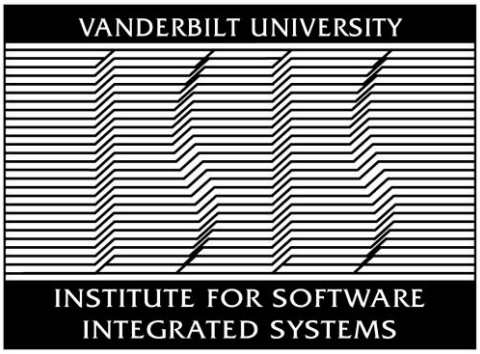


Institute for Software-Integrated Systems

Technical Report

TR#: **ISIS-15-111**

Title: **Software Quality Assurance for the META Toolchain**

Authors: **Ted Bapty, Justin Knight, Zsolt Lattmann, Sandeep Neema and Jason Scott**

This research is supported by the Defense Advanced Research Project Agency (DARPA)'s AVM META program under award #HR0011-13-C-0041.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Purpose

The purpose of this Software Quality Assurance Plan (SQAP) is to define the techniques, procedures, and methodologies that will be used at the Vanderbilt University Institute of Software Intensive Systems (ISIS) to ensure timely delivery of software that implements the META portion of the Defense Advanced Research Projects Agency (DARPA) Adaptive Vehicle Make (AVM) program.

1.2 Scope

Use of this plan will help assure the following:

(1) That software development, evaluation and acceptance standards appropriate for the META program interfaces to the AVM software are developed, documented and followed.

(2) That the results of software quality reviews and audits will be available for META program management and AVM program managers to support development, testing, and integration decisions.

(3) Important characteristics of the META program affecting quality, maintainability, and stability are documented for potential partners and customers.

2. Applicability

The SQAP covers quality assurance activities throughout all development phases of the DARPA AVM Meta Project. This plan represents efforts beginning at the end of the FANG-I program through the remainder of the Meta tool development.

While many pieces of the SQAP can be generalized to other ISIS software projects, the DARPA AVM program has unique requirements that require specialized attention, and limit our abilities to conduct a traditional QA process. The following list outlines some aspects of the AVM program related to our QA process.

- Research focused – The products generated at ISIS are typically cutting edge research products. These efforts involve technology that has not been developed, so, much of the software is produced by exploration or prototyping.
- Agile development is the approach we are following. Requirements are light. Interfaces are defined as needed in consultation with our partners. User Threads provide details of

the required functionality. Integration testing is where the bulk of the integration and user requirements are focused.
- Changing Requirements – Our process is designed to be adaptive as requirements change and evolve.

As the program matures and final products for delivery are assembled, the focus of the SQAP shifts in the following ways:

- R&D is no longer a driver. Maturing existing functionality and ensuring effective integration with team contributions is more important.
- Agile development continues with more focus on complete seamless execution of user threads.
- No new requirements are anticipated; rather fixing, testing, and validating the goals of the user threads is the key point.

3. Applicable Documents

The following documents can be used as requirements for the design and manufacture of ACIS software and form a part of this document to the extent specified herein. The issue in effect at the time of contract award shall apply unless otherwise listed below.

3.1 Contract Level Documents

META Contract and associated Contract Data Requirements List

3.2 ISIS Governing Documents

None applicable

3.3 Reference Documents

Broad Agency Announcement for Component, Context, and Manufacturing Model Library – 2 (C2M2L-2) From Tactical Technology Office DARPA-BAA-12-30 dated February 24, 2012 APPENDIX 1: PHILOSOPHICAL UNDERPINNINGS OF ADAPTIVE VEHICLE MAKE and APPENDIX 2: DEPICTION OF THE META-IFAB INTEGRATED TOOL CHAIN

4. Program Management, Planning and Environment

As the only management level document that is a META program deliverable, the SQAP also documents the basic management and planning functions performed in META.

4.1 The META SQA Plan

The META SQA plan is developed to provide META PM with the tools to deliver a robust product for use by industry partners, AVM teammates, and other DARPA/NASA projects. Data collection of metrics that characterize the software product and its quality are central to achieving that goal. Program management and development staff must understand what the metrics mean and take action accordingly. A portion of the project management discipline is expended to review project metrics to ensure the actions taken accomplished the intended results. Finally, the metrics collection, including defect burn down, is an important component of the final delivered open source project.

4.2 Organization

The organization with ISIS is project focused R&D activity. Aside from the PIs, there is little in the way of hierarchic structure. QA is not a diffuse activity however, since it remains a major accountability of the project management team. The team's organization structure is depicted in Figure 1.

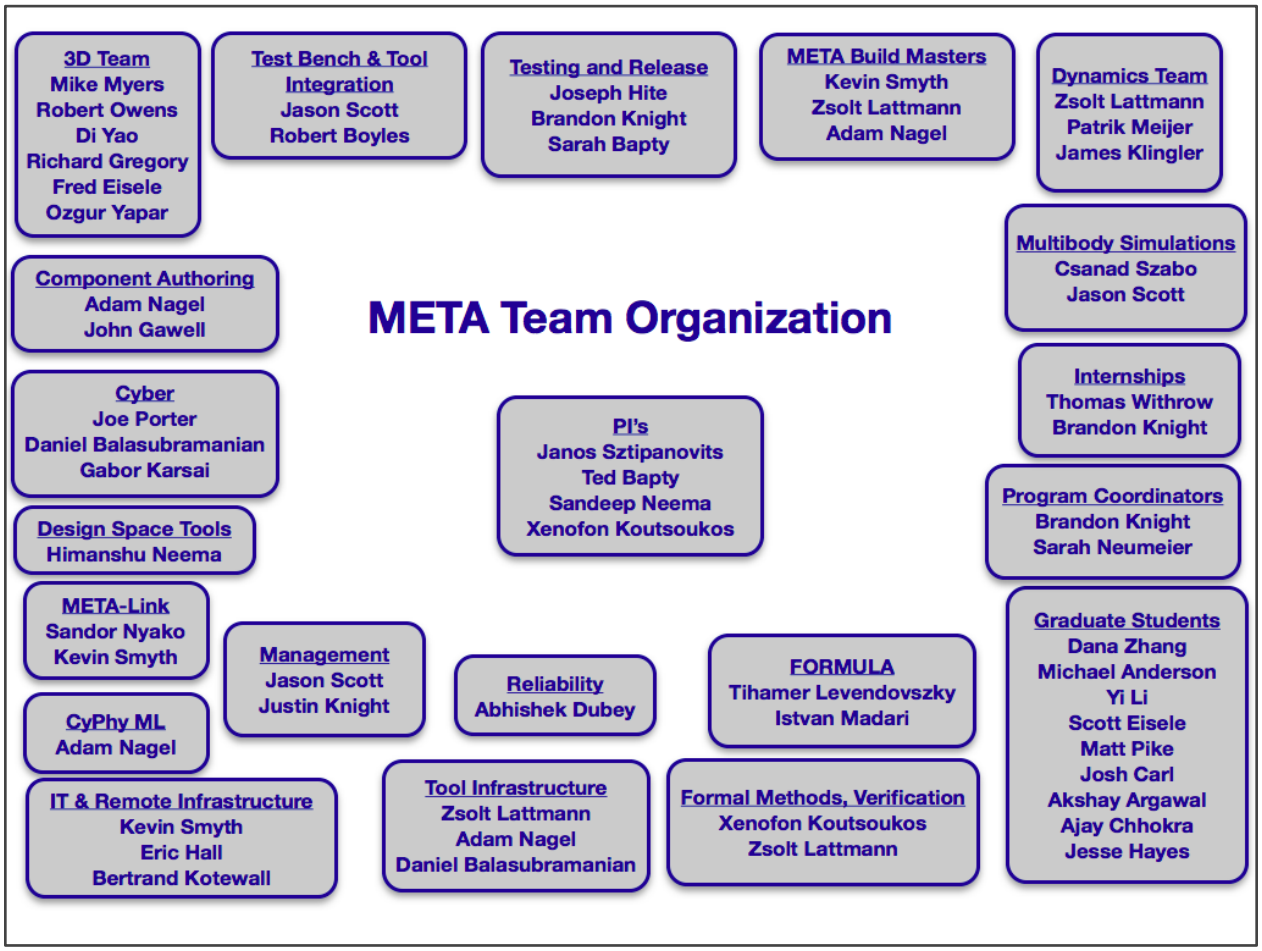


Figure 1: META Team Organization

4.3 Task Planning

Since the META program is being developed as an agile project, tasks are organized by a time box. Typically each spiral is about 4 weeks with 3 sprints, major weeklong iterations, and a final week for focused integration. Since the software is being continuously tested, the metrics are available on a weekly basis and reviewed on a monthly basis as part of the release decision process.

