers an ideal showcase for open technologies. Open software creates an ecosystem where vendors can find potential customers and partners. Tool vendors ensure their income by offering support, training and expertise to adapt and evolve the products. Additionally, in large-scale critical projects, expert support and the insurance of not being blocked by tool misbehavior are extremely valuable and key for reducing risks of schedule shifts.

For certification authorities, those in charge of accepting or refusing the airworthiness certification of a system, an open platform would help them sharing and practising new ideas. It would also be an excellent vehicle for clarifying specific issues, and it would lower their training costs.

From the public institutions viewpoint, open software creates a very convenient bridge between academia and critical industries, and constitutes the most effective way to have a better return on public fund investment, since the results of grants and funded projects have a better chance to be reused and applied.

Therefore, there are many synergies to explore in the use of FlOSS for critical systems, and the Open-DO initiative is a call for action in this respect.

5 The Open-DO initiative

The free/open-source concept is being successfully adopted worldwide, and many companies have integrated FlOSS in their business models. Unfortunately, Opening only the source is not enough for addressing the needs of critical systems.

Safety-critical development involves many different processes, and can leverage on the integration of a number of tools covering, at least, the following categories:

- Life-cycle management
- Requirements management
- Modeling tools
- Software development tools
- Verification Tools
- Testing tools
- Build infrastructure
- Configuration management
- Traceability management

Open-DO is an initiative which is being put in place to provide an infrastructure for DO-178B software development, based on open-source tools. Many open-source tools already exist for handling some of these activities. For example, OSEE (Open System Engineering Environment) [ose] is an integrated application life-cycle management system, AUnit [aun] and JUnit [jun] offer unit testing capabilities, etc.

Open-DO aims at federating these tools, providing a framework integrating them, and addressing workflow support, which handles the ordering of activities, allocation of resources, transition criteria between activities and inspection of objectives.

Addressing the workflow is one of the objectives in Open-DO. This is one point where Open-DO approaches Lean [MS05] and Agile [Mar03] methodologies. Although these two approaches are not widely considered suitable for high-integrity systems, these concepts are actually being successfully applied for developing systems following the DO-178B standard. In a verification-driven life-cycle, where requirements-based testing is a must, the paradigms of test-first and continuous integration allow for appropriate refinement of requirements (from high-level to low-level requirements) and early detection of errors. Note that these requirements can be formally defined and subject to formal verification.

When planning the development, the software life-cycle is defined and described, including the ordering of activities, allocation of resources, and the transition criteria between activities. Verification activities (reviews, analyses, tests, . . .) ensure that the workflow is followed as it was planned, and that the required criteria for transitioning between activities are met. Tools can be used to track these activities, properly logging these events so everything is traceable.

The use of open software tools eases their integration and communication, allowing users to adapt, add or replace the different components, and permitting easier modification and tracking of the workflow.

Beyond the production of tools, a key point is the production of qualification material, so these tools can address DO-178 processes. Open-DO will host open qualification materials, as well as other certification artifacts such as validation test suites, etc. They can guarantee that the required quality criteria are met.

Apart from qualifiable tools, Open-DO can also provide more elements which would simplify critical development. As we have seen, DO-178B requires the production of many documents, and templates can be provided to speed up document production and to make sure that the required elements are provided. It is also possible to develop certifiable components, which would come with the required artifacts to fit in the certification process.

6 Life-cycle management

The planning process defines the means of producing software satisfying both the system and safety requirements. It defines the software life-cycle, which includes the inter-relationships between processes, their ordering, feedback mechanisms, and transition criteria between processes and activities.Verification activities (reviews, analyses, tests, etc.) ensure that the workflow is followed as planned, and that the required criteria for transitioning between activities are met. Tools can be used to track these activities, properly logging these events so everything is traceable. Handling this workflow is one of the objectives in Open-DO, encountering Lean and Agile methodologies. Although these two approaches are not widely considered suitable for high-integrity systems, these concepts are actually being successfully applied for developing avionic systems following the DO-178B safety-critical standard. In a verification-driven life-cycle, where requirements-based testing is a must, the paradigms of test-first and continuous integration allow for appropriate refinement of requirements (from high-level to low-level requirements) and early detection of errors.

OSEE (Open System Engineering Environment) [ose] is an official Eclipse project which provides an integrated application life-cycle management system developed by the Apache Team at Boeing. Among other features, it permits to import heterogeneous life-cycle artifacts (such as requirements, test cases, models, source code) into a common database, capturing all project data.

