Spring 2007

Creating a Software Assembly Line

Gary Allan Howard
Regis University

Regis University
School for Professional Studies Graduate Programs
Final Project/Thesis

Disclaimer

Creating a Software Assembly Line

CREATING A SOFTWARE ASSEMBLY LINE

Gary Allan Howard

Regis University

School for Professional Studies

Master of Science in Computer Information Technology

Abstract

This thesis describes a technical solution that improved the software development efforts needed to verify and validate a medical device, herein referred to as the "medical device." The medical device had many software and hardware configurations that had to be developed, integrated, managed, and tested. There were a number of problems with the manual processes that were being used to verify and validate the product, so this project developed a system called the "Software Assembly Line" to continuously build software and automatically test it on multiple hardware configurations. As a result, software quality and predictability were improved, and the number of cycles required for formal verification and validation was reduced. The final project recommendation was to validate the Software Assembly Line according to 21CFR820.75, Process Validation.

Acknowledgements

The work would not have been possible without the support from my employer. I am grateful to my management team, in order: David Shanes, Greg Gulden, Neil Holland, and Dr. Ed Ogunro.

Michael Singleton, Wei Soon, Charlie Gragg, Dr. Richard Hursey, Neil Holland, Greg Gulden, and David Shanes provided critical reviews of this work that ultimately made it better.

Lynn Neuhardt helped define the requirements for each deliverable and prototype, and managed the Software Automation Team. John Arrizza developed the Continuous Build and Email Notification Systems.

Sue Karlin provided excellent guidance and advice throughout the project execution and the writing process. Joseph Gerber facilitated the writing process blended with humor and critical analysis and insights.

Cj has unconditionally loved me since the day I asked her marry me on July 18th, 1983, and then blessed me by becoming my wife on December 10th, 1983. She believed in me when I did not, and saw light when I saw darkness, and pushed me when I needed pushing, and loved me when I was unlovable.

Daryn is a gift that only GOD could give. I have loved her from the first time I laid eyes on her. She will always be my Sweetheart, and I am proud to say, "I am Daryn's Daddy!"

Cj and Daryn, I love you both. I can never repay the debut I owe.

Thank you for the gift of salvation. *John 3:16*.

Revision History

Revision	Date	Description
1.0	03/04/2006	Content updates made during project work.
2.0	08/16/2006	Added terminology and testing descriptions.
3.0	10/10/2006	Update content as the project near the end of project work.
4.0	10/20/2006	Update content as project completed.
5.1	11/02/2006	Update content after the project was done.
6.0	11/27/2006	Update content prior to prior to requesting independent review.
7.0	01/08/2007	Incorporate feedback from Cj Howard and Sue Karlin.
7.5	01/15/2007	First complete draft submitted to MSC696B.
8.0	01/22/2007	Complete re-write based on Joseph Gerber's feedback.
8.1	01/26/2007	Reorganize chapter 1 and chapter 2.
8.2	01/28/2007	Second complete draft submitted to MSCS696B.
8.3	01/30/2007	
8.4	01/31/2007	APA Formatting changes.
8.5	02/05/2007	Incorporate feedback from Michael Singleton and Wei Soon.
8.6	02/06/2007	Incorporate feedback from Charlie Gragg.
8.7	02/14/2007	Incorporate feedback from Sue Karlin.
8.8	02/19/2007	Incorporate feedback from Dr. Richard Hursey.
8.9	02/23/2007	Incorporate feedback from Neil Holland, Greg Gulden, and David Shanes.
9.0	02/24/2007	Final draft completed and submitted to MSCS696B.
9.1	03/13/2007	Cut and paste scanned images of PDF documents: Certification of Authorship of Professional Project Work (page 2), Authorization to Publish Student Work (page 3), Advisor/Professional Project Facility Approval Form (pages 4 and 5) into this document prior to submitting final to Regis University.

Table of Contents

Table of Tables

Table of Figures

Building a Software Assembly Line

Chapter One

The software industry has yet to provide processes, methods, tools, and techniques that have been adopted industry-wide for increasing the probability that most software projects will be completed successfully, on time and within budget (Standish Group, 1994). Several software development (Yourdon, 1989; Jacobson, 1998; Beck, 2002) and project management (Christensen, 2001; Wysocki, 2003), and testing (Burnstein, 2002) methodologies exist and are documented, which partially resolve these problems. However, there is no "silver bullet" (Brooks, 1975/1995). The report titled "General Principles of Software Validation; Final Guidance for Industry and FDA Staff" (Center for Devices, 2002) reports on the analysis of 3,140 medical devices between 1992 and 1999. It was found that 242 (7.7%) recalls were attributed to software failures; and 192 (79%) of the failures were introduced after the original product was distributed. Yet, software validation is a requirement of the Quality System Regulation (QSR) and documented in Title 21 Code of Federal Regulations (CFR) Part 820, and 61 Federal Register (FR) 42602 (Center for Devices, 2002). This author hypothesized that these problems could be mitigated by using existing technology to create a "Software Assembly Line" that would continuously build and test automatically software products.