The Meta development plan is organized into four-week "Sprints" as depicted in Figure 2Figure 2: Spiral Development Cycle. At the end of each sprint, a new version of the CyPhy-Meta tool is released to the test community.

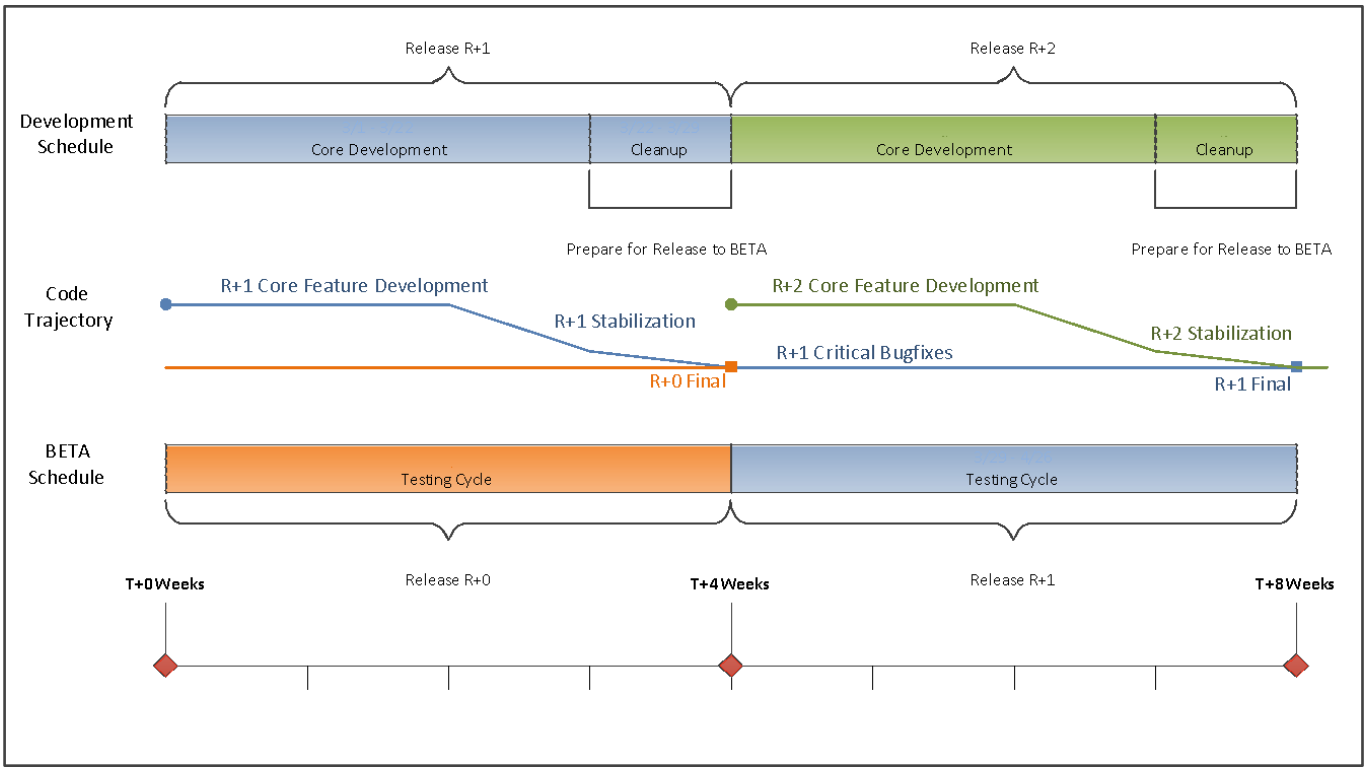


Figure 2: Spiral Development Cycle

These four-week sprints are divided into two phases: feature development and stabilization. The feature development phase is when the primary work of designing and implementing new features takes place. The stabilization phase emphasizes deeper testing, as well as the preparation of documentation for internal and external purposes, including BETA test scripts, example and test models, and software architecture documentation.

The *Spiral Development Cycle* diagram depicts the relationship between META team development cycles and BETA testing cycles. While the META team prepares release R+1, the BETA testing group is working with the previous release R+0. Once the META sprint for R+1 has completed, the tools are released to the BETA team for testing, while the Meta team moves on to R+2. During BETA testing of R+1, META tool updates will only be provided for issues that significantly block the testing process.

While the next version of the tools is being developed, the previous version is released to beta test. Figure 3 outlines the internal development cycle:

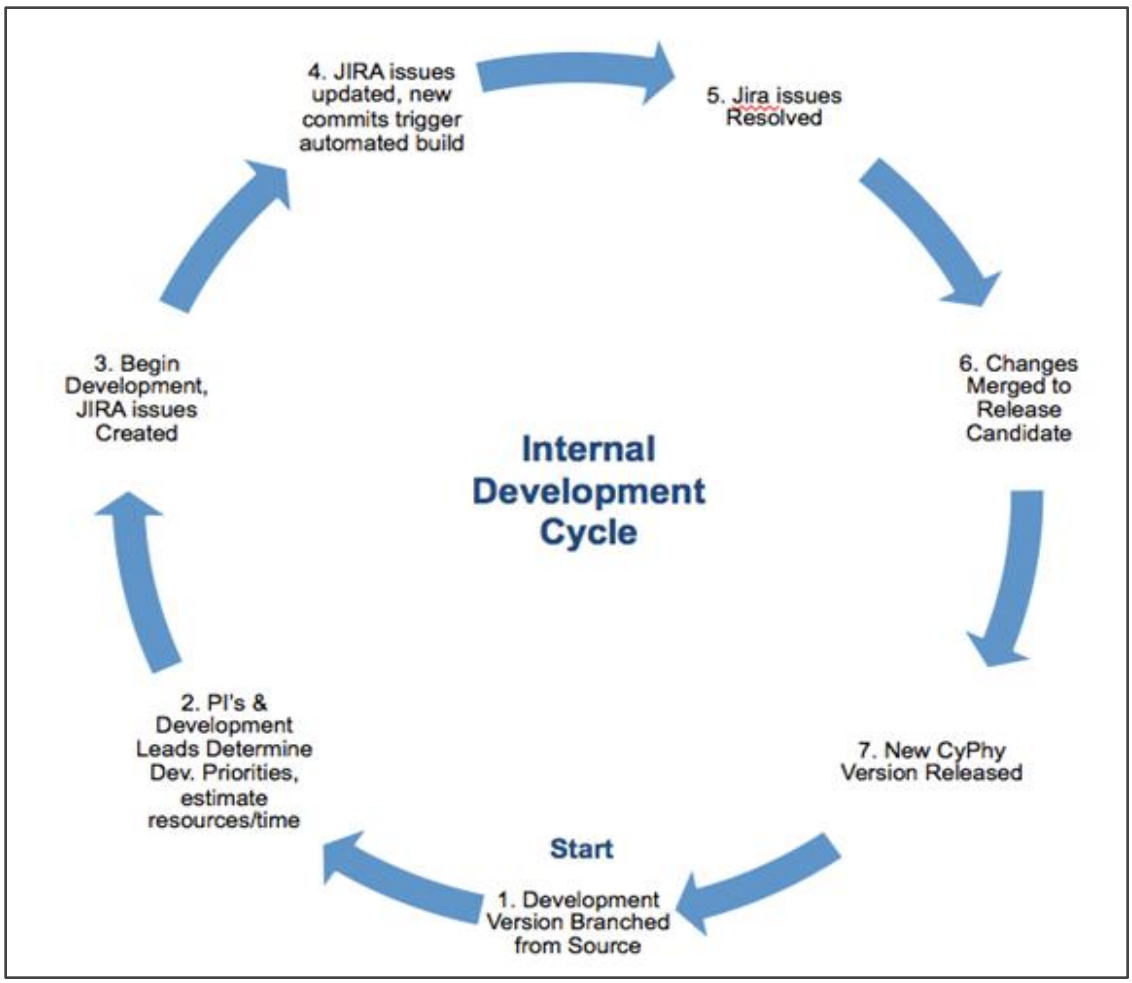


Figure 3: Internal Development/Testing Cycle

Each development group is responsible for collecting their own metrics and using it for planning their work. At the transition between sprints and in dealings with external partners, the project management team is responsible for identifying weaknesses based on the metrics.

4.4 Software Personnel Training

4.4.1 SQA Personnel

No training of SQA personnel is anticipated as the person tasked is the author of the SQA plan and he is also the project manager for META.

4.4.2 Software Developer Training Certification

Each member of the software development team has recent software engineering coursework including SQA. Project leads responsible for the largest chunks of META code have industry knowledge of code standards and quality assurance techniques. No certification is required.

