GM600 Software Development Plan

A drawing of a machine with various gauges and pressure gauges

Description automatically generated

# Overview

This document captures the software development decisions, activities, and logistics for developing all of the software that executes on the GM6000 Digital Heater Controller’s Control Board. This includes all the software that is needed to formally test, validate, manufacture, and release a GM6000.

# Glossary

| **Term** | **Description** |
| --- | --- |
| **Candidate Release** | A candidate release is a formal build that has a human friendly version identifier assigned to it. It is submitted for final verification before being declared a Gold Release |
| **CCB** | The Change Control Board is responsible for assigning priorities to software tasks and bugs as well as determining which bugs must be resolved in order to ship the software. The CCB is composed of the principal stakeholders for the project, and, by definition, they are empowered to make final and binding decisions with respect to the software’s quality and content. |
| **CI** | Continuous Integration is the practice of validating code (by compiling it and running automated tests against it) that has been checked in before merging the code to a stable repository branch. |
| **Confluence** | A web-based corporate wiki tool from Atlassian. |
| **DHC** | Digital Heater Controller. |
| **Doxygen** | Doxygen is the de facto standard tool for generating documentation from annotated C/C++ sources. <https://www.doxygen.nl/> |
| **Formal build** | A formal build is a software image that was:   * Built from a stable branch of source code that was tagged and labeled in the Git repositories * Built by the build server * Labeled with a canonical version identifier |
| **GitHub** | An SCM Git server is where the source code repositories are hosted as private repositories. |
| **Gold Release** | A gold release is a candidate release that has passed all of the verification testing and is publicly released to customers. |
| **Jenkins** | An open-source automation server that helps automate the building, testing, and deployment of software. It also facilitates continuous integration and continuous delivery. |
| **JIRA** | An issue tracking tool from Atlassian. |
| **MCU** | Microcontroller. |
| **MRS** | Marketing Requirement Specification. |
| **PLM** | Product Lifecycle Management manages all of the information and processes at every step of the product lifecycle across globalized supply chains. Windchill and SAP are examples of PLM tools. |
| **PRS** | Production Requirement Specification. |
| **Pull Request** | A pull request (PR) informs others about changes that developers have pushed to a branch in a GIT repository. |
| **QMS** | Quality Management System. |
| **SCM** | Software Configuration Management is the process and tools used to store, track, and control changes to the source code. For example, git is an SCM tool. |
| **Software BOM (SBOM)** | Software Bill-Of-Materials is a list of all third-party packages and their version information that were included in the released software. The software BOM can contain existing internal packages that are being used with the released software. |
| **SRS** | Software Requirements Specification. |
| **SWA** | Software Architecture document. |
| **SWD** | Software Detailed Design document. |
| **Ticket** | A ticket represents a unit of work with respect to the source code that is atomically merged to a stable branch in the SCM repository. Tickets are required to be formally identified and tracked by JIRA. |
| **Validation** | Validation is the process of checking whether the specification captures the customer's requirements, i.e., “are we building the right thing.” Examples of validation might be collecting Voice-Of-the-Customer inputs, running focus groups with users using mock-ups or prototypes, etc. |
| **Verification** | Verification is the process of checking that the software fulfils requirements, e.g., functional software testing. |

# Document References

| **Document #** | **Document name** | **Version** |
| --- | --- | --- |
| QMS-001 | Quality Manual | 1.16 |

# Roles and Responsibilities

The following roles need to be defined for the project. Which engineers are assigned to these roles is captured in the top level project plan.

