SIEMENS

Algorithm Validation Toolkit AVT2EXT

Algorithm Validation Toolkit

AVT2EXT

Implementation Plan

Revision 1.0

Last Change: April 30, 2010

Robert W. Schwanke
Fabian Moerchen
Philipp Hussels
Jie Zheng

History

Document History

Version/ Status	Date of Issue	Author	Change and Reason of Change / Change Request/CHARM
0.1 Draft		Robert W. Schwanke	Final draft prior to AVT2EXT release

History of released Versions

Version	Release date	Product Version

Table of Contents

1. Development Environment

1.1. Project Information

Project information is kept in a pair of GForge sites and a Mediawiki site maintained at Siemens Corporate Research. The public GForge site is used to

- Track feature requests and bugs
- Record weekly status reports and meeting results
- Distribute engineering code releases
- Distribute sample data
- Distribute non-Wiki documents
- Maintain version control over the implementation with SVN.

Access to the public GForge site is limited to the AVT development team, the SAIC contractor team, the NCI leadership team, and a strictly limited, designated set of early adopters.

The Mediawiki site is used to develop internal and external documents that are informal in nature or likely to need feedback from early adopters.

An additional, private GForge site is maintained for AVT documents that are Siemens proprietary.

Non-proprietary project information will be delivered to the NCI caBIG AVT wiki site and AVT GForge site at the conclusion of AVT2EXT.

Because the AVT team is drawn from several different organizations, each sub-team has its own internal tools, repositories, and practices that are outside the scope of this document. The sub-teams are:

- CMIV[1], located in Beijing, China.
- Database and Requirements Engineering, both located at Siemens Corporate Research[2] in Princeton, NJ.
- XIPHost, located at Mallinckrodt Institute of Radiology[3].

1.2. Development Platform

AVT2EXT is being developed on a variety of PC's running Windows XP, the Microsoft Visual Studio 2005 Professional Edition with Service Pack 1, the Eclipse software development environment, IBM's DB2 Express C database system, and

- **Java 6 SDK** preferably jdk1.6.0_10 or later from http://java.sun.com/javase/downloads/index.jsp
- **Ant-1.7.0** or later from http://ant.apache.org/bindownload.cgi
- **Tortoise SVN client** from http://tortoisesvn.net/downloads .

[1] Center for Medical Imaging Validation (CMIV), Corporate Technology, Siemens Ltd. China, 7, Wangjing Zhonghuan, Nanlu, Beijing, 100102, P.R. China
[2] Siemens Corporation, Corporate Research, 755 College Road East, Princeton, NJ 08540

[3] Mallinckrodt Institute of Radiology, Washington University Medical Center, 510 South Kingshighway Boulevard, St. Louis, MO 63110

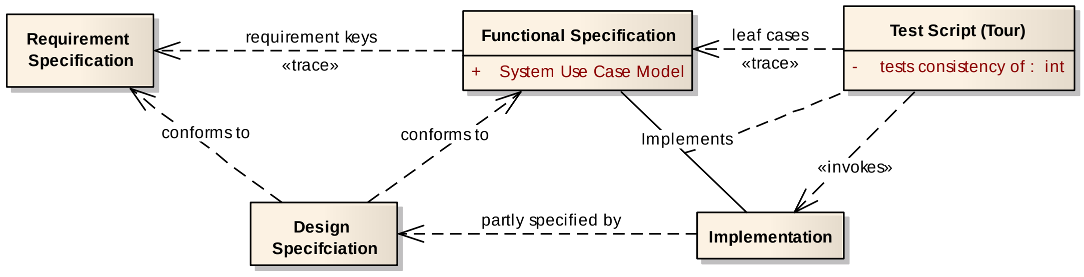
- **SVN command line client** from http://www.sliksvn.com/en/download for the automated build.
- **R package** strongly recommend to version 2.8.0 from http://cran.r-project.org/bin/windows/base/old/2.8.0/ .
- **Enterprise Architect**, for miscellaneous design models.

2. Iterative, incremental development

AVT is being developed using an incremental, iterative style of development, based on frequent integration, testing, and feedback to maintain a shared understanding of the project requirements. First we implement and integrate an end-to-end scenario, deploy it within the team for testing and feedback, then repeatedly augment it to cover additional use cases and their features and requirements, with frequent integrations to maintain consistency.

3. Key Development Documents

The following diagram describes the logical dependencies among key project documents.



- The Requirement Specification assigns a unique key to each system-level requirement.
- The Functional Specification consists primarily of a use-case category model, which decomposes the AVT functionality into finer grained pieces, associating the requirements to the use cases they constrain.
- The Test Script, a Wiki Tour, has a section for each leaf-level use-case category in the model, giving a sequence of user instructions that exemplifies that use case applied to a sample set of data. The whole set of these instruction sequences provides a coherent tour through all of the significant AVT features.
- The Implementation, of course, implements the Functional Specification. The Test Script invokes the implementation and compares the actual results to the expected results listed in the script, thereby testing consistency between the Implementation and the Functional Specification.
- The Design Specification only partially specifies the Implementation, addressing those topics where a document separate from the code was deemed helpful to the project. Whatever the DS does describe conforms to the Requirements Specification and the Functional Specification.

All of these documents are developed concurrently, according to the needs of the project. In particular, we seek to draft the test scripts as quickly as possible, as a practical way of agreeing on what the software ought to do in a particular context, and of assuring that the pieces all fit together to make a coherent whole.

4. Testing

Requirements specification, use-case modeling, and testing are all performed by the same engineer, who thereby has the best perspective on the importance of any bugs found, whether it is the implementation or the specification that should change, and how the overall progress toward the goal is going.

Each time someone finds a bug, they create a Bug Tracker item in Gforge, describing how to reproduce the bug. The Requirements Engineer (RE) reviews and refines the tracker item, assigning it to the most appropriate implementer for diagnosis and repair, and rating it for severity and priority. When the implementer has fixed the bug, he passes the tracker item back to the RE for validation. If the RE agrees that it has been fixed, he marks it **complete**, otherwise he documents the remaining flaws and sends it back for more work.

If the bug is of a type that could recur without being immediately obvious, the RE also inserts "re-test" instructions in the test script to check that the bug has not re-occurred.

5. Installation

The AVT build script (written in Ant/XML) creates a self-contained installer that isolates AVT from version-skew problems that it could have if it relied on separately-installed libraries.

6. Design approaches

For architecture and detailed design approaches, the reader is referred to the AVT2EXT Design Specification.

7. Experience

This development environment was very effective for supporting our distributed development team. We frequently found that bugs could be fixed within 24 hours because the tester and implementor were in widely different time zones, permitting nearly 24x7 engineering if needed. In particular, the self-contained installer saved a lot of time that would have been wasted on ruling out version-skew bugs. (In one particularly memorable case, the bug turned out to only happen on multi-core processors, but not on single-core processors. We detected this overnight by running exactly the same installer on machines in Princeton, St. Louis, and London. One machine was single core, the other two were dual core – and exhibited the bug.)