4.4.3 META project management

The techniques and metrics identified in this document are evolving and should be used based on experience and gap analysis when QA levels are perceived to be dropping. The document provides an organizing framework for collecting, analyzing the data so management can take action and follow-up to verify results are in line with projections.

4.5 Tools and Environment

The high level architecture showing the relationships between Model Integration, Tool Integration and Execution Integration platforms is depicted in Figure 4.

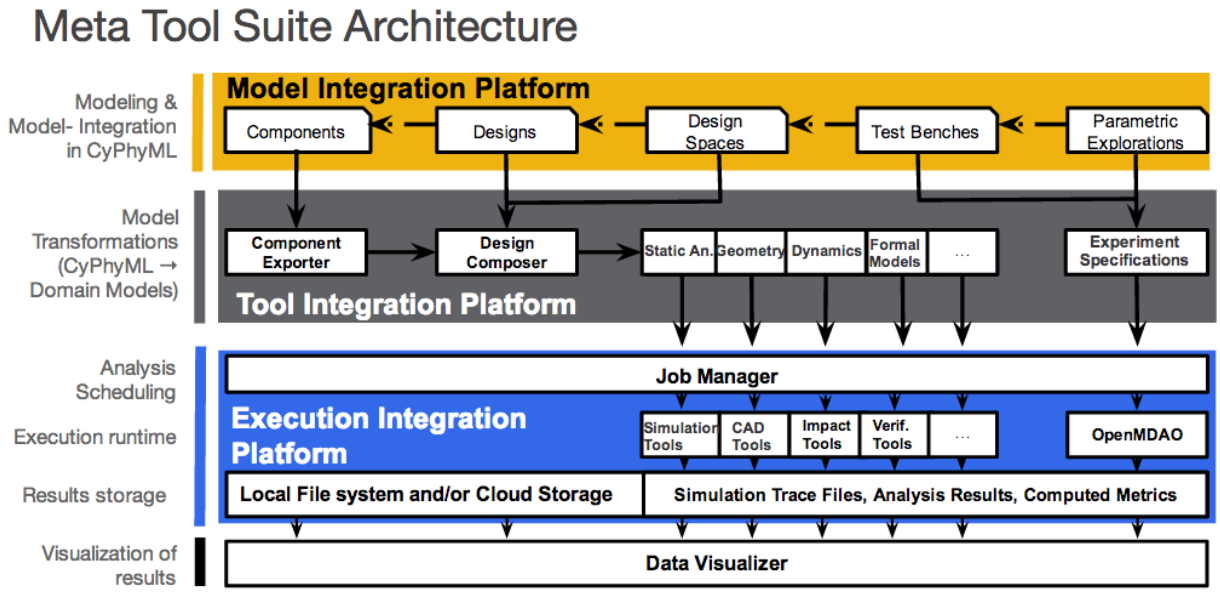


Figure 4: High Level Architecture Integration View

META generates tools for the design, exploration, specification, simulation, testing and manufacturing of complex reliable robust systems. It also relies upon software tools for development, synthesis, testing, analysis and reporting functions. The META toolchain entries listed in Table 1 are current as of the end of Post Gamma.

Tool or Model	Current Version	Dependencies (other tools and models)	Notes (criticality, test tools, etc)
GME	14.3.5		
Python	2.7.6		
Open Meta-CyPhy	14.03.25913	GME, Java, Python	
Java	7.0.550		Meta-Link
Dymola	14.0.294		Dynamics
ProE	Creo 2.0 M070		CAD
OpenModelica	1.9.1Beta2		Dynamics
OpenFOAM	2.2.0		CFD
Abaqus	6.13-1		FEA
Nastran	2013.1		FEA
SwRI AVM Tools	42	GME, Open Meta-CyPhy, LS-Dyna	Blast/Ballistics
LS-PrePost	4.1		Blast
LS-Dyna	7 (rev. 79055)		Blast
CTH	10.3		Ballistics
ForgePort	0.6.10.0	GME, Open Meta-CyPhy	VF tool
PSU-MAAT	1.0.33		TDP editor
PSU-HuDAT	1.0.8	Creo	
PSU-RAMD	2.2		
Mayavi	4.3.1	wxPython 2.8, configobj, Envisage	FOV
C2M2L Modelica Lib	R2838		
C2M2L Lib	11		
Seed Model	RC9		
FEA Seed	6.4		
Comp Spec	2.5		

Table 1: AVM Tools at the End of Gamma Testing

5 SQA Program Requirements

This section defines the SQA review, reporting, and auditing procedures used at VU/ISIS to ensure that internal and external software deliverables are developed in accordance with this plan and contract requirements. Internal deliverables are items that are produced by software development and then integrated with project builds using configuration control procedures. External deliverables are items that are produced for delivery to the META Community. These include scheduled program releases and final configuration deliverables.

5.1 User Threads as Requirements

During spiral planning, user threads to accomplish significant complex, chained tasks are defined. These provide insight to developers and testers about the intended use of META. Test cases are defined by external organization (Beta testers, and other AVM contractors) to examine the META behavior in completing the user thread activities. Each release has identified user threads as the major content.

5.2 Innovation and Improvement

ISIS as a learning organization has a practice of analyzing current conditions to seek improvement, a structured approach for implementing changes on a small scale that can be measured and a forum for broadcasting those changes and the results to the broader community. Innovation is important to upgrade processes and integrate appropriate technology. ISIS approach to innovation is to collect data to confirm where more effort is spent than value is generated. Using group discussions, different approaches to process change, tool addition, computer resources or other avenues are selected for pilot test. Once improvement is noticed, then the changed is propagated to other groups.

5.3 Program Resources Allocation Monitoring

Program resources that are planned and monitored include personnel, computer systems, and software licenses. At the start of each spiral, personnel are assigned to development and test teams. Institutional computer systems are assumed for the development life cycle. Software license needs may change for a variety of reasons, but adjustments are made to ensure the delivered configurations will operate with the appropriate software licenses.

5.4 Best practices of software development

A sample of practices that ISIS has applied to their agile development process to increase the quality of final products are listed below. Some of these approaches are a result of either process improvement activities or innovation.

Two sets of eyes on every change is a general principle followed during development. Pair programming, peer (code) reviews and commit reviews are all procedures used in order to follow this principle.

Pair programming is used occasionally to ensure that code is inspected by 2 people and knowledge is distributed within the team. This method works best in non-routine tasks (e.g. writing new code or debugging a complex algorithm), and sessions should not last for more than 4 hours. Pair programming is also a good practice to quickly bring junior team members on board.

Peer code reviews are conducted in order to verify the developed code is of good quality, implements the desired functionality, and will integrate well. While code reviews help achieve these goals, they will not substitute testing and other QA processes.

Commit reviews are done by a dedicated person who merges (integrates) the code. During this procedure the person screens the changes made before merging it to the repository main (release) branch.

Feature design documentation and reviews are recommended to ensure that the implementation will fulfill the requirements, and developers working on the same feature have a common understanding of the problem and the proposed solution. Before implementing a feature, team members should discuss it and write a 1-3 page document, which would be informally reviewed and accepted as the baseline for the work to be done.

Automated tools used for executing (code) tests and static code analysis are in place.
Automated test execution tools run tests as a part of the continuous integration process. The tests are executed each time a change is integrated to the release branch. The test results are accessible through a web interface. In case of a test failure, the person(s) who made the change are automatically informed via email. This allows immediate discovery of code breaks.
Static code analysis is performed on the source code to ensure adherence to coding style, standards, and to detect bad practices. The static code analysis is executed by the build system for each build. These results are checked periodically by the developers; any significant concerns are reviewed in greater detail.

5.5 Inspections

During each sprint within a spiral, code and design are informally reviewed within the development team. These results are often captured informally with the team leaders' notes. Prior to final delivery, a plan is generated by the project manager to identify the highest impact and highest risk modules for latent defects in the scheduled final delivery. These modules should be the subject of a formal design and code review. Recommended changes should be analyzed for likely impact in earlier deliveries.

- The first step is an analysis of the JIRA data base and Beta test results to find the most common types of defects. This list is part of the code review package.
- Second is the principal engineer for the modules being reviewed to provide annotations on what has changed, what test cases have changed to accommodate the code change, and whether any downstream dependencies have been identified.
- Third is the selection of the review team and chair. Date and time should be set for the review with guidance on how long the time period should be from the appendix on Review and Inspection Guidance.
- Once the review is complete, the team lead and the principal engineer should list all actions to be taken, identify latent defects, and identify additional testing as necessary. The final review team report should include the closeout of actions, testing, and defect removal and analysis.

5.6 Test Case Management

Test cases are developed for both unit testing and thread testing. Unit test cases should be successfully run before spiral integration. Thread test cases should be run during Beta Testing. The results of these test cases are used to define the META capabilities for each release.