| **Role** | **Responsibility** |
| --- | --- |
| **Software Lead** | The Software Lead is the technical lead for all software contained within the GM6000 Control board. This role is responsible for:   * Software architecture * Software detailed design * SRS requirements * Resolving (software) technical issues * Ensuring the software-specific processes are followed (especially reviews) * Signing off on the final releases * All the responsibilities of a software developer |
| **Software developer** | The Software Developer writes and tests code. This role is responsible for:   * Assisting with software architecture * Assisting with software detailed design * Assisting with SRS requirements * Implementing code and unit tests * Participating in design and code reviews * Following the defined SDLC processes |
| **Software Test Lead** | The Software Test Lead is responsible for all things related to software verification:   * Creating the formal test plan and test matrix * Creating test reports * Resolving (software testing) technical issues * Ensuring that the software test-specific processes are followed * Signing off on the final releases * All the responsibilities of a Software Tester |
| **System Engineer** | The System Engineer is responsible for the system level design, algorithms, etc. across all disciplines. This role is responsible for:   * Creating the PRS requirements and system architecture * Resolving system technical issues * As needed, participating in design and code reviews * Signing off on the software test plan * Signing off on the final releases |
| **Software Tester** | The Software Tester formally tests the software. This role is responsible for:   * Assisting with the test plan and test matrix * Assisting with test reports * Developing and executing test cases * Authoring bug tickets * Following defined software development life cycle (SDLC) processes |
| **Hardware Lead** | The Hardware Lead is the technical lead for all things hardware. This role is responsible for:   * Creating the electrical hardware architecture * HWR requirements * Resolving EE hardware technical issues * Ensuring that hardware development processes are followed * Signing off on the final releases * As needed, participating in design and code reviews |

# Software Items

The software items covered under this development plan are:

1. Software that executes on the GM6000 control board when it is shipped to a customer
   1. This software item requires formal testing and verification before being released.
   2. The software shall be programmed in C/C++ and conform to *SW-1002 Software C/C++ Embedded Coding Standard*.
2. Manufacturing test software (that executes on the GM6000 control board) that is used when manufacturing the GM6000
   1. This software item will be informally verified by engineering before being released to manufacturing.
   2. The software shall be programmed in C/C++ and conform to *SW-1002 Software C/C++ Embedded Coding Standard*.
3. Engineering test software used for external testing (e.g., UL, FCC, etc.) and internal formal testing (e.g., HALT, etc.)
   1. This software item will be informally verified by engineering before being used in external or internal testing.
   2. The software shall be programmed in C/C++ and conform to *SW-1002 Software C/C++ Embedded Coding Standard*.
   3. This software is not for public release.

All of the above software items—even those not formally verified—are required to go through the formal process with respect to software builds.

# Documentation Outputs

1. The supporting documentation shall be created in accordance with the processes defined in *QMS-010 Software Development Life Cycle Process* document.
2. A Software Architecture document shall be created and assigned a formal document number. The Software Lead is responsible for this document.
3. A Software Detailed Design document shall be created and assigned a formal document number. The Software Lead is responsible for this document.
4. The Doxygen tool shall be used to document the code level details. A formal document number will be assigned to the Doxygen output and must be included on the home page of the Doxygen output. The Doxygen output shall be created as part of the CI build process.
5. Code reviews are performed as part of the Pull Request process. The review comments and actions are available by accessing the GitHub repositories for the project. Later, the review comments and artifacts can be retrieved using GitHub REST APIs for consolidation into a stand-alone document.
6. Design reviews are done iteratively as part of the ticket workflow process.
   1. Review comments are captured as “comments” in the documents under review section. After the updates or actions have been satisfactorily completed, the comments are marked as resolved. (But the comments are not deleted.)
   2. The review comments can only be deleted after a snapshot of the document has been archived.
7. The following documentation artifacts are captured in Confluence as wiki pages. The Software Lead is responsible for these items:
   1. Instructions on how to set up a developer’s local build environment
   2. Instructions on how to manage the tools on the build server
   3. Instructions on how to setup the CI platform
   4. The list of all development tools (and their version information) that were used for development
8. Internal release notes shall be created for each formal build that is provided to the software test team for verification testing. These notes can be captured in an email or on wiki pages. The Software Lead is responsible for these artifacts.
9. For candidate releases, a single release notes document shall be created and assigned a formal document number. The Software Lead is responsible for this document.
10. A Software Bill-Of-Material (SBOM) document shall be created and assigned a formal document number. The SBOM shall be updated every time there is a new gold release. The Software Lead is responsible for this document.
    1. The SBOM shall identify and reference open source and proprietary licensing requirements and agreements.
11. A release cover page shall be created when releasing the software into the PLM system. The Software Lead is responsible for supplying the content for this document. The specifics of the cover page are dedicated by the PLM system.

