VALIDATION AND VERIFICATION OF MODULAR SOFTWARE FOR SOFTWARE-DEFINED RADIOS

Carlos R. Aguayo Gonzalez (MPRG, Wireless@Virginia Tech, Blacksburg, VA, USA; caguayog@vt.edu); and Jeffrey H. Reed (MPRG, Wireless@Virginia Tech, Blacksburg, VA, USA; reedjh@vt.edu).

ABSTRACT

The future of wireless technology relies on the flexibility provided by Software-Defined Radio (SDR). This technology enables the development of exciting new applications and promises drastic reductions in development costs. However, there are many challenges that must be overcome before the full potential of SDR can be reached. In particular, the appropriate validation and verification techniques are needed to ensure the quality of SDR systems and their software components.

This paper presents a framework for the validation and verification of modular software for SDR. It includes development and management techniques, tools, and metrics that facilitate quality assurance in SDR modules and systems. This approach enables agile development techniques, facilitating unit and integration testing strategies. We describe the features that make modular and SDR software testing unique and suggest techniques to address them, along with the tools and infrastructure required. Elements of this framework are applied to the SCA-based CIREN system being developed at Virginia Tech as part of the Smart Radio Challenge. The general principles, however, are applicable to any component-based architecture.

1. INTRODUCTION

In many aspects, software-defined radio (SDR) is shaping the future of wireless technologies. It provides a flexible platform that can support a myriad of new applications. It also delivers multiple benefits for equipment manufacturers by improving intellectual property reuse and technology insertion. As a result, more applications can be brought to market faster. Unfortunately, as communication systems get more complex, so is the software required to implement them. Developers are required to split the functionality into independent modules that can be assembled together. Developing stand alone software components allows not only the partitioning of complex systems into pieces of manageable complexity, but also the dispersed development of the different pieces. In order to deliver all the advantages of SDR, it is necessary to develop software components that are correct, robust, maintainable, and reusable. In other words, the software needs to be of high quality and developed with these principles in mind.

Developing quality software requires continuous validation and verification activities as part of a software quality assurance process. Verification is performed during development with the intent of steering the process so that the product and intermediate artifacts meet specifications. Verification answers the question: Are we building the product right? Validation is applied at the end of the development process. It checks the correspondence of the final product to user expectations. In other words, validation answers the question: Did we build the right product? [1].

Validation and verification activities take different forms, ranging from informal peer reviews to mathematical proof of correctness using formal methods, with testing being one of the most widely used. Testing is a process of technical investigation that increases the level of confidence that the product will behave according to the expectations. Unfortunately, testing cannot guarantee correctness for any system of meaningful complexity. Due to the size of the state space in software systems, it is virtually impossible to perform exhaustive testing on a system of practical complexity.

Testing can consume a large portion of development resources. It is estimated that about 50 percent of the total cost and development time of software projects is spent testing [2]. For large, critical systems this percentage is bound to be significantly higher. The earlier the testing process begins, the sooner problems and failures are detected, avoiding expensive consequences further into the development process.

For this reason, it is necessary to provide the tools and methodologies required to perform efficient testing of modular software for SDR. This paper presents a framework for the validation and verification of software components for SDR. We describe the features that make modular and SDR software testing unique and suggest techniques to address them, along with the tools and infrastructure required.

2. CHALLENGES IN MODULAR SOFTWARE TESTING

It is argued that the quality of software components increases with frequent reuse by revealing faults and errors through repeated use cycles, making further validation and verification activities irrelevant. The assumption of a competitive market, where only quality software prevails, further supports the argument [3].

However, this is not always correct. While a component may fulfill the needs of a particular user within a specific context, it may be unsuitable or unreliable for a different user with other needs in a different context. Therefore, it is still necessary to perform validation and verification activities regardless of the component's use in other projects.

Software components are usually specified and implemented with a specific application context in mind. Although they may be developed as reusable, general components, developers might make assumptions about certain aspects

of the application context, resulting in a context-dependent component. Similarly, assumptions are often made when designing test cases for them. The context for which components are developed and tested may not be the actual context under which they are used. This is more evident when components are reused across platforms and projects. This uncertainty makes it necessary to test components under the specific operational conditions where they will be deployed, despite their correct operation in other contexts. Needless to say, all modular components that are expected to be reused are required to be packaged with its unit tests.