The META Project Manager maintains a RYG status board of test bench status and META thread capability Status for each delivery. Typically these status charts are presented at the PI meetings or other venues for the META user community.

5.7 Defect Reports and Change Requests

Defects in the delivered products are documented by the open tickets generated by the Beta Test Team. Internally discovered defects are covered by the JIRA data base.
The most important metrics for management and reporting are:

1. Defect Rate: this is analyzed for module occurrence density, defect source, and time to closure.
2. Defect Insertion Ratio: this is the rate of defects as a result of fixes to other defects. This analysis includes source, missed opportunities to find or prevent the defect, and time to discover it after insertion.
3. Failure Interval is a valuable metric for measuring improvement in stability.

5.8 Software and Project Document Deliverables

The META project Manager is responsible for reviewing deliverable software documentation including the META Final Report. Review checklists will be used to review these documents.

These reviews will help ensure that documentation is in compliance with applicable contract instructions.

Final Report software documentation should include R&D results as well as product documentation. Important contents include: User Thread definitions and their status, user documentation for build, execution and test, and a High Level System Architecture Diagram describing vision and details of the architecture to the development team.

Software documentation must be based on some published convention such as found in IEEE Software Engineering Standards Source code commenting requirements should be spelled out in an appropriate appendix. Both Software documentation and comments are covered in code reviews.

5.9 Requirements Traceability

Traceability is identified through the use of a spreadsheet matrix which will tie individual Contract End Item (CEI) deliverables and document entries to lower level entries. These Traceability products are produced and maintained by the project manager.

5.10 Software Development Process

The META program is developed using an agile process. Control over spiral and sprint contest is established by consensus of the development team and its customers in the META/AVM community at kick off meetings each month. The project manager and team leads review progress during the month and evaluate test results, execution notes, and other teammate's analysis at the end of the period in build release decisions.

5.11 Project reviews

5.11.1 Formal Reviews

Given the R&D nature of the project and the agile method of development, formal reviews are minimized. Almost all decisions are the result of informal meeting or telecons with the extended AVM/META team.

5.11.2 Informal Reviews

Where the module interfaces with components generated by other organizations, there is an informal design review held with all organizations affected.
The project manager or team lead will ensure all action items generated during this review process are identified and tracked during development. The project management team is responsible for ensuring all action items have been closed.

5.11.2.1 Code Walk-throughs

Because of the wide range of languages and tools for auto-generated code, the Code reviews should be tailored based on recent experience for the languages and toolsets used. It is also important to highlight change history and defect history for the modules in review. Eventually, enough history in reviews will build up that bug classes will become important to consider.

5.11.2.2 Baseline Quality Reviews

This review ensures that: (1) the code has been tested and meets module specifications, except as noted; (2) that any changes to applicable software module design documents have been identified; (3) that appropriate validation tests have been run; (4) that the functionality of the baseline is documented. (5) that all software design documentation complies with this plan. (6) that tool and techniques used to produce and validate the Software Sub-System are identified and controlled.

5.12 Test Benches

Test benches represent environment inputs and composed models connected to a range of testing and verification tools for key performance parameters. Test benches work by composing a user designs and executing the designs in the environment specified in the Test Bench. Tests can range from Finite Element Analysis in the thermal and structural domain to multi-domain analysis using the Modelica language to Manufacturing cost and lead time analysis. The major components of a Test Bench are the workflow definition which defines which analysis tools should be used, the top level system under test which defines the user design(s) that should be tested, the environment inputs which are specific for each type of analysis, and the metrics of interests which will be used to compare designs against a set of requirements for that design.

Test benches offer user the ability to rapidly compose designs for a variety of different analysis by using one source model. The goal of this design is to allow users to create a virtual test environment once and run numerous designs through this environment without having to manually set-up each individual design. This also ensures that each design is subjected to the same environment to allow for the best possible comparison of designs. Ultimately the goal is that users spend less time setting up analysis and more time analyzing results to design the best possible product for the requirements.

Another goal of Test Benches is to enable rapid response to changing requirements of a design. Again instead of a user manually setting up one of a number of designs in a new environment to assess designs against new requirements, they can modify a few parameters in the Test Bench and begin the analysis of the designs with the updated environment in a matter of minutes instead of days.

5.13 Software Configuration Management

Software configuration management is the progressive, controlled definition of the shape and form of the software deliverables. It integrates the technical and administrative actions of identifying, documenting, changing, controlling, and recording the functional characteristics of a software product throughout its life cycle. It also controls changes proposed to these characteristics. As the software product proceeds through requirements, analysis, implementation, test, and acceptance, the identification of programs are identified in the SDP. This assurance process occurs during the Baseline Quality Review mentioned above as its configuration becomes progressively more definite, precise, and controlled. Software configuration management is an integral part of the project configuration management, using the same procedures for identification and change control that are used for documents, engineering drawings, test verification procedures, etc.

5.14 Release Procedures and Software Configuration Management

The need for control increases proportionally to the number of individuals that use the products of software development. As a result, different control procedures will be used depending on use. The ISIS software configuration management process revolves around the disciplined use of two tools: software version-control system (Subversion) and a ticket-based tracking system (JIRA). Figure 5 presents an issue summary page from JIRA which is typically reviewed while making work or release decisions.

All development tasks are tracked in the JIRA system. Each sprint is associated with a future software release version, with these versions making up the milestones used within the system. All development tasks are tracked with JIRA tickets, including new features, improvements, refactoring, and correction of defects. Each ticket includes a "Target Version," marked either with an upcoming milestone or with a "backlog" tag.

All work for a given ticket typically occurs in a Subversion branch dedicated only to that task. Once the task is completed and has passed alpha testing, the code changes are scheduled to be merged into the relevant software release lines. These dedicated branches create a clear definition of the changes related to a specific issue, allowing the team to reliably apply or remove them to the correct software release versions based on changing conditions, or defer changes to future sprints.

Branches are merged during a weekly "Merge Day". On this day, all completed tasks are merged to the relevant software release lines. The Subversion repository and the JIRA system are also reviewed for inconsistencies, and are corrected if necessary.