7 Requirements management

The first step in the development process is to get the system specification that defines what the system must (and must not) do.

Requirement management systems have so far had small attention from the open community. However, the situation is expected to change with the future release of the Open Requirement Management Framework (ORMF) [orm], an official Eclipse project. OSEE (see sections 6 and 15) embeds also a basic requirement management system.

8 Modeling

Modeling is rapidly gaining importance in the development process of critical systems: for instance, the next revision of DO-178, DO-178C, will include a full annex about model-driven design.

The most prominent open initiative within the modeling arena is the Eclipse Modeling Project, which collects a set of tools for the definition of (domain-specific) modeling languages and model manipulations (including code generation and model-based analysis): Eclipse implementation of UML has become the *de facto* industrial standard. In the high-integrity and embedded domain, TOPCASED has recently emerged as one major innovation vector within the European industrial research community. TOPCASED leverages on the Eclipse Modeling Project and provides an integrated collection of modeling environments in the form of an open Eclipse plug-in(s): at the moment of writing, it includes graphical front-ends for UML, SysML and AADL; the TOPCASED UML plugin is expected to became the official Eclipse solution for UML modeling[1]. In the area of synchronous modeling languages, Scicos [sci] offers an open environment for dynamic system modeling, simulation, analysis and code generation.

Open modeling initiatives are however not restricted to modeling environments only: the Gene-Auto [gen] consortium aims at providing an open framework for the development of qualified (in the DO-178B sense) multi-language code generators for synchronous languages such as Matlab/Simulink [mat] or Scilab/Scicos [sci].

9 Software development

One of the most prominent examples of a successful open-source development tool is Eclipse [ecl], an open-source framework which promotes the integration of different tools. Its openness encourages more projects to get involved, and it is boosting collaboration between third-party tools.

At the core of software development we find compilers. The GNU Compiler Collection (GCC) [gcc], which supports front ends for many programming languages and back ends for most architectures available today, has become one of the most popular compiler for the development of free and proprietary software.

Debugging is also an integral part of software development, and GDB [gdb], the reference open-source debugger, is an advanced and actively developed tool supporting native and cross debugging.

In addition to these tools, there are others that can help improving the quality of the software.

Most certification standards encourage the use of a coding standard that constrains language features and constructs to a well-defined subset. This approach facilitates safety analysis, avoiding error prone or hard to analyze features. There are open-source tools like GNATcheck [gnaa] or AdaControl [ada] for Ada. Code metrics are often used to evaluate the complexity and help understanding the structure of the source code. The GNATmetric tool [gnac] for Ada and CCCC [ccc] for C and C++ are part of the open-source tools available for that.

These are examples of open-source tools useful for safety-critical development among many others. The goal of an open framework, such as Open-DO, is to make the integration of additional

[1] http://wiki.eclipse.org/MDT-Papyrus-Proposal

tools easy.

10 Static analysis

Static analysis techniques, when applicable, provide worst-case information that can be used to derive properties that will hold under any operating conditions. They are very valuable in safety-critical development since they help establishing formal guarantees about resource availability (memory and timing).

Stack usage can be analyzed with GNATstack [RBHC07], an open-source static stack analysis tool that constructs the full call graph for a given application, annotated with local stack requirements, and then it extracts its worst-case stack usage. This information can be used as evidence of stack overflow avoidance for the certification of high-integrity and high-reliability applications.

Timing properties can be analyzed with Cheddar [SLNM04], an open-source real-time scheduling tool which verifies task temporal constraints of real time applications. It supports state-of-the-art scheduling policies, such as Rate Monotonic Scheduling (RMS) and Earliest Deadline First (EDF). When combined with a deterministic and analyzable tasking model, such as the Ada 2005 Ravenscar profile [VZP05], concurrency can be used in a reliable manner in mission-critical applications.

11 Dynamic analysis

Dynamic analysis techniques get program properties while running or simulating the application. Dynamic memory usage and structural coverage information are typically extracted using this kind of techniques. GNATmem [gnab] is an open-source Ada tool that obtains accurate dynamic memory usage history. It monitors dynamic allocation and deallocation activity in an executing program and extracts information about incorrect deallocations and possible sources of memory leaks.

The quality of a testing campaign is usually measured in terms of amount of structural coverage achieved. Project Coverage [cou] will produce a Free Software coverage analysis toolset, together with the ability to generate artifacts that allow the tools to be used for safety-critical software projects undergoing a DO-178B software audit process for all levels of criticality.