The basic concept of the Software Assembly Line is similar to a manufacturing Assembly line. As depicted in Figure 1 below, a manufacturing assembly line typically has several automated stations that operate either serial or in parallel to build a product. Raw materials are injected into a manufacturing assembly line at discrete points and are assembled into a final product. Each assembly station is calibrated, and the calibration data is used to determine

whether raw materials or assembly processes have been performed to specifications. Throughout the process acceptance tests are conducted, and the product is either accepted or rejected. The manufacturing assembly line must be validated before it produces final assemblies so as to ensure that the final product has been manufactured correctly

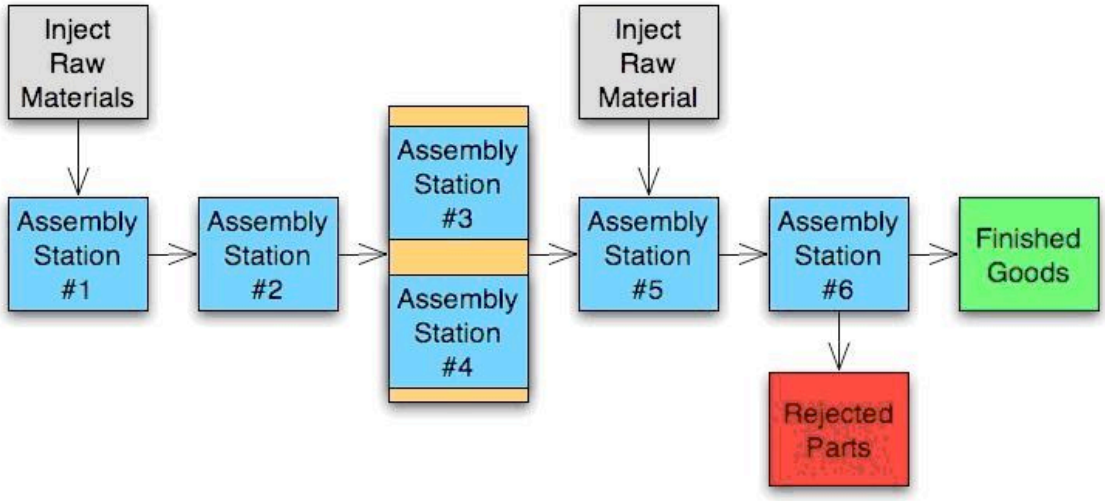

Figure 1. A conceptual manufacturing assembly line.

In Figure 1, raw materials are injected at stations #1 and #5. Each assembly station has unique upper and lower control limits that are used to determine whether a subassembly has passed or failed the assembly requirements tests for that station. Failures can occur at any station, and raw materials are not added, nor are assembly steps performed on any part that fails at any of the first four stations. Stations #3 and #4 are examples of parallel processing which accommodates workflows that are slow and would reduce the overall assembly line cycle time. Parallel assembly stations decreases the cycle time of the assembly step, thereby keeping the overall cycle time at a desired rate.

Figure 2 illustrates a Software Assembly Line that was completed in November 2006.

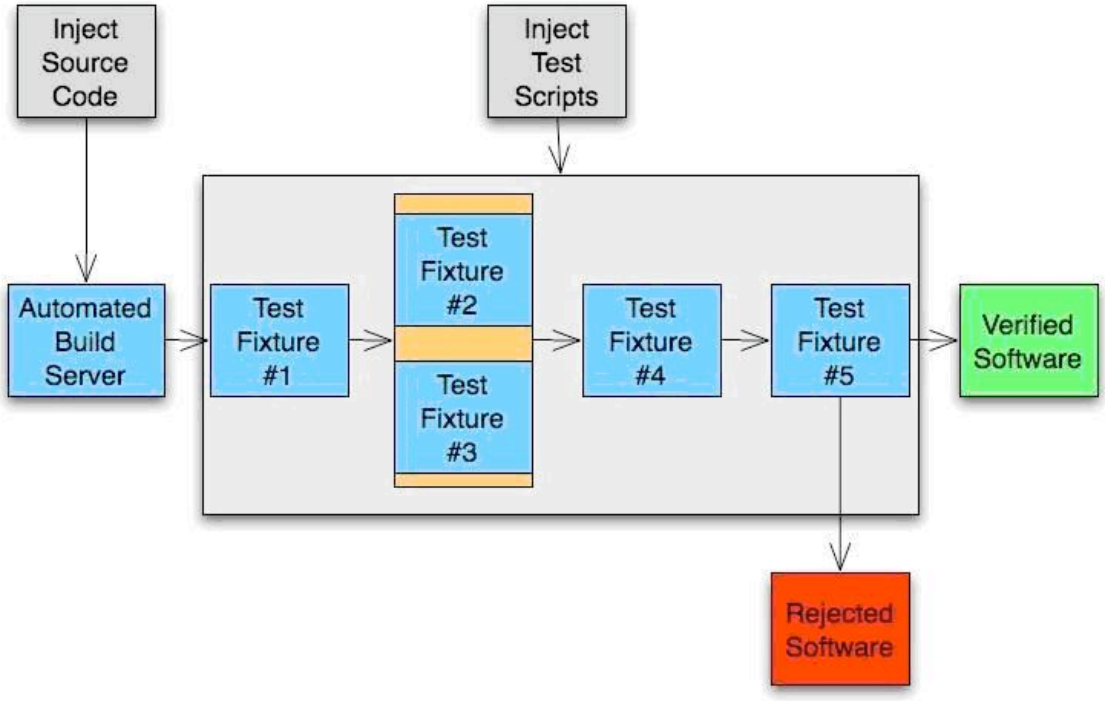

Figure 2. A conceptual software assembly line.