# Requirements

1. The supporting documentation shall be created in accordance with the processes defined in QMS-004 Requirements Management document.
2. The MRS is a formal document with an assigned number that captures all of the top-level user and business needs. The Product Manager is responsible for this document.
3. The PRS is a formal document with an assigned number that captures the system level requirements that are derived from the MRS. The System Engineer is responsible for this document.
4. The HRS is a formal document with an assigned number that captures the hardware level requirements that are derived from the MRS and PRS. The Hardware Lead is responsible for this document.
5. The SRS is a formal document with an assigned number that captures the software level requirements that are derived from the MRS and PRS. The Software Lead is responsible for this document.
6. Traceability. The project has the following traceability requirements:
   1. All requirements will have a globally-unique requirement number assigned to them.
   2. The SRS requirements shall be traceable to at least one parent requirement in the MRS or PRS.
   3. All SRS requirements shall be traceable to at least one or more Software Architecture output items.
   4. Software Architecture:
      1. All output items identified shall be assigned a unique identifier. An output item is a documentation section that contains design outputs.
      2. The output items identified shall be traceable to at least one MRS, PRS, or SRS requirement.
   5. Software Detailed Design:
      1. All output items identified shall be assigned a unique identifier. An output item is a documentation section that contains design outputs.
      2. All output items shall be traceable to an output item in the software architecture document.
      3. All output items shall specify the source code (by directory) that implements the design.
   6. A trace matrix document shall be created (with an assigned number) that enumerates this traceability process. The System Engineer is responsible for the trace matrix.

# Software Development Life Cycle Process (SDLC)

1. All project specific SDLC processes shall be developed in accordance with the processes defined in *QMS-010 Software Development Life Cycle Process* document.
2. There are four phases: planning, construction, verification, and release.
3. The Planning phase shall consist of planning, gathering requirements, creating the software architecture, and preparing the tools and infrastructure needed for the construction phase.
   1. This process will generally follow an iterative, agile kanban process with tasks captured in JIRA.
   2. All code checked into GitHub during this phase requires a ticket. The ticket workflow shall be the same as the workflow described under the construction phase.
   3. With respect to software development, the planning phase is considered waterfall in that the construction phase shall not begin until the planning phase has completed.
   4. The planning phase is exited after the following deliverables have been completed:
      1. The first draft of the *SWA-1327 GM6000 Software Architecture* document has been reviewed.
      2. The foundational skeleton application can be successfully built by the CI server (including skeleton automated unit tests).
4. The Construction phase shall consist of detailed design, implementation, testing, and bug fixing.
   1. This process will generally follow an iterative, agile kanban process with tasks captured in JIRA.
   2. All code checked into GitHub during this phase requires a ticket. The following steps are associated with each ticket during this phase. Some of the steps (except for Pull Request and Merge) can be waived with verbal approval from the Software Lead.
      1. Requirements—The requirements for the work to be performed are identified. Generally, the requirements are listed or captured in the JIRA card.
      2. Detailed design—The developer is responsible for documenting the detailed design associated with the work to be performed in the SDD.
      3. Design Review—This is a peer review of the detailed design.
      4. Coding and Unit testing—The code is written and verified using unit tests.
      5. Pull Request and Code Review—The completed and tested code is submitted for peer review by creating a Pull Request. The Pull Request cannot be merged until after the reviewer(s) approves the changes, there are no build errors, and all automated tests pass. It also assumes the merge is a trivial merge; that is, there are no merge conflicts that need to be resolved.
      6. Merge—The Pull Request is merge to a stable branch.
   3. The software team is responsible for additional system level and integration testing. The testing activities will be tracked as cards in JIRA. For each integration test cycle the following steps are performed:
      1. A test plan has been created. The test plan includes the goal or purpose of each test case, specifies how each test case will be executed, and what the pass/fail criteria is.
      2. The test plan has been executed.
      3. A summary report of the testing has been created.
   4. The Construction phase is only exited when the Release phase ends.
5. Verification shall consist of the formal verification of the software. The verification phase runs in parallel with the construction phase, i.e., the first pass through the test plan shall begin as soon as features and functionality are realized in code.
   1. The Software Test Lead is responsible for the formal verification of the software. *The SWQA-1401 Software Test Plan* documents how the testing will be conducted.
   2. The management of reported bugs shall be done in accordance with the processes defined in QMS-011 Software Defect Tracking Workflow document.
   3. All bugs found will be logged as bug cards in JIRA.
   4. The Verification phase is only exited when the Release phase ends.
6. The Release phase starts before the end of the verification phase, and it ends with the release of the software into the PLM system.
   1. During the release phase, the CCB is responsible for approving all code changes and fixes.
   2. The CCB is responsible for determining when the software can be released into the PLM system.
   3. The CCB is responsible for approving the scope of regression testing after merging bug fixes into the upcoming release branch.
7. The Release phase (as well as the Construction and Verification phases) can be restarted—as determined by the CCB—to address issues found in the software released to the PLM prior to physically shipping units.