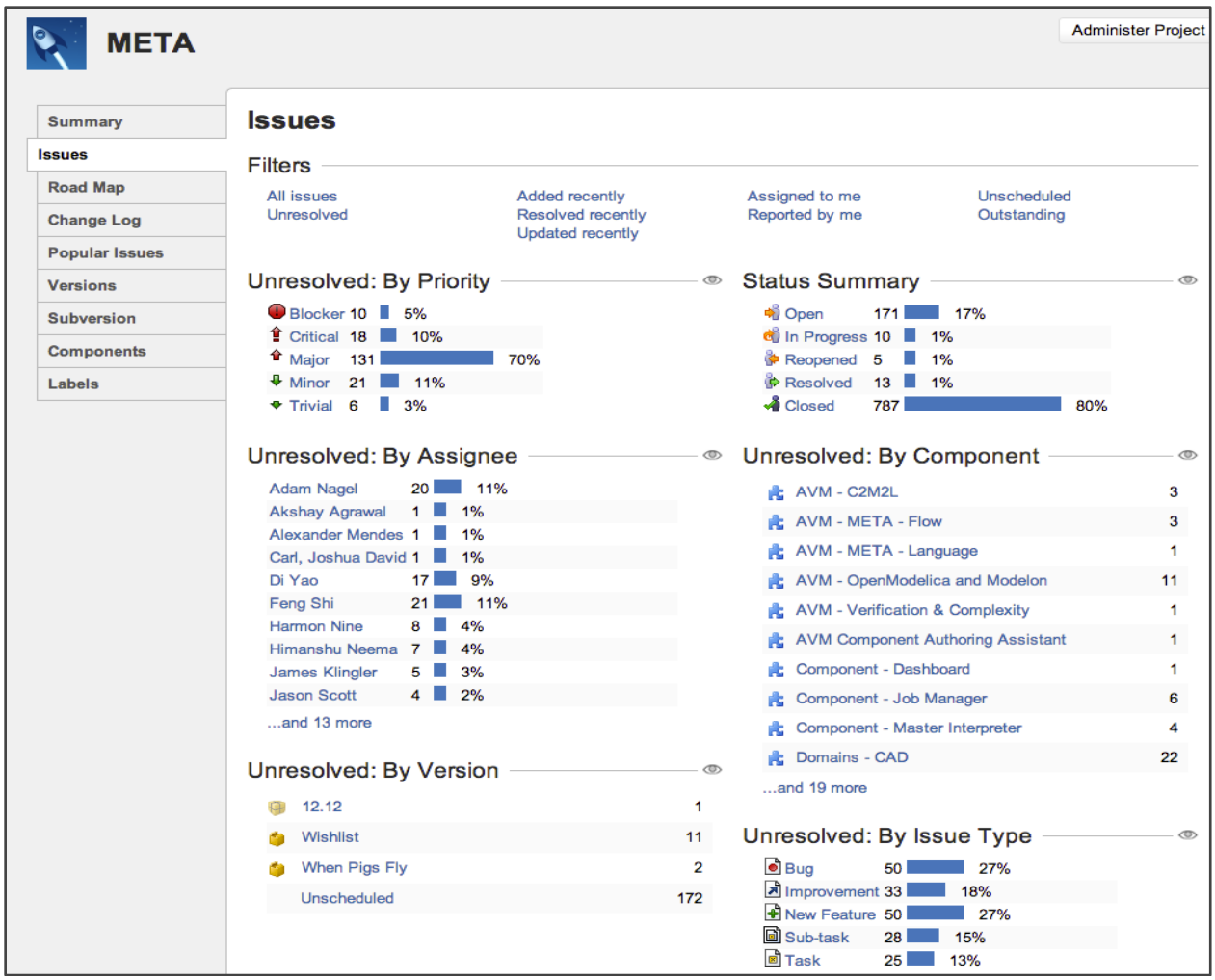


Figure 5: JIRA Issue Summary page for META

5.15 Change Control

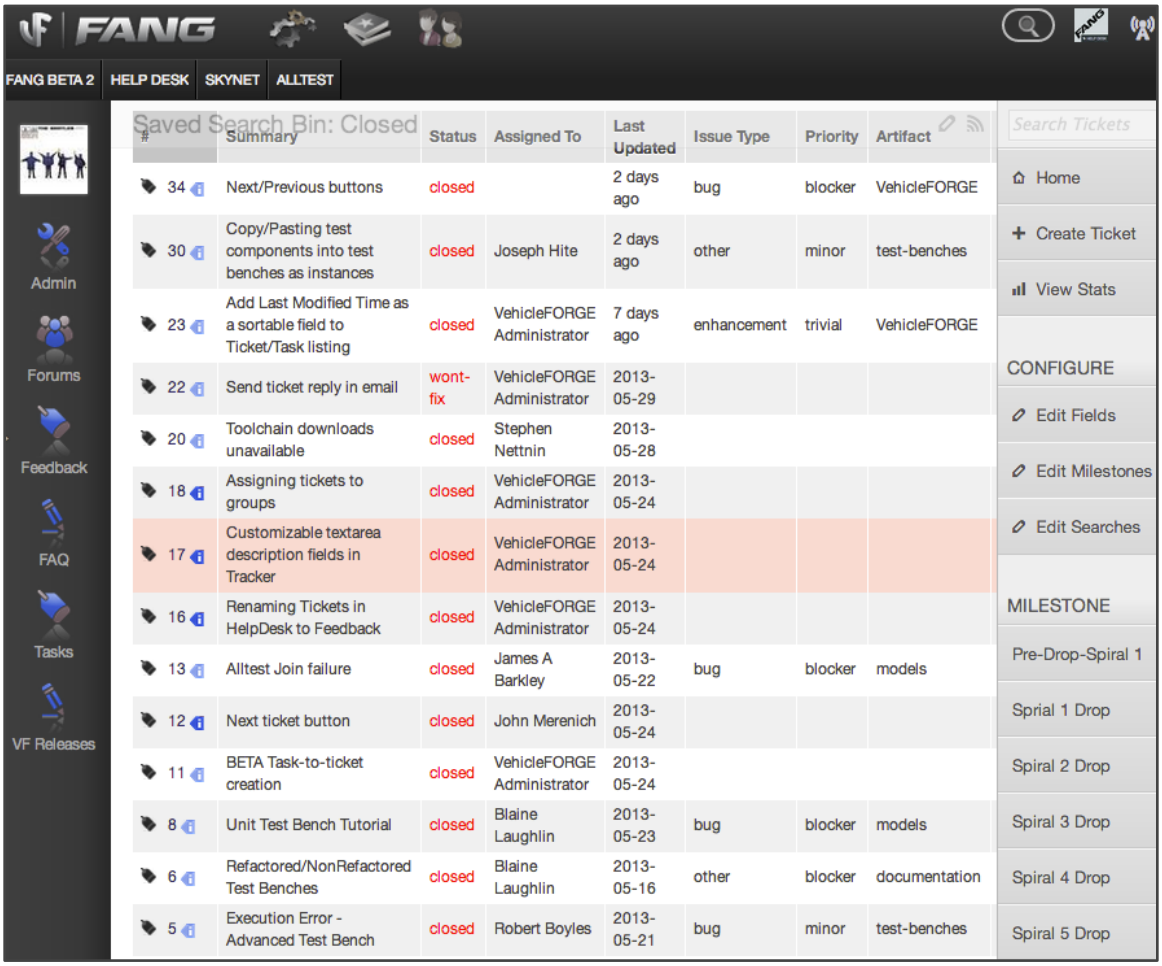
Change control for software will begin during the integration phase and must start when software identified with a numeric release is given to someone outside of software development for use in their work.

5.16 Problem Reporting

The Meta team tracks tasks and reports bugs internally using the previously mentioned JIRA system. Bugs found through informal reviews and unit testing are reported by the developers in JIRA directly.

Bugs identified through Beta testing are reported by the test community into the VehicleFORGE (VF) ticket tracking system, depicted in Figure 6. When an issue is verified, the ISIS support

team creates a JIRA task based on the VF ticket. This process eliminates "bugs" related to user error, inadequate documentation, etc.



#	Summary	Status	Assigned To	Last Updated	Issue Type	Priority	Artifact
34	Next/Previous buttons	closed		2 days ago	bug	blocker	VehicleFORGE
30	Copy/Pasting test components into test benches as instances	closed	Joseph Hite	2 days ago	other	minor	test-benches
23	Add Last Modified Time as a sortable field to Ticket/Task listing	closed	VehicleFORGE Administrator	7 days ago	enhancement	trivial	VehicleFORGE
22	Send ticket reply in email	wont-fix	VehicleFORGE Administrator	2013-05-29			
20	Toolchain downloads unavailable	closed	Stephen Nettnin	2013-05-28			
18	Assigning tickets to groups	closed	VehicleFORGE Administrator	2013-05-24			
17	Customizable textarea description fields in Tracker	closed	VehicleFORGE Administrator	2013-05-24			
16	Renaming Tickets in HelpDesk to Feedback	closed	VehicleFORGE Administrator	2013-05-24			
13	Alltest Join failure	closed	James A Barkley	2013-05-22	bug	blocker	models
12	Next ticket button	closed	John Merenich	2013-05-24			
11	BETA Task-to-ticket creation	closed	VehicleFORGE Administrator	2013-05-24			
8	Unit Test Bench Tutorial	closed	Blaine Laughlin	2013-05-23	bug	blocker	models
6	Refactored/NonRefactored Test Benches	closed	Blaine Laughlin	2013-05-16	other	blocker	documentation
5	Execution Error - Advanced Test Bench	closed	Robert Boyles	2013-05-21	bug	minor	test-benches

Figure 6: Beta Testing Issue Reporting (Beta.VehicleFORGE.org)

5.17 Continuous Build

The ISIS Meta project utilizes an automated build and testing system to track our tool development. The build and test system, Jenkins[1], is an open source continuous integration tool. Jenkins is written in Java and is a web-based platform. We have deployed a NUnit plugin to enhance our testing capabilities. This process maximizes the level of working software during our development process. Each build is triggered by a developer's repository check-in.