The Automated Build Server continuously monitors the Configuration Management for a change in source code. It then compiles and links software changes to produce the next available binary image. The binary image is moved to the Test Fixture #1 to determine whether it meets the entrance criteria needed for further processing. As in the manufacturing assembly line, Test Fixture #2 and #3 use parallel processes to run test scripts that take longer to run than the test scripts that run subsequently at stations #4 and #5. The Software Assembly Line "understands" the workflow and causes binaries to move from Test Fixture to Test Fixture. Failures can occur

at any test station, and binary images that either control limit at any Test Fixture are not subjected to further testing down the Software Assembly Line.

The Test Fixtures depicted in Figure 2 are the central nervous system to the Software Assembly Line. Test Fixture #1 was responsible for detecting a new binary images and testing them. It was also considered a Primary Test Fixture, which means it was responsible for pushing the binary images to subsequent Secondary Test Fixtures.

Figure 3 illustrates the operation of a Test Fixture. Each Test Fixture injects a new binary image and, when necessary, test scripts. The Test Fixtures export pass/fail status information, which was subsequently sent to recipients through an email notification system.

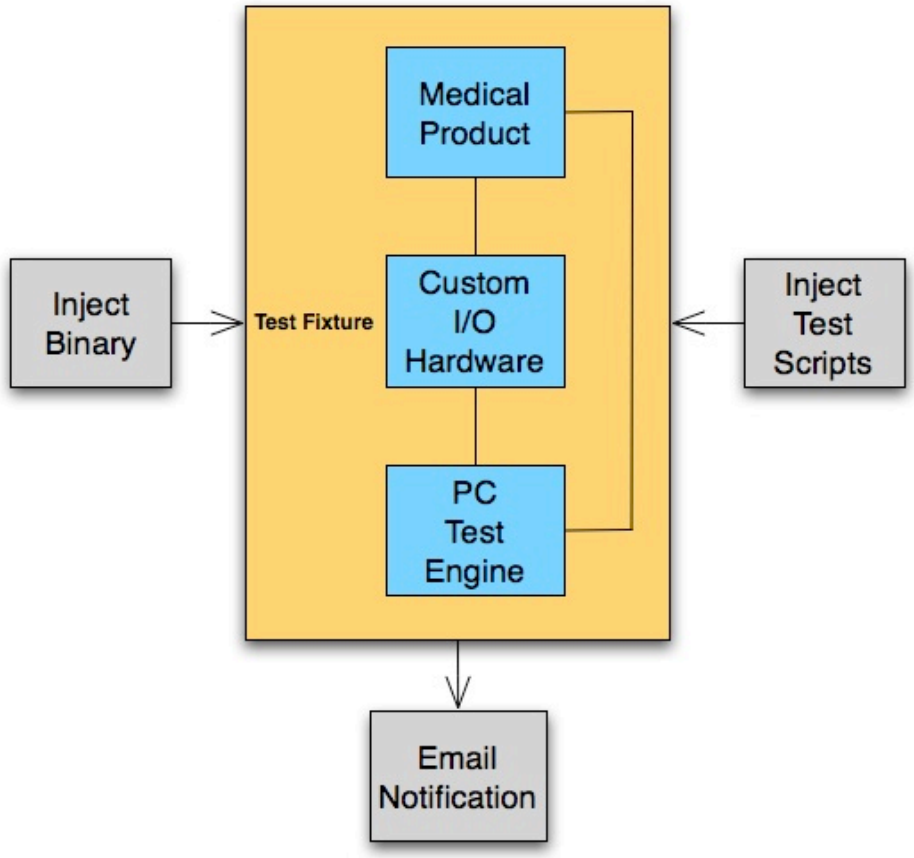

Figure 3. Generalized Test Fixture.

Each Test Fixture has three major components: (1) a medical device, (2) custom Input/Output (I/O) hardware, and (3) a personal computer (PC) Test Engine. The medical device was the hardware and software under test. Custom I/O hardware permits a Test Engine to control power, to pinch valves, to monitor Light Emitting Diodes (LED) status, and to monitor sounds emitted from the medical device. The Test Engine coordinates all activities within the Test Fixture.

Test Engines were PCs programmed with Tcl/Expect and Perl. Each Test Engine was configured to poll a specific directory for changed binary images. When one was detected, the Test Engine downloads it to the medical device. The software download process then caused the medical device to automatically turn off. The Test Engine, through the Custom I/O Hardware, detects that the medical device has been turned off and caused it to restart by cycling power.