When assembling applications completely or partially with reusable components, integration testing is a critical part in assuring the quality of the system. It checks module compatibility and can be seen as a procedural continuation to unit testing and builds from it. A fault detected during integration testing can be seen as a gap in unit testing or as incomplete or ambiguous specifications. Integration testing is crucial to detect certain faults that are not likely to be identified during unit testing such as:

- **Inconsistent interpretation of the specifications.** This may occur when each interpretation is reasonable but inconsistent (centimeters vs inches).
- **Violations of domains or capacities.** This can happen when some modules have implicit assumptions on ranges and capacities (e.g. buffer overflows).
- **Side effects on parameters or resources.** Can occur when a module uses a resource that is not explicitly mentioned in its interface. This usage may collide with another modules.
- **Missing or misunderstood functionality**
- **Non-functional problems.** Performance requirements can have a great impact when integrating SDR systems.
- **Dynamic mismatches.** This is specially important in frameworks that allow dynamic binding of modules.

3. CHALLENGES IN SDR SOFTWARE TESTING

There are many aspects of SDR that make their validation and verification a challenge. Not only do they present all the difficulties that radio and software development independently have, but also new complications due to the extreme flexibility. Furthermore, a fault in the analog RF or digital conversion modules, can trigger a sequence of events resulting on a failure on seemingly correct but unrobust software.

The real-time applications that SDR support impose severe performance requirements on the devices and software that implement them. Unfortunately, this is an area where performance improvements due to Moore's law are not going to help because the data rates of communication systems are increasing at least as fast as processing power. Therefore, the validation of performance requirements, either explicit or implicit, presents an important and challenging area in SDR development.

In order to deliver the required processing power, SDR platforms are often heterogeneous, combining general purpose processors(GPP), digital signal processors (DSP), and field-programmable gate arrays (FPGA) adding further failure points in their interfaces and control structures that need to be validated.

SDR platforms are usually very constrained in terms of memory, power, size, and weight, requiring optimized code which complicates the validation efforts of software modules. Although development of SDR modules is done in resourceful platforms, with plenty of memory and power to run simulations and code with debug symbols that make debugging easier, the final versions are optimized and striped from debug symbols. Optimized code can be very difficult to debug and validate. There are some common optimization techniques than can be implemented without platform-specific features, although this is sometime unavoidable when trying to achieve maximum performance. This creates a challenge when developing and managing modular software that is intended be reused.

As mentioned before, it is impossible to anticipate all scenarios that a software module will experience during its operational life. While developers do a great job at designing test cases for the initial use case, they are usually not sufficient or adequate for future use cases. Furthermore, SDR are expected to support multiple applications simultaneously adding extra uncertainty in terms of executional environment. A rule of thumb is that reusable software takes at least twice as long to develop than target specific code [4]. It may take longer to develop SDR modules that include some degree of optimization and are expected to operate in different platforms with constrained resources.

Adding to the validation and verification challenges, the communication standards keep evolving, adding more complexity and increasing the pressure to get devices to market faster. For this reason, SDR developers often have to deal with incomplete and changing specifications, which increases the risk of some faults getting through the testing process and affect the final product. Furthermore, test vector generation may be difficult for SDR. Test scenarios can get quite complex; for example, in order to test the receiver path of a radio, it may be necessary to create a flexible transmitter that provides enough control over the transmitted signal to inject faults and generate specific scenarios. This may become a really challenging task in some cases.

4. A FRAMEWORK FOR SDR TESTING

The software industry is utilizing agile development techniques to improve resource utilization and reduce the risks related to incomplete and changing requirements. These agile techniques have their foundation on test driven development, early and continuous delivery of software, and preparedness for requirement changes, among other principles [5]. However, they have not found widespread acceptance within the SDR community. This is mostly because of the need to optimize code for deployment on embedded platforms and the traditional tendency to follow the waterfall development process, with well defined specifications.

It is possible to follow agile development techniques in SDR if the correct tools are available. One of the most important tools required is an automatic test framework that supports software developers and increases the chances of success in the adoption of agile techniques. Based on the material covered in previous sections, and previous development experience, we identified the requirements for an automatic SDR test framework:

- Unified unit, integration, and system testing
- Standardized test case description
- Automatic execution of test cases
- Automatic oracle and reference implementations
- Automatic test result and coverage report capabilities
- Support for correctness, performance, and characterization tests
- Support for generation of test vectors
- Profiling support
- Interface with test equipment
- Debug support

There are numerous unit test frameworks for different languages and with different philosophies such as JUnit [6], CppUnit [7], CUT [8], Check [9], and cxxTest [10]. This last one is currently used in the Open-Source SCA Implementation::Embedded (OSSIE) [11] project to define and execute unit tests for signal processing components. There are also tools developed to verify compliance of systems and modules with specific architectures and specifications such as JTAP and WTT [12] for the Software Communications Architecture (SCA) [13].