[1] http://jenkins-ci.org/

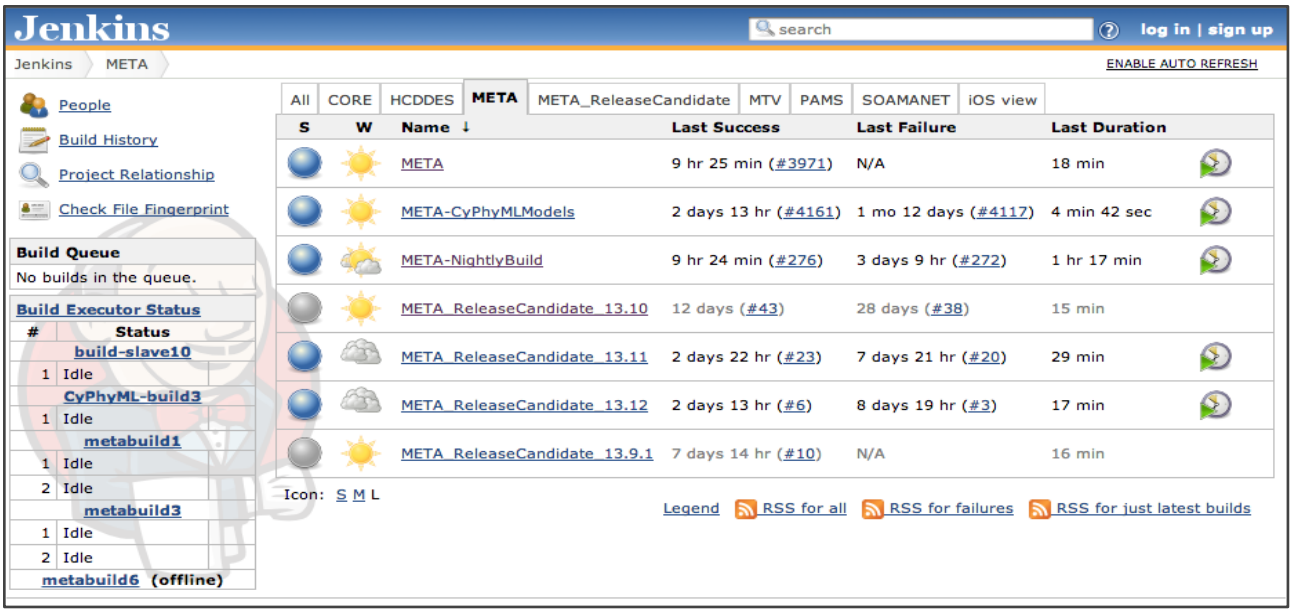


Figure 7: Jenkins interface showing the status of the Meta development branch

The Jenkins system regularly, after every source code change in our version control system, compiles the Meta tools as they are being developed. Builds that do not compile can be diagnosed, with fixes merged into the appropriate build. Figure 7 presents a Jenkins Status Page.

The following lists outlines the automate test battery all newly committed code will be tested against. Note that there are several tests within each category; the number is listed in the far right column of Table 2 below.

Builds that do not pass the test battery are not possible to merge with the main development branch.

All Tests

Total Tests	Pass	Skip	Fail	Duration	Package
17	17	0	0	41 sec	CADTeamTest
25	25	0	0	3.7 sec	ComponentAndArchitectureTeamTest
10	10	0	0	29 sec	ComponentExporterUnitTests
12	12	0	0	1 min 4 sec	ComponentImporterUnitTests
47	47	0	0	1 min 56 sec	ComponentInterchangeTest
20	20	0	0	5.6 sec	ComponentLibraryManagerTest
14	14	0	0	52 sec	CyPhyPropagateTest
1	1	0	0	0.16 sec	CyberTeamTest
3	3	0	0	1.2 sec	CyberTeamTest.Projects
24	24	0	0	49 sec	DesignExporterUnitTests
9	9	0	0	4.9 sec	DesignImporterTests
5	5	0	0	2.3 sec	DesignSpaceTest
1	1	0	0	0.99 sec	DynamicsTeamTest
266	266	0	0	2 min 17 sec	DynamicsTeamTest.Projects
59	59	0	0	38 sec	ElaboratorTest
214	214	0	0	1 min 57 sec	MasterInterpreterTest.Projects
6	6	0	0	3.1 sec	MasterInterpreterTest.UnitTests
1	1	0	0	1.1 sec	ModelTest
1	1	0	0	1 sec	PythonTest

Table 2: Automated Tests

5.18 Services and Resources Provided to AVM

ISIS will configure and maintain a remote execution service to provide execution of Meta Test Benches for the Gamma Test period and pre-test period testing.

ISIS will, under this contract, manage and maintain the following items:

Installation and configuration Meta Job Execution Server (*Jenkins*)
Installation and configuration of Meta CyPhy Software installers and dependencies
Installation and configuration of 3rd Party Software necessary for Gamma Test

Remote Execution.
Note the physical computational resources will be provided by ISIS. Currently 160 VM's and 20 physical machines have been allocated as remote processing nodes for the Gamma period. The Meta remote processing nodes use the virtual machine environment provided by VehicleFORGE.
Installation and configuration Meta Job Server (Jenkins1)

The Meta Job Server will need to be updated when the CyPhy software is updated to reflect the proper CyPhy version numbers, etc. so that the remote jobs are routed to the correct job servers.

Installation and configuration of Meta CyPhy Software distributions

The Meta CyPhy software will be updated on the dates in the schedule below as well as any other times the Meta CyPhy software updates are released to the Gamma Test community.

- November 27, 2013 - Meta 13.18 Install
- December 10, 2013 - Meta 13.19 Install
- January 6, 2014 - Meta 14.01 Install
- January 13, 2014 - Meta 14.01 Final Install following jury period.
- January 27 - May 15, 2014 - Gamma updates as needed.
- ~ March 28, 2014 - Mid Gamma Content Upgrade.

Installation and configuration of 3rd Party Software necessary for Gamma Test

The third -party software necessary for the Gamma Test falls in two categories:

- Software from AVM performers
- Commercial Software.

AVM Performer Software:

- SwRI Blast Tools**
- SwRI Ballistics Tools**
- SwRI Corrosion Tools*
- Ricardo Python Based Tools*
- PSU/iFAB Detailed Analysis tool configuration (analysis is performed on PSU resources)
- iFAB Conceptual Analysis Tool*
- iFAB Structural Analysis Tool
- iFAB RAM-D Analysis Tool**

**Local Execution* ***Local and Remote*

COTS Software:

Creo CAD Software***
LS-DYNA (Livermore Software Technology Corporation)
CTH Impact Simulation Software (Sandia National Laboratories)
OpenFOAM for CFD computations
Abaqus for FEA computation
Dymola for Dynamics computation

***Local and Remote*

Remote Processing Node support plan
ISIS will be actively available to support Gamma participants during Central/Eastern time zone business hours.

During the Gamma period, ISIS will perform assessments of the remote processing infrastructure. If resources are judged to be insufficient, more resources will be allocated from VehicleFORGE. Long term solutions to inefficiencies, job failures and node allocation will be considered during these weekly assessments. In the event of job failures, the necessary remote personnel will prioritize the resolution above other tasks to insure the fix in a timely manner.

Assessments will take place weekly on Thursdays from 2--3pm Central.

5.19 Software Testing

5.19.1 Unit Test

All code will be unit tested to ensure that the individual unit (class) performs the required functions and outputs the proper results and data. Proper results are determined by using the design limits of the calling (client) function as specified in the design specification defining the called (server) function. Unit testing is typically white box testing and may require the use of software stubs and symbolic debuggers. This testing helps ensure proper operation of a module because tests are generated with knowledge of the internal workings of the module.

5.19.2 Integration Test

There are two levels of integration testing. One level is the process of testing a software capability. During this level, each module is treated as a black box, while conflicts between functions or classes and between software and appropriate hardware are resolved. Integration testing requirements are shown in Attachment 2. Test cases must provide unexpected parameter values when design documentation does not explicitly specify calling requirements for client functions. A second level of integration testing occurs when sufficient modules have been integrated to demonstrate a scenario or user thread.

5.19.3 Alpha Testing

Once a developer has completed a new feature, improvement, major bug fix or development task, a package or zip of installers and documentation are wrapped. The documentation includes a background description, user tool instructions, and any other notes. These are then sent to multiple internal ISIS alpha testers for testing.

Alpha testers will first address the specific software update that has been implemented into the tools using the latest installer. If the alpha tester is able to successfully complete the task, they will notify the tester and then proceed to testing other core Meta GUI and Test Bench features. A standard checklist is used to sign-off on the core tools by the alpha tester. This process ensures that the new code implementation did not negatively affect other core Meta features. An example of this checklist is seen in Figure 8.

CL01	META_x64.msi	Title	*META Installer testing*			Accepted		Name	Patrik Meijer
Version	14.04	Version	Software			Changes Required		Date	July 18, 2014
Revision	r25423	7 (64-bit)	Windows			Retest Required			
Build	#18	13.11.14.1878	GME			Full Retest Required			
Link	http://build.isis.vanderbilt.edu/job/META_ReleaseCandidate_14.04/18/	2010	Visual Studio						
		2014 (64-bit)	Dymola						
		1.9.0 (r17414)	OpenModelica						
		2.7.6 (32-bit)	Python						