The Test Engine has two connections with the medical device: an RS-232 serial port and an Ethernet port. The RS-232 serial port is used to issue commands to, and monitor output from, the medical device. The Ethernet port is used to monitor output from the medical device. The Test Engine was also connected to the custom I/O hardware via an RS-232 serial port. This connection was used to switch electrical power and pinch valves on and off, and to monitor speaker sounds and the light emitting diodes (LED) on the medical device.

The Test Engine ran its assigned test scripts on each binary image, then polls for a new one. Whether a new binary image is found or not, the Test Engine obtained the latest test scripts as well as the latest email notification lists it had to use to send pass/fail email test notification messages. Test scripts and email notification lists were updated to ensure each Test Engine was

running the latest approved testing scenarios. Each Test Script was written to elicit a specific behavior. In other words, each one has known inputs and expected outputs.

At the inception of this project, all the hardware and software technology that was needed to create the Software Assembly Line existed. However, the technology had not yet been deployed to continuously test and verify both hardware and software 24 hours per.

Statement of the Problem to Be Investigated and Goal to Be Achieved

The business is a medium size, international, pharmaceutical and medication delivery company. The company competes in a market where, "**...**revenues for IV therapy and vein access were worth more than $3.2 billion in 2005. At an average annual growth rate (AAGR) of 4.1%, the market will reach almost $3.8 billion by 2010." (Piribo Online, 2006, p. 1) Broadly, there are four types of medication delivery systems: injectables, pills, patches, and inhalants. Injectables are further subdivided into syringe and intravenous delivery methods. Intravenous delivery methods consist of either gravity fed or electromechanically powered IV pumps. Industry-wide, IV pumps deliver a wide range of medications that must be delivered within +/-5% accuracy to avoid catastrophic patient events.

The company designs and manufactures life-sustaining medical devices using a waterfall business process that was appropriate to regulated environments. Industry-wide device development took between 18 to 96 months to introduce a medical device to the market. The company operates at the lower end of the industry-wide range and delivers products within 48 months. Millions of dollars are spent during product conception and production in testing both hardware and software. Most testing methods are manual, and they are labor intensive, and they require specialty training, and they are hard to repeat. Because of these issues, the testing methods are not optimal.

A waterfall development process, which is defined in Chapter 3, was used to develop the medical product. Process validation, as defined by 21CFR820.75, was not applied to this process nor should it be. The Software Assembly Line was used to continuously build software and automatically test it on a medical device. It was designed to achieve process validation. To accomplish this it was documented, controlled, monitored, and reviewed. It has not been validated to 21CFR820.75, but all the necessary preconditions for process validation rigor have been achieved.

Relevance, Significance or Need for the Project

The Software Assembly Line contributed to the launching of the medical device and helped the company gain FDA acceptance and market approval. This was accomplished by using it to run automated tests during formal verification and validation. The Software Assembly Line was used to automatically run all test suites and regression suites. Future projects will require fewer cycles through formal verification and validation by using this Software Assembly Line.

Barriers and/or Issues

This project started during the third quarter of 2005, during the Design phase of the medical device project, and it had not been included during the budget planning process that occurred during the forth quarter of 2004. Without a budget, workable prototypes had to be demonstrated in order to obtain the funding needed to create the Software Assembly Line's infrastructure.

Elements, Hypotheses, Theories, or Questions to be Discussed/Answered

The main claim of this thesis is that the Software Assembly Line provides an effective, reliable, and predictable method to achieve continuous verification testing through automation.

This thesis demonstrates that the company should invest resources to formally validate the Software Assembly Line in accordance with regulation 21CFR820.75, Process Validation.

Limitations/Scope of the Project

This project was completed with management approval. The results were used during development efforts to determine whether the medical device was ready for both formal verification and validation. The results and data collected were not included in any formal reports that were submitted to the design history file because the Software Assembly Line had not been validated. This project did not validate the Software Assembly Line, but recommended that it be validated according to 21CFR820.75.

Definition of Terms

Glossary and acronyms have been separated into Table 1 and Table 2, respectively. Federal Regulations use terms differently than Software Development Life-Cycles do. For example, verification and validation have different meanings when used in regulations. Wherever a potential conflict existed, an expanded definition has been given to provide the reader a definition that is specific to regulations.

Glossary

Term	Definition
21CFR820.75	Process Validation: (a) Where the results of a process cannot be fully verified by subsequent inspection and test, the process shall be validated with a high degree of assurance and approved according to established procedures. The validation activities and results, including the date and signature of the individual(s) approving the validation and where appropriate the major equipment validated, shall be documented. (b) Each manufacturer shall establish and maintain procedures for monitoring and control of process parameters for validated processes to ensure that the specified requirements continue to be met. (1) Each manufacturer shall ensure that validated processes are performed by qualified individual(s). (2) For validated processes, the monitoring and control methods and

