The Firmware Architecting Process

Michael Milendorf
Sun Microsystems, Inc., Computer Systems Division
One Network Drive, Burlington, MA 01803, USA
Michael.Milendorf@sun.com

Firmware is the ROM-based software that controls a computer between the time it is turned on and the time the primary Operating Environment takes control of the machine. This paper describes the Firmware Architecting Process at Sun Microsystems Company. The process incorporates the sequential stages of firmware project design and review, firmware architecture review, code design and testing, code review, request to integrate, putback to the master sources, firmware built verification, firmware quality assurance, and release to customer. Sun Microsystems firmware, also known as OpenBoot ROM, is an implementation of IEEE 1275-1994 Standard for Boot Firmware and is done entirely in the ANSI Forth X3.215-1994.

OpenBoot is a *firmware operational environment* which is in the *boot ROM* of every computer system bearing Sun Microsystems logo. All *OpenBoot* binaries to support different Sun Microsystems platforms are built out of the single *master source code tree*. Forty engineers from three different engineering groups, located in California and Massachusetts work together to provide firmware support for all Sun systems from *Desktop* to *Workgroup* and *High-End Mainframe-class Servers*. The success of such enterprise depends on the use of the carefully designed *firmware architecting process* and on the standard *Open Architecture* for firmware, implemented in wonderful programming language *Forth*.

The whole *firmware architecting* or development process is broken in several sequential stages: *project design and review, firmware architecture review, code design and testing, code review, request to integrate, putback* to the master sources, *firmware built verification, firmware quality assurance* and finally *release to customer.*

Project design and review: During the early stages of the *firmware architecting process* ideas are brainstormed and *design proposal* documents are drafted. When the *design proposal* and the schedule for the project delivery are finalized, a *design review meeting* discusses the *design proposal*. At that time, new interfaces and other architectural changes are identified for submission to the *firmware architecture review counsel*.

Firmware architecture review: The *architecture review process* is a forum for engineers to get advise from Sun Microsystems technical leaders and a method to coordinate projects going on in different departments of the company. The *architecture review* focuses on the establishment, maintenance, communication and enforcement of an overall architecture and the creation, dissemination, and evolution of a compelling technical vision of Sun Microsystems products several years into the future. Another intention of the *architecture review* is to provide the project's team a consultation service with a group of senior engineers whose expertise is in the area of their project. This consultation has the benefit of exposing various liabilities that the project may have in relation to other projects or existing part of the system. The project team is given advise about what other projects engineers should be worked with, to reconcile any conflicting architectural issues. The *architecture review* also attempts to identify any duplication of effort, over-engineering, quality problems, or dangerous effects on the strategic direction of the system architecture.

To support the *architecture review* two organizational structures have been created within Sun Microsystems Company: The *System Architecture Council* (*SAC*) and *Architecture Review Committees* (*ARCs*). *Firmware Architecture Review Committee* (*FWARC*) is dealing with *firmware architecture* in the company. *SAC* and *FWARC* work together to organize and maintain the definition of the architecture, standards, and interfaces that Sun Microsystems firmware must confirm to.

Each firmware project goes through several stages for architectural approval. First, a *one-pager* is E-mailed to *SAC* with the short description of the project. When the project is being assigned to one of subcommittees, in our case *FWARC*, the project submitter sends additional materials to *FWARC* and request an *inception review* with the *FWARC* Chair. The *FWARC* meeting is scheduled and the project is presented to *FWARC*. Multiple meetings may occur ending with the *commitment review* during which *FWARC* members vote on the project's architecture. The project could be approved unconditionally, approved with changes required or rejected. The final *FWARC* opinion is binding on the project. The project is expected to carry out all required changes and consider all advised changes noted on the opinion.

FWARC is concerned specifically with all new interfaces between *OpenBoot* and *Operational Environment* (*Solaris OS*). Among interfaces considered are new *device tree nodes*, new *device tree properties*, new *user commands*, new interfaces between *FCode* drivers, *FCode* diagnostics and *OpenBoot*. *FWARC* is also verifying a new project compliance to *IEEE 1275-1994 Standard for Boot Firmware*. The committee tries to enforce the coherent consistency in firmware behavior visible to customers among all Sun Microsystems platforms.