In the context of the French government “Competitive Clusters” initiative to encourage research, a two-years funding was awarded to the project, submitted as part of the Free Software thematic group.

The core idea is to analyze coverage from machine-level execution traces out of an instrumented execution environment such as a target microprocessor emulator. For common traditional processors, Qemu [qem] is used for this purpose. It is a reliable and efficient Free Software emulator that is being adapted to generate run-time execution traces. This allows very flexible non application-intrusive analysis on final target code with emulators running on development hosts. Additionally, coverage analysis can be performed on the object code that will run in the final production system.

Compliance with such certification levels represents an interesting challenge, in particular regarding the Modified Condition/Decision Coverage (MC/DC) criteria. Additionally, the advantage of doing object-level analysis is that it can be extracted both object-level and source-level (with the help of the compiler to allow for mapping source and object code) coverage information. Project Coverage will support, at the object-level, both instruction and branch coverage; at the source-level, statement, decision, and MC/DC coverage analysis will be available.

Unit testing, which verifies individual units of source code, will be supported by means of consolidating the coverage results from different executables exercising a given part of the code. Integration and system testing will also be addressed by combining the results of successive executions of a given executable with different input data.

Open-DO software components, as Project Coverage, will come with material to demonstrate that their outcome is dependable, and it will therefore allow qualification of the tools for use in a certification context, up to the strictest level for DO-178B. It opens the open-source concept to other artifacts than source code. An important goal of this project is to raise awareness and interest about safety-critical and certification issues in the free libre open-source community.

This tool targets safety-critical development above all, but it is obviously of interest to projects who want to incorporate a good software practice such as the evaluation of the quality of their testing.

Project Coverage is a representative example of tools that the Open-DO initiative will federate: tools fulfilling a certification activity (structural coverage analysis in this case), and whose software components and associated qualification material are going to be open. Industrial users will be able to modify it, to adapt it to specific requirements, and to reuse it. This is the open source philosophy brought to qualification material, and a radical shift for the high-integrity software community.

12 Testing

In safety-critical software systems, the testing strategy is driven by requirements, and depending on the level of these requirements, the type of test is different. For high-level requirements, which are derived from system requirements and system architecture, there is integration (functional) testing. For low-level requirements, which are successive refinements of high-level requirements so that source code can be directly implemented from them, unit testing is used.

FitNesse (Framework for Integrated Tests) [fit] is an open-source framework which provides the capability of easily defining acceptance tests and run them automatically. It addresses high-level functional testing, based on the definition of inputs and expected outputs.

There is a well-known family of open-source unit testing frameworks (AUnit [aun], JUnit [jun], etc.) whose intent is to facilitate test-first development by easing writing and running unit tests. These unit tests are well suited to verify that low-level requirements are correctly implemented.

13 Build infrastructure

Continuous integration is a software engineering practice in which individual changes are immediately tested and reported on when they are added to a larger code base, providing rapid feedback so that defects can be identified and corrected very quickly. This practice fits very well the Lean and Agile development methodologies, which encourage testing as early as possible, and safety-critical development, where there are clearly defined requirements which drive the development and the cost of late detection of problems is very high.

An example of an open-source light build bot is Savadur [sav], a client-server based application that can easily interact with configuration management tools to trigger new builds when commits are done.

14 Configuration management

Certification standards like DO-178B require configuration management of all life-cycle artifacts including requirements, design, code, tests, test results, and documentation.

Version control tools track the history of files. There are very widespread open-source tools for that, like Concurrent Versions System (CVS) [cvs] and its successor Subversion [sub]. They are extensively used both in open and proprietary software, and their flexibility allows for easy integration with other tools.

Other DO-178B configuration management processes, such as identifying, controlling and regenerating the different artifacts, and problem tracking, need to be addressed by Open-DO.

15 Traceability

Traceability is a critical aspect in high-integrity software development which is often addressed *ad hoc* by the development teams.

The goal is to be able to trace requirements to models, code, tests, test results, and coverage information, at least. Additionally, this end-to-end traceability needs to be extended following a process-oriented approach, adding workflow support to handle the ordering of activities, allocation of resources, transition criteria between activities and inspection of objectives.

The Open-DO initiative would like to address this critical part, which is required by safety-critical standards. The challenge is to create a useful and easy-to-use tool but flexible enough to cover the heterogeneous spectrum of software engineering practice in safety-critical systems.