Term	Definition
	data, the date performed, and, where appropriate, the individual(s) performing the process or the major equipment used shall be documented. (c) When changes or process deviations occur, the manufacturer shall review and evaluate the process and perform revalidation where appropriate. These activities shall be documented. (Food and Drug, 2001)
Acceptance Tests	A black-box testing method used to prove the product is complete.
Black-box	A testing method that uses external interfaces to exercise system or sub-system.
Biomed Scenarios	A black-box testing method used to test workflows related to bio-medical scenarios.
Concept phase	The Concept phase is used to understand the customer needs, define the initial program requirements and architectural needs and document them in the Program Review Committee (PRC) Concept phase Review Document. High-level program schedules and funding requirements are defined and the program is approved through the PRC process.
Check-in	To send modified, new or deleted changes to a source code repository system.
Clinical Scenarios	A black-box testing method used to test workflows related to clinically relevant scenarios.
Debug	Identify and correct the root cause of a problem.
Debug Output	Programmed data sent to either a RS-232 or Ethernet port to aid in the debugging effort.
Definition phase	The Definition phase follows the Concept phase. It has the following outputs: (1) Product Requirements Document, (2) User Interface Requirements Document, (3) Software Requirements Document, (4) Risk Analysis, (5) Hazard Analysis, (6) Program Plan, and (7) Software Development Plan. This phase further refines the project schedule, deliverables, risks, resources and funding needs.
Design phase	The Design phase builds on the Definition phase and is used to design, document, write, and test a software product. Requirements, architecture, design, code and test are continuously redefined throughout the Definition phases.
Expect	Expect is a tool primarily for automating interactive applications. (Expect, 2007)
Installation Qualification	Establish by objective evidence that the facilities, equipment and systems, can be installed as intended.
over-the-shoulder	An informal review process that two or more persons use to review a software change using a computer.
Operational Qualification	Establish by objective evidence that the facilities, equipment and systems, as installed or modified, perform as intended throughout the anticipated operating ranges.
Performance	Establish confidence that the process is effective and reproducible.

Term	Definition
Qualification	(Glossary, 2007)
Random-Button-Pusher Test	A white-box test used to detect erratic user interface behavior brought on by unexpected user gestures.
Record and Replay	A white-box testing method used to capture keystrokes or touch screen events and replay them back to test a product feature or workflow scenario.
RS-232	In telecommunications, RS-232 is a standard for serial binary data interconnection between a DTE (Data Terminal equipment) and a DCE (Data Circuit-terminating Equipment). It is commonly used in computer serial ports. (RS-232, 2007)
Smoke Test	A series of white and black-box tests that are used to determine the system is adequately useful to groups outside of software development. These tests must achieve 100% pass rate to consider a smoke test successful.
Software Validation	Confirmation by examination and provision of objective evidence that software specifications conform to user needs and intended uses, and that the particular requirements implemented through software can be consistently fulfilled.
Stage Gate	The System Owner is responsible for calling a Stage Gate meeting between project phases to demonstrate, with objective evidence, the current phase has been completed successfully, the project is compliant with processes, and the project is ready to proceed to the next phase.
Sustaining phase	The Sustaining phase follows the Transfer phase. Software development efforts are limited to supporting on-market products when approved by program management. The Sustaining phase defines how validated software is modified and when re-validation is needed.
Tcl	Tcl (originally from “Tool Command Language”, but nonetheless conventionally rendered as “Tcl” rather than “TCL”; and pronounced like “tickle”) is a scripting language. It is commonly used for rapid prototyping. (Tcl, 2007)
Transfer phase	The Transfer phase follows the Design phase. The Transfer phase is used to both formally verify and validate the program. The software effort during this phase is to resolve any discrepancies and assist both the verification and validation teams with white-box testing methods where black-box testing methods are not adequate for verifying or validating that a requirement has been met. A software system must be validated before it can be transferred to production.
Unit Tests	A white-box testing method that is used to test class and/or function level features.
Validation	Establishing documented evidence that provides a high degree of assurance that a specific process will consistently produce a product meeting its predetermined specifications and quality attributes. Validation activities may consist of, but are not limited to, the

Term	Definition
	following elements: Installation Qualification (IQ), Operational Qualification (OQ), and Performance Qualification (PQ), Validation Plan, Validation Protocol, Validation Report, and Validation Tests.
Validation Plan	A written plan for validation that identifies the scope, approach, resources, schedules, the types and extent of activities, and the role risk management will play.
Validation Protocol	A written plan stating the details of how the validation will be conducted, including test parameters, product characteristics, production and test equipment, and decision points on what constitutes acceptable test results.
Validation Report	A summary document that reviews all the validation deliverables and activities against the Validation Plan and summarizes the evidence and conclusions to report that the system is validated and acceptable for its intended use.
Validation Tests	A black-box testing method used to prove User Needs have been met.
Verification	Confirmation by examination and provision of objective evidence that specified requirements related to a product or process has been met.
Verification Tests	A black-box testing method used to prove product specifications have been met.
White-box	A testing method that uses internal interfaces to exercise system, component, class or function-level features.

Table 1: Glossary of terms.