Code Design: Based on *FWARC* recommendations the *project design proposal* document is improved and the code is being developed, written, compiled and tested. *OpenBoot* firmware code is written in *SPARC Forth Assembly*, *ANSI Forth* or *IEEE 1275-1994 FCode programming languages*. All firmware developers at Sun Microsystems are using common *OpenBoot* sources under *Unix*-based *Source Code Control System*. The system is designed to support teamwork code development and allows many engineers to share a single common *source code tree*.

Code Review: When the code is tested by the developer, a *code review* (inspection) meeting is called. Firmware group conducts *measured code reviews*. The following parameters are recorded and analyzed for each code review: the number of defects or issues identified in the source code, the number of people participating in the review, the staff hours spent, the meeting duration, the deliverable size, the lead time given to prepare, and the conclusion of the reviewers. With a moderator coordinating the review, the source code is walked through,

defects are identified, discussed and recorded. The code reviews generally last about an hour to two hours. Longer meetings found to be unproductive. If a lot of defects are found, the code re-review meeting may be needed to inspect all incorporated changes based on the first review. Code reviews are proven to be the most effective way to increase firmware quality and reliability. Measured code reviews significantly reduce the number of defects found in software after release.

Request to integrate: After the code is reviewed, the developer prepares his changes to integration into the common sources. We use *bug tracking database* and *request to integrate* (*RTI*) Unix-based *tools* at this stage of the *firmware architecting process*. Each bug fix, or enhancement to the existing code is recorded in the *bug tracking database*. The submitter assigns priority and severity of the bug (or *request for enhancement - RFE*), describes the problem and how it was fixed. The *bug tracking database tool* automatically assigns a bug (*RFE*) identification number and logs in the data. This identification number is later referred in the *RTI tool* which is used to coordinate the process of code integration into the common sources. At the time of *RTI* submission, a firmware developer selects an advocate for the *RTI* from the list of available advocates and a reviewer from the list of available reviewers. Both lists represent senior level firmware engineers. The developer includes a pointer to the source tree with the code to be integrated, any associated *FWARC* case numbers, and bug (*RFE*) identification numbers, and the *RTI* is submitted to the team.

The comment section of the *RTI tool* allows to type in comments which are recorded for this *RTI*. When an *RTI* is submitted, the *RTI tool* automatically assigns it a unique number. The advocate, reviewer and all firmware engineering team members receive an automatic E-mail announcing that the *RTI* has been submitted for the review. It is a responsibility of the reviewer and the advocate to review every *RTI* assigned to them in a week time frame. They can use comments section of the *RTI tool* to communicate any issues to the submitter, pertaining to the pending *RTI*. The submitter shall respond to any issues raised by the reviewer or the advocate by negotiating or fixing the code to the final satisfaction of both the reviewer and the advocate, at which time the *RTI* is *reviewed* by the reviewer and *approved* by the advocate.

Putback: When pending *RTI* is *approved*, the code submitter contacts the gatekeeper via E-mail, who opens the *master sources* for the putback. After the submitter executes the putback of his sources to the *master sources*, the gatekeeper closes and re synchronizes the *master sources*, and verifies new code by building *OpenBoot* binaries for all platforms, making sure the build process was not broken by the *putback*. The *gatekeeper* is also updates *bug tracking information* to the *integrated* state for all bugs (*RFE*) fixed by the *putback*.

Firmware quality assurance and release to customer: Every official *OpenBoot* binary build goes through semi-automatic *firmware quality assurance* process. The process verifies compliance of *OpenBoot* release to the *IEEE 1275-1994 Standard for Boot Firmware*, platform specific commands, the *device tree* and the *device tree properties*. If problems are found they are registered through *bug tracking database tool*. The responsible engineer goes back to the *code design* stage of the *firmware architecting process* to fix the problem.

The *Firmware Architecting Process* employed at Sun Microsystems allows to meet the demand of the corporation to develop and deliver robust, portable and reliable firmware solutions across all Sun platforms in timely fashion. *Forth Programming Language* fits in the process as an implementation tool. In this respect, *Forth* is not different than any other programming language. The Sun Microsystems *firmware architecting process* is an example of how complex software systems written in *Forth* can be designed and managed between many individual contributors and even teams.