# Cyber Security

1. The cyber security needs of the project shall follow the processes defined in *QMS-018 Cyber Security Work Instructions*.
2. The cyber security analysis and control measures shall be documented in the software architecture document. The Software Lead is responsible for the cyber security content.

# Tools

1. The software that executes on the DHC’s Control Board hardware shall be compiled with the GCC cross compiler for the specific microcontroller.
   1. The version of the compiler shall not be changed during the construction and release phases unless there is a documented compiler bug that impacts the software.
   2. The compiler toolchain shall be archived along with the source code in the GitHub.
   3. The compiler toolchain shall be tagged and labelled when a formal build is performed.
2. The GCC MinGW compiler shall be used for building automated unit tests. The GCC compiler is used because of its capability to instrument the test executables for the collection of code coverage data. Any compiler version 9.2 or higher can be used.
3. The gcovr python library shall be used to generate the raw code coverage metrics. Any version 5.2 or higher can be used.
4. The Catch2 test framework shall be used for automated unit tests.
   1. Version 2.x of the framework shall be used.
   2. The Catch2 framework shall be archived along with the source code in the GitHub.
   3. The framework shall be tagged and labelled when a formal build is performed.
5. Doxygen shall be used to generate code level documentation.
   1. The Doxygen output pages shall be converted into a Windows Help File (.chm)
   2. The Doxygen installer, the Graphviz installer, and the Microsoft Help Compiler shall be archived in a long-term storage location.
6. No static code analyzer will be used on this project.
7. Python shall be used for all internal script development.
   1. All developers, testers, and the build servers shall use the same version of python.
   2. The Python installer for the version of python being used shall be archived in a long-term storage location.
8. JIRA shall be used to track tickets and bug cards.
9. Confluence shall be used for project specific software wiki pages.
10. GitHub shall be used to host the GIT repositories.
11. Jenkins shall be used for automating Continuous Integration tasks.
    1. All formal builds are performed using Jenkins and its build agents.
    2. Jenkins will be installed on VM and the VM will be archived in long term storage for each Gold release.

# Software Configuration Management (SCM)

1. GitHub private repositories shall be used to version control all of the source code for the project.
   1. A single repository shall be used. The repository URL is https://github.com/xxxxx/gm6000.
   2. The repository will also contain all third-party packages and the cross-compiler tool chain used when building for the target hardware.
2. The branching strategy shall be a modified [trunk-based](https://www.atlassian.com/continuous-delivery/continuous-integration/trunk-based-development) development model.
   1. The main branch shall be used for all candidate releases.
   2. A child branch of main—develop—will be used as the stable branch for day-to-day development and pull-requests. The develop branch will be merged to main for candidate releases.
   3. Each ticket will be used to create a short-lived branch off of the develop branch. The ticket number shall be part of the branch name.
3. A project unique build number, generated by the Jenkins CI tool, shall provide the canonical build identifier for all formal releases.
   1. The build number shall be included as part of the SCM tag or label that is applied to each formal build. This is the canonical version identifier.
   2. The source code shall contain compile-time, human-readable version identifiers, that are set before the code is merged to the develop branch. The human-readable version identifier is not be used as the canonical identifier since there will be many formal builds with the same human-readable version identifier.