However, it is desirable to have a single toolset to support validation and verification activities at all levels, from unit to system testing, to ensure consistency and a reduction in the learning curve, while spreading the maintenance/development costs across the duration of the project(s). In order to achieve this universality, it is necessary to define a unified way of describing test cases. XML descriptors are a favorite because of its platform independence and wide availability of support tools. XML is also the language of choice for other descriptors in open, popular architectures such as the SCA. This description of test cases must include:

1) Operational parameters
2) Executables
3) Inputs
4) Comparison type
5) Expected outputs
6) Reference implementation if available

We know that test vector generation can be quite complex in SDR. Fortunately, SDR developers are often familiar with simulation software such as MATLAB. This software is widely used by the SDR community to simulate communication systems and to perform algorithm analysis before the actual software implementation is developed. These MATLAB simulations can be used for testing purposes in two different ways: one is to generate the test vectors required to exercise specific aspects of the implementation and as a reference implementation required by a comparison test oracle. This approach has been followed to test some signal processing components of the Cognitively Intrepid Radio Emergency Network (CIREN), an ongoing development effort at Virginia Tech as part of the Smart Radio Challenge to develop an agile sommunication system that automatically establishes data links in the licensed family radio service (FRS) [14] band to support first response teams following major disasters, while avoiding primary users and interference.

The general operation of a comparison oracle is shown in Fig. 1 where the goal is to find an input combination that yields different outputs from both modules, setting the output of the XOR to "true" indicating an implementation error. Note that the reference implementation does not have to be more reliable than the module under test, as long as they are developed independently so it is easy to identify the one that is in error in case of discrepancies. In order for this to be practical, it is necessary to handle the type conversions (e.g. double-precision floating point to unsigned integers) and automatically interface MATLAB-generated vectors with the target platforms.

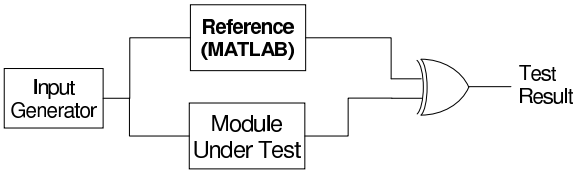

Fig. 1. General comparison oracle

It is not always straight forward to apply the pass/fail criteria for SDR modules. Many have a range in which their operation may be inconsistent but still considered correct. For example, the performance of a synchronization module, phase-locked loop, may have slight variations in its performance, depending on the noise applied to the input, and still be operating correctly. Because SDR modules often display this kind of behavior, it is necessary to provide support for performance and characterization tests that execute and analyze the results of hundreds or thousands of similar test cases and provide a measure for the correctness of the system.

One of the most important features in a test framework is the automatic execution of test cases. While the identification of test cases is a highly intellectual task which requires creativity, experience, and lateral thinking, the execution has to be as mechanical and automatical as possible. The less error-prone manual interaction is required to execute a set of test cases, the more likely the test cases will be executed as often as they are required. This is especially true for development efforts where part of the specifications are changing and constant regression testing is required. Furthermore, automated test execution is one of the critical requirements of agile software development, which relies on test-driven development. To automatically execute test cases, a generic way of instantiating the test cases (drivers) and an automated way of applying the pass/fail criteria (oracles) are

necessary. We already talked about some of the requirements for automatic oracles and how we can use MATLAB for this. Test drivers provide the means to invoke the software under test with the appropriate execution environment. This may be simple, as in the case of stand-alone components such as SCA resources, where it is only necessary to instantiate the component and feed the inputs. On the other hand, there may be other component models on which significant scaffolding is required to instantiate the modules, drive the tests, and collect the results.

After the tests have been executed and the results determined, it is necessary to develop a report. Appropriate test reports can greatly simplify the selection and re-execution of failed test cases. There are other requirements in terms of reporting capabilities that need to be considered, one of the most important ones being test coverage. The metrics vary from project to project and can include lines of code executed, execution paths, loops, or multiple condition coverage [15]. Having a metric for the test effort is necessary in the software development process to establish test and development stages and better estimate the overall development effort. There are many tools and integrated development environments that provide code coverage mechanisms Texas Instruments' Code Composer Studio [16] and Microsoft's Visual Studio [17]. Unfortunately, these tools are not integrated into test frameworks and the collection of reports may be cumbersome.

Because of the stringent performance requirements and the constrained resources on platforms, it is necessary to provide profiling support in the test framework for both performance and memory footprint. While a specific count of clock cycles spent in a specific module is ideal, sometimes there is not such support for the target platform and developers have to settle for indirect metrics of performance such as percentage of execution time. Having profiling support is critical for efficient optimization of SDR software modules and applications. There are plenty of tools that help developers with code profiling, CCS [16] and oProfile [18] are good examples, but unfortunately they are not integrated into a test framework and they have to be executed separately. The same applies for memory profiling tools such as EXMAP [19].