Acronyms

Acronym	Definition
AARG	Average Annual Growth Rate
Auto-V&V	Automation Verification and Validation (Auto-V&V)
Biomed	Bio-Medical
CFR	Code of Federal Regulations
CI	Continuous Integration (Fowler, 2006)
COTS	Commercial Off The Shelf
DCE	Data Circuit-terminating Equipment
DTE	Data terminal equipment
FDA	Food and Drug Administration
FR	Federal Register
I/O	Input / Output
IT	Information Technology
IQ	Installation Qualification
IV	Intravenous
OCR	Optical Character Recognition
OP	Operating Procedure

Acronym	Definition
OQ	Operational Qualification
PC	Personal Computer
PQ	Performance Qualification
PRC	Program Review Committee
R&D	Research and Development
SG3	Study Group 3
SDLC	System Development Life Cycle
SIQ	Software Installation Qualification
SLOC	Source Lines of Code
SOP	Standard Operating Procedure
Tcl	Total control language.
XML	Extensible Markup Language

Table 2: Acronyms.

Summary

This chapter describes the situation the medical device project was in when the project began. An overview of the Software Assembly Line has been provided, and it has been compared with a hypothetical manufacturing assembly line to illustrate how it works.

Chapter 2 reviewed published works that are relevant to the Software Assembly Line concept. Chapter 3 describes how the company uses its “waterfall” methodology and how the Software Assembly Line was constructed. Chapter 4 tells the history of the project. Chapter 5 discusses lessons learned and the recommendations made for consideration.

Chapter Two

Review of Literature and Research

Research for this project was done in two phases: at project onset and in preparation for writing this thesis. The former is described in the "Research Methods to be Used" section of Chapter 3, and summarized here.

Initial research was limited to informal interviews and literature reviews. The objective was to identify the companies existing needs and requirements and to assemble concepts and ideas relating to automated testing. Later research involved review of literature on process validation, especially as related to 21CFR820.75, Process Validation.

Overview of All Literature and Research on the Project

The author met privately with company team leaders and managers with responsibilities for verification of products, software development, and configuration management and found a few common themes regarding the software currently in use, such as: (1) it lacks stability; (2) it did not have adequate or reliable external interfaces; and (3) it was difficult to build.

The first two points did not require literature review but rather a determination of the root cause of the problem. Software development was using a "code-and-fix" model, which was the root of the first problem. The problem was resolved by slowing the development team down and having it start performing "over-the-shoulder" reviews for every software change.

The lack of reliable external and internal interfaces were the result of a series of miscommunications between the Software Engineering Team and Software Automation Team. The Software Engineering Team was not aware that the Software Automation Team was using output from the medical device, as described in Chapters 1, as the expected output for

automation tests. This was resolved by establishing a contract between the two teams by which medical device output was honored as external and internal interface contracts.

The third problem was the result of another miscommunication between two other groups: Configuration Management and Software Engineering. An automated build server resolved this problem. Research related to the build server is discussed in the next section.

The research done, while writing this thesis, was limited to the area of process validation. Because the author's conclusion recommend validating the Software Assembly Line in accordance with 21CFR820.75, it was necessary to perform a literature review so as to inform company management of the methods and techniques needed to achieve a validated Software Assembly Line.

Literature and Research that is Specific and Relevant to the Project

Martin Fowler's seminal work, Continuous Integration (Fowler, 2000, revised 2006), influenced the software industry to implement the best practices of continuous integration. Fowler (2000) credits McConnell (McConnell, 1996) as defining a daily build as a best practice and "…it has been long known as a feature of the Microsoft development approach…" Continuous build as defined by Fowler (2000) is "…a fully automated and reproducible build, including testing, that runs many times a day. This allows each developer to integrate daily thus reducing integration problems."

Fowler (2000) defines success criteria for automated daily builds as follows:

1. A centralized source code in one location
2. Automated build with one command to start
3. Automated test with one command to start
4. "Best Executable" is available to the team

Successful builds are defined as:

1. All sources are checked out from configuration management
2. All files are compiled
3. Resultant objects are linked and deployed
4. The system is started and a smoke test is run
5. All steps occur without human intervention and pass 100%.

Fowler also credits McConnell for defining the term, "Smoke Test," as: "Switch it on and see if smoke comes out."

"Process Validation" is a term "…used in the medical device industry to indicate that a process has been subject to scrutiny to ensure that the results of the process (a product, a service or other outcome) can be practically guaranteed" (Hojo, 2004, p4). Therefore, it is directly applicable to the Software Assembly Line project. Hojo further defines "process validation" as "…establishing by objective evidence that at process consistently produces a result or product that meets its predetermined requirements." (Hogo, 2004, p5) This is exactly what this author is recommending relative to the Software Assembly Line.

Hogo's work cited here was "Quality Management Systems – Process Validation Guidance" written by Study Group 3 (SG3) and endorsed by the Global Harmonization Task Force (GHTF). GHTF guidance documents provide "non-binding guidance to the regulatory authorities for use in the regulation of medical devices" (2004, p1), and are "intended to assist manufacturers in understanding quality management system requirements concerning process validation" (2004, p3). As Hogo points out on page 4, "The theory of process validation is reasonably straight forward, and guidance is provided for reaching decisions on whether to validate or not." This author believes that one uses guidance documents to find one's way

through the quagmire of regulations. Hogo has practical advice that harmonizes well with the author's recommendations.