Id	Test action	Pass		Comment	Id	Test action	Pass		Comment
1	Automatic Tests				4	Dynamics w/ OM			MasterInterpreter, CyPhyPET, CyPhy2Mocelica_v2, run_PCC using OpenModelica, simulate_om
1.1	DynamicsTeamTest				4.1	MasterInterpreter on Dynamic TestBench	Local		
	CyPhy2Modelica_v2-, CyPhyPET- and CimLight-Interpreter						Remote		
1.2	run_selected_set_of_tests.py				4.2	PCC	Local		
	simulate_om/dymola, run_PCC, run_parameter_study, run_optimizer						Remote		
1.3	MasterInterpreterTest				5	Model Management			ModelicaImporter on MSL and external Libary, simulate_dymola/om, CLM_light, VF, ComponentImporter
	MasterInterpreter calling CyPhy2Modelica_v2, CyPhyPET				5.1	Import MSL components			
2	Design Space			DESERT (w/ constraints), DESERT Configuration to Component Assembly					
	DriveLine generate configurations				5.2	Import components from non-MSL			
3	Dynamics w/ Dymola			MasterInterpreter, JobManager, CyPhy2Modelica_v2, SoT, simulate_dymola, dashboard, run_PCC, run_parameter_study					
3.1	MasterInterpreter on Dynamic TestBench	Local			5.3	CLM Light Local Wiring			
		Remote							
3.2	SoT	Local			5.4	DriveLine download new Engines			
		Remote							
3.3	PCC	Local			6	Parameter Editor			Component, Test Component, Component Assembly, Design Container, TestBench, SoT
		Remote				Edit parameters in valid contexts			
3.4	sResponce Surface Experiment	Local			7	Dashboard			Dashboard all aspects on all models (done only when new Dashboard arrived)
		Remote				overwrite_dashboards.py			

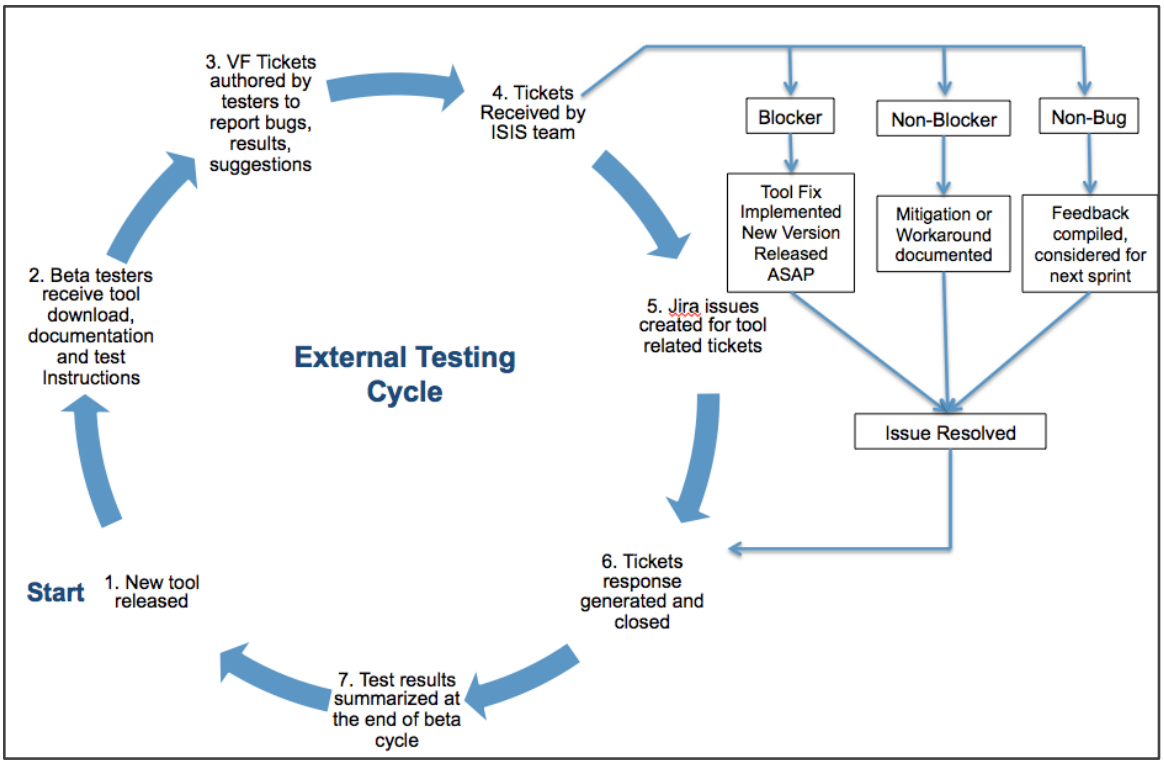
Figure 8: Installer Acceptance Test Checklist

If an alpha tester is unable to successfully test the new feature, the tester records the issues in a JIRA ticket and assigns it to the developer's original ticket. The variety of issues seen can include: software errors, documentation or instruction errors, or a new error with the core Meta tools that was not occurring in a previous version.

Once the developer fixes the issue, the alpha tester installs the new version of the tools and re-tests the feature and core tools using the same or updated documentation. A Meta version is not released until two versions of the alpha-sign form is filled out below, stating successful results.

5.19.4 Beta Testing

Once there is alpha sign-off on the new tool or feature, as well as sign off on the corresponding Meta tool version, all of the items required are prepped to be sent out for beta testing. The beta testing process is seen in the diagram below:



The beta testing cycle begins with the tool installer and documentation being released and posted to the beta community on the VehicleFORGE resource page. Once everything is posted, the testers sent an email with a brief summary of what has been released. Also, URL link locations are posted within the email for each of the following:

- **Tool installers:** Tool installers such as and updated Open Meta-CyPhy version, HuDAT, SwRI tool, etc.

- **Meta-CyPhy Release Notes:** These list the latest new features, improvements, bug fixes, or tasks that were implemented in the Open Meta-CyPhy version. Updated subversion release notes, will be the same as the version they originated from, but will have highlighted items indicated what is new in the subversion. For example, 14.03.2 release notes will be the same as 14.03.1, except for the new items listed which are highlighted on the notes.

- **New tool or feature overview and user instructions:** This documentation will contain all pertinent information necessary to for testers to understand and use the tool. The sections in this document are the purpose, procedures, installation notes, tool background,

requirements tested (Test Bench document), theory of operation, instructions for use, metrics, troubleshooting, and future enhancements.

- **Specific testing instructions:** These instructions are written in a "task" through VehicleFORGE's ticket system. Tasks usually include brief background context and instructions for testing. Tasks will sometimes be assigned to testers depending on what needs to be tested.

Beta testers will begin the testing process by downloading and installing the tools, reading through the tool documentation, and using the instructions on the VehicleFORGE task for specific testing instructions. Testers will submit feedback tickets for both specific issues with the task-at-hand, as well as general suggestions. There are several different types

5.19.5 Gamma Test

Gamma testing follows a similar flow to Beta, except the users are much less familiar with the META approach. There is more effort expended in ensuring minimal defects are released and a broader community is involved in deciding both what should be fixed and what should be released [and when].

There are checklists for release readiness, release agreement, and certifications that both tools and Test Benches function properly. The two forms shown in Figure 9 and Figure 10 were used in the Gamma Release process.

Functions Properly	Tool
	Component Importer
	Meta-Link
	Component Authoring Tool
	CLM Light
	DESERT
	Master Interpreter
	PCC
	Project Analyzer

Figure 9: Core CyPhy Tools Test Checklist

Scoring	Local	Remote	Test Bench
			Blast
			Ballistics
			Conceptual MFG
			Detailed MFG
			Completeness
			Ergonomics
			Ingress/Egress
			FOV
			Field of Fire
			Transportability
			Counting
			Dynamics (Surrogate)

Figure 10: Core Test Bench Functionality Checklist

5.20 Meta Release Schedule

Meta releases are available for every code change; however public releases undergo a more involved release process. External public releases follow our 4-week internal development sprints. Below in Figure 11 is the release schedule for 2014