While all the features described thus far facilitate the validation and verification tasks, there is another important feature that comes into play after the tests have revealed errors in the software: debugging support. Developers need to execute the test cases in a controlled environment with enough observability so they can trace the faults. While there are many debug environments that allow the step-by-step execution of programs, they are, again, not integrated into a test bed and their operation usually require a context switch from command line to graphical user interfaces, requiring much manual interaction and slowing the overall process. For more advanced component models, such as the SCA, it may be difficult to provide debugging support due to the distributed processing capabilities inherent to the architecture. In this case, other approaches shall be followed to provide insight into the system's operation.

Because we are ultimately dealing with the validation and verification of radio devices, interfacing with test and measurement equipment is required. With the tight integration between the analog RF modules and the digital domain, it is necessary to validate the effects of the software logic on radio emissions as well as the characterization of radio modems. To this end, controlled radio environments and spectrum analysis tools are required. Seamless integration between test equipment and the rest of the test framework is desirable and there are already examples of succesful attempts [20].

5. CONCLUSIONS

The specific features of the component model and development methodologies used in a SDR implementation will ultimately dictate the final requirements for a test framework. The scope of the project, the sensitivity and criticality of the application, and the resources available will ultimately define the sophistication of the test framework required. Critical applications, and markets where development agility is required, will justify greater investments in test infrastructure.

The more users share a component model and a standardized way to describe test cases, the easier and cheaper it will be to develop test tools that support the validation and verification tasks. At Wireless@Virgina Tech, as part of OSSIE and CIREN projects, we have taken significant steps towards the implementation of an integrated toolset for agile testing of SCA software components and waveforms. Our approach relies on cxxTest for unit test and custom tools to perform debugging and integration testing. Separate efforts have been done to integrate test equipment and to profile applications and framework modules. Further development is required to refine and complete the tools with the remaining features described in this paper.

So far there is no integrated, comprehensive test framework openly available that includes the aforementioned features. There are important efforts and developments in individual aspects such as unit test frameworks and profiling tools, including the work of groups such as the IEEE 1900.3 that is recommending practices for conformance evaluation of SDR software modules. The difficulty and importance of thorough testing in complex systems, such as SDR, is well understood by the software community. The availability of appropriate test frameworks and tools will ultimately accelerate development, thereby increasing the confidence on SDR technology.

ACKNOWLEGMENTS

This work was supported by Texas Instruments, NSF, and the Wireless@Virginia Tech Partners

References

[1] S. R. Rakitin, *Software Verification and Validation for Practitioners and Managers*, 2nd ed. Artech House, 2001.

[2] G. J. Myers, *The art of software testing (Revised and Updated)*. John Wiley & Sons, 2004.

[3] Beydeda, S. and Gruhn, V. (Eds.), *Testing Commercial-off-the-Shelf Components and Systems*. Springer, 2005.

[4] M. Pezze and M. Young, *Software Testing and Analysis: Process, Principles, and Techniques*, 1st ed. Wiley, 2007.
[5] R. C. Martin, *Agile Sofytware Development: Principles, Patterns, and Practices*, 1st ed. Prentice Hall, 2003.
[6] "JUnit Testing Framework," Available at: http://www.junit.org/.
[7] "cppUnit Website," Available at: http://cppunit.sourceforge.net/cppunit-wiki.
[8] "C Unit Test System," Available at: http://sourceforge.net/projects/cut/.
[9] "Check: A unit testing framework for C," Available at: http://check.sourceforge.net/.
[10] "cxxTest Website," Available at: http://sourceforge.net/projects/cxxtest.
[11] "Open-Source SCA Implementation::Embedded," Available at: *http://ossie.mprg.org*.
[12] "JTAP: Software-Defined Radio Certification Suite," Available at: https://jtel.spawar.navy.mil/products.asp.
[13] S. C. A. S. V2.2, *http://jtrs.army.mil*.
[14] "Family Radio Service," Available at: http://wireless.fcc.gov.
[15] P. C. Jorgensen, *Software Testing: A Craftsman's Approach*, 2nd ed. CRC Press, 2002.
[16] "Code Composer Studio IDE," Available at: http://www.ti.com/.
[17] "Microsoft Visual Studio," Available at: http://msdn.microsoft.com/.
[18] "OProfile - A System Profiler for Linux," Available at: http://oprofile.sourceforge.net/news/.
[19] "EXMAP Website," Available at: http://www.berthels.co.uk/exmap/.
[20] "Software Defined Radio: An Integrated Test Method for Designing Software Communications Architecture (SCA) Compliant Radios (Application Note)," Available at: *http://www.tek.com/Measurement/App_Notes/37_18369/eng/37W_18369_0.pdf*.