Summary of What is Known and Unknown about the Project Topic

Volumes have been written both about software testing and how to manage it, and about process validation. This author did not find any work that directly tied software testing, development, verification and validation to the current development activities that are the subject of this thesis. As illustrated in Figure 4, all software development work occurs in the Design phase of medical device development. In this phase, automated testing is generally discussed to ensure that the software system is functioning properly. In "'Continuous Verification' in Mission Critical Software Development," Chang et al. (1997) describe continuous verification in a way that is closely related to the work that is the subject of this thesis. Chang's project achieved continuous verification with a robotic data tape system used in a test program for the B-2 Bomber. Continuous verification was used throughout a "waterfall" development model. However, that project did not attempt to achieve process validation.

The Contribution this Project Will Make to the Field

The goal of this project was to design and build an automated Software Assembly Line that can be validated according to regulation 21CFR820.75, Process Validation. The Design phase uses configuration management and software builds, smoke tests, verification tests, and validation tests, each of which are validated individually for their intended use.

The Software Assembly Line assembled configuration management, build processes, automated testing, test fixtures, and medical device software and hardware to achieve continuous verification. Automated test performed by the Software Assembly Line included: Smoke Tests,

Random-Button-Pusher Tests, Units Test, Clinical Scenarios, Biomed Scenarios, Record and Replay, Verification Tests, Validation Tests, Regression Tests, and Acceptance Tests.

The following contributions were made: (1) A deployed Software Assembly Line demonstrated the project was an effective mechanism to achieve continuous verification, that is cost-effective, repeatable and predictable. (2) Successful deployment proved that validation of the Software Assembly Line could be achieved. (3) As much as 90% of the requirements based testing could be replaced by automation.

Chapter Three

Methodology

As stated, the hardware and software technology were in place at project commencement. However:

- Ethernet communications between the PC Test Engine and medical device were not stable;
- The communications team was upgrading the hardware and software on the medical device to replace an on-board Ethernet controller with a separate microprocessor to perform external TCP/IP communications;
- A dormant USB (Universal Serial Bus) was being enabled between the main microprocessor and the new communications microprocessor;
- There was no planned budget for the Software Assembly Line.

To obtain funding to start the project, the author proposed to management the solution described herein including a demonstration of a small prototype. The project started with a simple goal: construct a system that would build and test the software and hardware for a medical device, and would demonstrate predictable and repeatable results through automation. The project became known as a "Software Assembly Line."

In addition, the Information Technology (IT) department had decommissioned three 6' x 36' x 17" racks that were converted to hold seven 1u rack-mount serves running Microsoft® Windows XP Servers, a medical device, and custom Input-Output boards. The Microsoft® Windows XP Servers hosted the PC Test Engines, Automated Build Servers and the email notification systems. After the demonstration of the small prototype, the full project was approved by management and was deemed economically feasible. To provide the infrastructure

needed to support the Software Assembly Line, additional hardware, air conditioning and electrical upgrades, costing approximately $75,000 were made.

Research Methods to Be Used

Research consisting of informal interviews and literature reviews was conducted between July 2005 and September 2005. Informal interviews were conducted with management, software engineers, software automation engineers, and verification engineers to understand what problems were blocking software development and software automation. The results were as follows:

Management needs were not being fulfilled because:

- The Software Team could not accurately predict schedules and schedule impact;
- Schedules and plans changed to frequently;
- The software had a high level of defects;
- The program was behind schedule and over budget.

Software engineers identified several significant problems:

- Compilation and linking issues that were occurring almost daily were preventing reliable software development and unit testing.
- Once built, binary images downloaded to the medical device were not stable and often prevented the medical device from functioning.
- The internal and external network communication software was not stable.

The Verification team indicated that:

- The software was not stable. Items that worked in one release were not working the next;
- Defects escaping development were excessively high.

Through the informal interview process, the author identified the following testing opportunities that could be fulfilled by the Software Assembly Line:

- Smoke Tests.
- Random-Button-Pusher Tests.
- Unit Tests.
- Clinical Scenarios.
- Biomed Scenarios.
- Record and Replay.
- Verification Tests.
- Validation Tests.
- Regression Tests.
- Acceptance Tests.

When the interviews were concluded a plan was identified to resolve each problem. The undertaking was to resolve each problem identified. The purpose was to create a Software Assembly Line that would continuously build software and automatically test it on multiple Test Fixtures that were configured to operate in a predetermined testing role. This effort started immediately. As problems were resolved and the Software Assembly Line was being constructed, objective evidence was also being produced in the form of automated test results that demonstrated the medical device software and hardware was maturing.

The primary effort was to assemble existing software and hardware components into a Software Assembly Line. Assembly work (building and configuring the three 19” racks) and software tool development needed to create the Automated Build Server, and update the Test Engine was done by company personnel.

Informal interviews and responses to weekly presentations to executive management revealed the following changes could be made during the project to incrementally improve the medical device software.