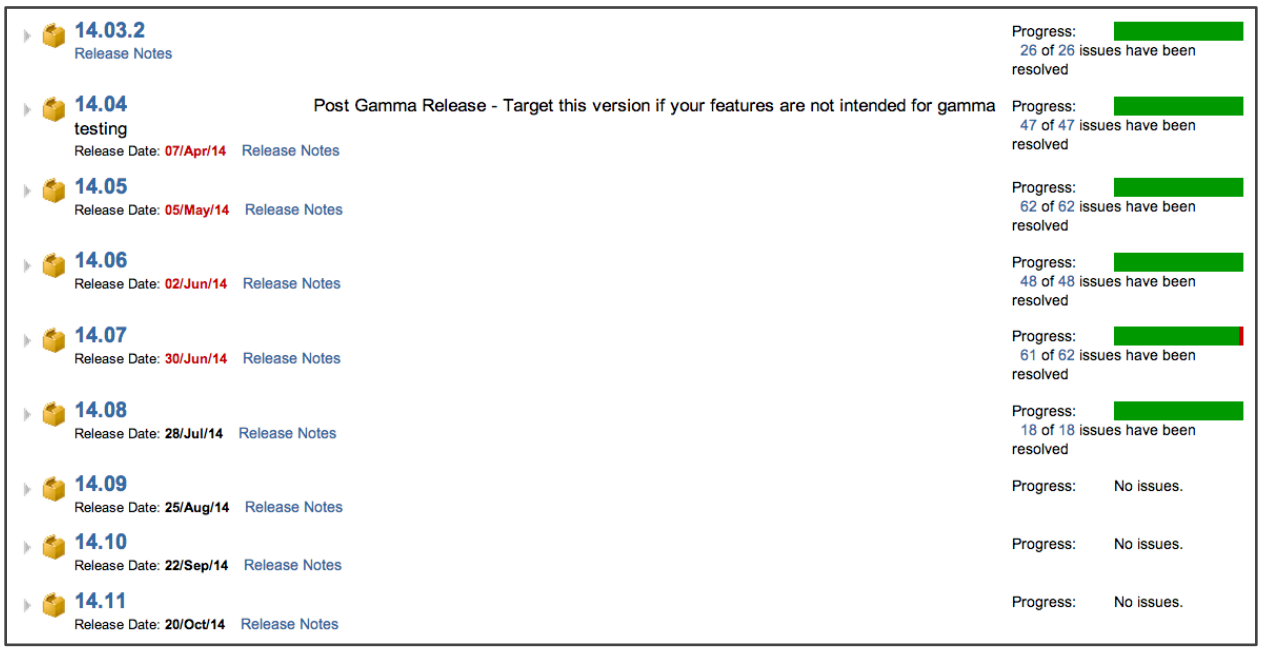


Figure 11: 2014 META Release Schedule

5.21 Quality Metrics

Most metrics have been mentioned before, but the following are important to characterize maturity, complexity, maintainability and quality of the final software products:

- Total Defects
- Top 10 modules for defects
- Defect Sources
- Failure Interval
- Closure rate
- Time to closure
- Maximum time to closure
- Burndown rate
- Number of defects with closure rates longer than 4 spirals
- Defect Insertion rates

- Time to detect inserted defects
- Complexity Number per module
- Number of modules with Complexity greater than 10.
- Rework Size and efforts

This section lists outlines the metrics collected to assess our QA efforts. While the number cannot capture the entire process, they are useful points of information. We have prioritized metrics that are lightweight to collect given our tight development schedules.

Continuous Builds – Every time a developer commits code to our build server, an automated build and test is triggered. We collect data on the numbers of successful builds across the team and on an individual developer level.

Bugs & Bugs fixed – Our JIRA system tracks and details bugs and other issues. We are able to assess the numbers of bugs reported, the time required to fix it, and successfully fixed bugs.

User feedback – We have a number of methods of collecting user feedback. The primary method is VehicleFORGE tickets emerging from beta testers, FANG competitors, and other users. These tickets can be assessed in terms of quantity, turnaround time, and by issue category.

Successful user threads – The AVM effort is organized into user threads that make up all actions a FANG user would conduct. This organization method allows us to assess our progress with the overall program and understand where we are being successful.

Execution of Development Plan – Our development is organized into four-week sprints with goals outlines and tracked throughout the phase. Following a sprint, we analyze the previous sprint's progress.

6 Appendix

6.1 Coding Documentation Requirements

- A high level language shall be used except when approved by SPM.
- Each method, function and class will be identified with their own comment header. The contents of the header should identify the purpose and any assumptions the user or caller must be aware of.
- Coding documentation will, at a minimum, describe reasons for code branching and a description of each variable name at their point of memory allocation.
- Naming conventions shall be used that clearly distinguish literal constants, variables, methods and class/object names. Class/object names should be nouns, methods should be verbs. Variables shall not be re-used for different purposes, except in trivial cases such as loop counts and indices. In addition, all names will contain at least 2 (two) characters to facilitate global pattern searches.
- Coding complexity conventions for a Class shall be established, such as the use of the Cyclomatic Complexity Matrix. A description of how to calculate Cyclomatic complexity index can be found in Chap 13 of Software Engineering a Practitioners Approach by Roger S. Pressman.; McGraw-Hill. The design will not exceed a complexity index value (Vg) of 10, without the approval of the SPM.
- Dispatcher logic shall include a default clause, and loops shall include an escape clause except in forever loops.

6.2 Testing Requirements

a. Unit Testing:
Environment; Specify testing environment. i.e. if and when stubs and drivers and/or other application routines, special hardware and/or conditions are to be used.
Logic Complexity: Calculate the Cyclomatic complexity matrix index which specifies the number of test cases required to ensure that all code is executed at least once. A description of how to calculate Cyclomatic complexity index can be found in Chap 13 of *Software Engineering a Practitioners Approach by Roger S. Pressman*, McGraw-Hill.
Boundary Analysis: Specify tests that will execute code using boundaries at *n-1, n, n+1*. This includes looping instructions, while, for and tests that use LT, GT, LE, GE operators.
Error handling: Design tests that verify the recording of all detected and reportable errors that a program is designed to find and report.
Global parameter modification: When a program modifies global variables, design tests that verify the modification. That is; initialize the variable independent of the program, verify memory contents, run the program, check that memory contents have been modified.
Mathematical limit checking: Design tests that use out of range values that could cause the mathematical function to calculate erroneous results.

Cessation of test: Specify the conditions under which a testing session stops and a new build is made. Regression testing is required, according to steps 2 through 6 above, of all lines of code that have been modified.
Documentation: The documentation must show that the tests have shown that the topics in items 2 through 6 above have been addressed.

b. Integration Testing;
This type of testing addresses the issues associated with the dual problems of verification and program construction. Integration is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested modules and build a program structure that has been dictated by design. The following topics are addressed in the STP.
Critical module definition: Decide which classes/modules contain critical control operations. These class/modules should be unit tested as soon as possible and not wait for subordinate class/object completion. Use of program stubs may be necessary.
Object grouping: Decide what modules comprise an integration group by use of scenarios and appropriate architecture diagrams. It is desirable to integrate at low levels to make bug definition easier. Choose objects that are related to a specific function like command uplink.
Depth vs. breadth testing: Decide how to test a group of objects/classes. It is suggested that breadth testing be used when interfacing with the hardware. Use stubs, if required, to test dispatcher control modules. Use depth testing when a function is well defined and can be demonstrated, e.g. and application mode like timed exposure.
Regression testing: Integration regression testing is required whenever an interface attribute has been changed, e.g. the value of a passed parameter.
Top down vs. bottom up: Use top down testing to verify major control or decision points. Use bottom up to test hardware driver type programs.

c. System testing:
System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Each test may have a different purpose, but all work to expose system limitations. System testing will follow formal test procedures based on hardware, software and science requirements as specified in the STP.

d. Validation Testing
The purpose of validation is to prove that the META software performs as specified in the User Threads.
Validation tests/procedures will identify a testing method and pass/fail criteria.
When ranges are specified in the requirements, tests cases will include boundary values at *n-1, n, n+1* where possible.
When LT or GT limits are specified, the measured value should be recorded.

e. Testing Documentation:
Testing documentation must be sufficient to provide evidence that the testing objectives as stated in the preceding sections or the STP have been met.

6.3 Inspection and Code Review Guidance

Time required: About 200 LoC can be scheduled for review in an hour. Reviews should be no more than 90 minutes long.
Location: Free of distraction, environmentally comfortable, support for several laptops and a projector [with screen or appropriate flat surface].
Planning for each review session should include access to user threads, code, test cases, Test Benches, and results of the Test Benches. All previous Bugs and JIRA tickets for that module should also be available.
Principal engineer should provide informal notes on the changes and their expected impacts, defects, design choices, etc.
Reviewers should include a chair with overall view of the META program, a domain specialist, a language lawyer, and one other technical contributor in addition to the principal engineer.
The Chair and Principle engineer document the actions, changes, defects, downstream issues, and expected results within 24 hours of the review session.
Within 2 weeks, a subset of the review teams to verify the closure of action items, defect removal and analysis, and Test Bench results.
Attachment 4: Defect Types
As a minimum, the following defect types should be covered:

- Documentation
- Syntax
- Build, package
- Assignment
- Interface
- Checking
- Data
- Function
- System
- Environment

6.4 Sample Checklist

- Language
- All functions are complete
- “Includes” are complete
- Check variable and parameter initialization
- At program initialization
- At start of every loop
- At entries
- Calls [Pointers, Parameters, use of &]
- Names [Consistent, within declared scope, use of “.” for structure/class refs.]
- Strings [Identified by pointers, terminated in NULL.]
- Pointers [Initialized NULL, only deleted after new, always deleted after use if new.]
- Output format [line stepping proper, spacing proper]
- Ensure {} are proper and matched
- Logic Operators [Proper use, proper ().]
- Line Checks [Syntax, Punctuation]
- Standards compliance?
- File Open and Close [properly declared, opened, closed]
- Meaningful error messages
- Consistent style
- Clean style
- Computation considerations
- Unused Code
- Security Issues
- Adequacy of Comments