# Testing

1. The software team is responsible for unit testing and integration testing.
2. The source code is organized by namespaces (per *SW-1002 Coding Standard*). Each namespace is required to have at least one unit test for the code it contains.
   1. If the namespace has a direct target platform dependency, the unit test shall be a manual test that executes on the target platform.
   2. The BSP directories are exempt from the unit test requirements.
   3. If the namespace contains implementation for the UI, the unit test shall be a manual test that executes either on the target platform or the simulator. The automated unit test requirement is relaxed with respect to the UI because the test infrastructure does not include tools for automated verification of the UI’s visual presentation.
   4. All other namespaces shall have an automated unit that is a stand-alone application that returns pass/fail. Automate unit tests are executed as part of the CI process for all builds. All automated unit tests are required to meet the following code coverage metrics:
      1. Line coverage >= 80%
      2. Branch coverage >= 60%.

Because of the tools being used (gcc and gcovr) the branch coverage metrics are not always correct for C++ code. Consequently, the branch coverage threshold is intentionally lower to compensate.

1. The Software Lead is responsible for defining development milestones where integration testing shall be done.
   1. At least one integration test milestone shall be an automated sanity check (using the functional simulator) that will be executed as part of the CI build process.
2. Third-party packages will be verified as part of the overall software function during formal verification.
   1. Unit tests are not required for third-party packages.
3. It is permissible—under certain circumstances—for integration and formal software verification testing to use the software functional simulator instead of a physical hardware target.
   1. The Software Test and Software Leads must approve, on a per test case or per test suite basis, when the functional simulator can be used in integration and formal testing.

# Deliverables

This section provides a summarized list of deliverables that the software team is responsible for on the project. There are numerous other deliverables (e.g., the GM6000 Software Test Plan) that are not summarized here because the software team in not responsible for the deliverables.

| **Deliverable** | **Phase(s)** | **Notes** |
| --- | --- | --- |
| SDP-1328 GM6000 Software Development Plan | Planning  Construction | A reviewed first draft is required to exit the planning phase. |
| SRS-1324 GM6000 Software Requirement Specification | Planning  Construction | A reviewed first draft is required to exit the planning phase. |
| SWA-1327 GM6000 Software Architecture | Planning  Construction | A reviewed first draft is required to exit the planning phase. |
| SDD-1329 GM6000 Software Detailed Design | Planning  Construction |  |
| SDX-1330 GM6000 Software Doxygen Output | Planning  Construction |  |
| SW-1002 Software C/C++ Embedded Coding Standard | Planning |  |
| Code Review Artifacts | Construction | This is captured in GitHub as part of the Pull Requests. |
| Design Review Artifacts | Construction | These are point-in-time versions of the documents with review comments that have been archived. |
| Developer and Build Server Setup Instructions | Construction | Wiki pages. |
| CI Platform Setup and Maintenance Instructions | Construction | Wiki pages. |
| Software Development Tools List | Construction | Wiki pages. |
| SWT-1331 GM6000 Software Requirements Trace Matrix | Construction  Release | These are iterative drafts created during the Construction phase. |
| SWB-1332 GM6000 Software Bill of Material | Construction  Release | These are iterative drafts created during the Construction phase. |
| Engineer Test Software | Construction | This is a formal build of the Engineering test software images. |
| Manufacturing Test Software | Construction | This is a formal build of the Manufacturing test software images. |
| Integration Testing Artifacts | Construction | Per each test milestone, the test plan, procedure, and report documents will be archived. |
| Internal Release Notes | Construction | These are provided per each formal release that is deployed for internal testing. |
| SWR-1333 GM6000 Release Notes | Release | These are updated for each Candidate Release. |
| <plm-id> GM6000 Software Release Cover Page | Release | This is updated for each Gold Release, and it includes references or links to the released software images. |

# History

|  |  |  |  |
| --- | --- | --- | --- |
| **Author** | **Description** | **Date** | **Rev** |
| V. Dinkley | Initial Draft | 05/27/2023 | 0.1 |