One of the major open tools to manage traceability is OSEE, which defines a complete, end-to-end traceability model among artifacts covering the full life-cycle of the product.

16 Conclusions

The FlOSS community has the potential to extend the open development philosophy, which so far has addressed the source code, to open more than the code. It includes opening certification artifacts, such as qualification material for tools, validation test suites, etc. It would pave the way to FlOSS components for critical systems.

In the other direction, there are quality aspects that are addresses in safety-critical development in a very comprehensive and rigorous manner, that would be very valuable to increase quality in FlOSS. The high-integrity community can share the experience they have in producing code limiting errors, as much as possible.

The goal of the Open-DO initiative is to create an open ecosystem where the industrial community, tool providers, and public institutions can find the synergies to increase the productivity, and export high-integrity methodologies outside the safety-critical arena.

The cooperative Open-DO initiative is the meeting of three worlds: open source, Lean/Agile development, and high assurance certification. The objective is to implement and improve tools and methodologies to develop evolvable safety-critical software.

Open-DO is a call for action, waiting for new ideas to come. Information about the project will be posted at: http://www.open-do.org.

Bibliography

[ada] AdaControl. http://www.adalog.fr/adacontrol2.htm.

[aun] AUnit, Ada unit testing framework. http://libre.adacore.com/aunit.

[ccc] CCCC - C and C++ Code Counter. http://cccc.sourceforge.net.

[cou] Project Coverage: Free Software meets DO-178B. http://libre.adacore.com/coverage.

[cvs] CVS - Concurrent Versions System. http://www.nongnu.org/cvs.

[DEF97] Ministry of Defence. DEF STAN 00-55: Requirements for Safety Related Software in Defence Equipment. August 1997.

[ecl] Eclipse. http://www.eclipse.org.

[Eur03a] European Cooperation for Space Standardization (ECSS). ECCS-E-40B Space Engineering — Software. November 2003.

[Eur03b] European Cooperation for Space Standardization (ECSS). ECCS-Q-80B Space Product Assurance — Software Product Assurance. October 2003.

[fit] FitNesse (Framework for Integrated Tests). http://fitnesse.org.

[gcc] GCC, the GNU Compiler Collection. http://gcc.gnu.org.

[gdb] GDB: The GNU Project Debugger. http://www.gnu.org/software/gdb.

[gen] Gene-Auto. http://gforge.enseeiht.fr/projects/geneauto.

[gnaa] GNATcheck. http://www.adacore.com.

[gnab] GNATmem: Heap usage monitor. http://www.adacore.com.

[gnac] GNATmetric: The GNAT Metric Tool. http://www.adacore.com.

[jun] JUnit. http://www.junit.org.

[Mar03] R. C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall, 2003.

[mat] The MathWorks. http://www.mathworks.com.

[MS05] P. Middleton, J. Sutton. *Lean Software Strategies: Proven Techniques for Managers and Developers*. Productivity Press, 2005.

[ope] Open-DO. http://www.open-do.org.

[orm] ORMF (Open Requirements Management Framework). http://eclipse.org/ormf.

[ose] OSEE (Open System Engineering Environment). http://eclipse.org/osee.

[qem] QEMU: Open Source Processor Emulator. http://bellard.org/qemu.

[RBHC07] J. F. Ruiz, E. Botcazou, O. Hainque, C. Comar. Preventing Stack Overflow using Static Analysis. In *DASIA 2007 - Data Systems in Aerospace*. Naples, Italy, May 2007.

[RTC92] RTCA. RTCA/DO-178B: Software Considerations in Airborne Systems and Equipment Certification. RTCA, December 1992.

[sav] Savadur. http://repo.or.cz/w/savadur.git.

[sci] Scicos. http://scicos.org.

[SLNM04] F. Singhoff, J. Legrand, L. Nana, L. MarcÃl'. Cheddar: a Flexible Real Time Scheduling Framework. Volume 24(4). ACM SIGAda, December 2004.

[sub] Subversion. http://subversion.tigris.org.

[top] TOPCASED (Toolkit in OPen-source for Critical Application and SystEms Development). http://www.topcased.org.

[VZP05] T. Vardanega, J. Zamorano, J. A. de la Puente. On the Dynamic Semantics and the Timing Behaviour of Ravenscar Kernels. *Real-Time Systems* 29(1):1–31, 2005.