- Organize the Software Automation Teams efforts so testing scripts would be in line with software development efforts. These efforts were organized according to the following areas:
    - Basic Therapy
    - Bolus Therapy
    - Piggyback Therapy
    - Advanced Therapy
    - Non-Therapy
    - Event Data
    - Long running therapy (over 24 hours)
    - Unit Test Cases
    - Biomed Services
    - Edge condition and drug concentration algorithm accuracy
    - Dose rate calculation accuracy
    - Inter-Channel Sequencing Therapy
- Setup a continuous build system.
- Setup an automatically cycling Smoke Test to run all day long
- Integrate the existing software download mechanism into the Test Engine to automatically load the latest binary image produced by the build system and restart the smoke test.

- Have the Software Engineering Team resolve the Ethernet communication stability issues between the medical device and the Test Engine.
- Have the Software Engineering Team honor the internal and external interface contracts established with the Software Automation Team, which included:
    - Ensure that interfaces with the Test Engine were not broken as medical device evolved. Disabling interfaces had been the root cause of most interface issues.
    - Work with the Software Automation Team to identify and fix all test output data that was being streamed to either the RS-232 or Ethernet port, and to guarantee this information would be changed only in coordination with the Software Automation Team.
- The Software Automation Team focused their efforts on
    - Fixing and repairing Test Engine problems
    - Working with other teams to identify what could be automated to support the testing role each Test Fixture would perform.
- The hardware team would repair and provide ten functioning medical devices for use by the Software Assembly Line.

Literature was reviewed as needed throughout the project. Bits and pieces of useful information were found which allowed the author to make tough choices regarding what to automate and what not to automate. The author did not find a “silver bullet”. There is a vast amount of literature that describes how software engineers can perform Unit Tests within an Integrated Development Environment (IDE) and Test Driven Development (TDD) methods (Beck, 2007). Other methods were found describing Commercial-Off-The-Shelf (COTS)

products that design and perform automated tests (JUnit, 2007). Also, early defect detection methods were found also (Prevent, 2007).

Unfortunately, no published work was found that addressed continuous verification or attempted to prove the results could be validated according to 21CFR820.75, Process Validation. The project cited by Chang et al. (1997) was the closest to the work that the author proposed. The software effort required in Chang's work was relatively small: 7K (Source Lines of Code) SLOC compared to the 1.3M SLOC the author's project was designed to automatically test and verify.

This researched convinced the author that this work must be attempted. The results of this project are covered in Chapter 4.

Life-Cycle Models To Be Followed

The Software Assembly Line project used rapid prototypes within a waterfall process. Development of rapid prototypes began in September 2005 during the Design phase of the medical device project to deliver each requirement identified in Chapter 3. The author alone approved or rejected each prototype as it was delivered and demonstrated.

The following paragraphs describe the waterfall business process that was used to develop the medical device. Because the Software Assembly Line started during the Design phase (Figure 4) and without a budget, the author used rapid prototypes both to build the Software Assembly Line and to prove that the medical device was ready for transfer to production.

Iterations are used through each life-cycle phase to refine and redefine project activities. The process has the following phases: concept, definition, design, transfer, and sustaining, which are described below. A "Stage Gate" meeting is held and approved by management before going

from each phase to the next one. Projects proceed following the logical progression illustrated in Figure 6.

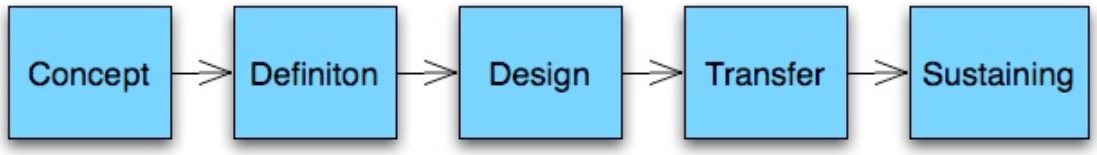

Figure 4. Software Development Life-Cycle

The Concept phase includes understanding the customer needs and defining the initial program requirements and architectural needs. This information is documented in the Program Review Committee (PRC) Concept phase Review Document. High-level program schedules and funding requirements are defined and the program is approved through the PRC process.

The Definition phase follows the Concept phase. It has the following outputs: (1) Product Requirements Document, (2) User Interface Requirements Document, (3) Software Requirements Document, (4) Risk Analysis, (5) Hazard Analysis, (6) Program Plan, and (7) Software Development Plan. This phase further refines the project schedule, deliverables, risks, resources and funding needs.

The Design phase builds on the Definition phase and is used to design, document, write, and test a software product. Requirements, architecture, design, code, and tests are continuously redefined throughout the Definition phases.

In the Transfer phase the program is both formally verified and validated. The software effort during this phase is to resolve any discrepancies and assist both the Verification Team and

the Validation Team with white-box testing methods where black-box testing methods are not adequate. Software systems must be validated before they can be transferred to production.

In the Sustaining phase software development is limited to supporting on-market products as approved by program management. This phase defines how validated software is modified and when re-validation is needed.

The waterfall method facilitates smooth transitions between phases. For the Software Assembly Line project, timely production of prototypes for unplanned development activity proved successful within the existing waterfall framework.

Specific Procedures

To ensure that the quality of the medical device software continued to improve, the author required mandatory on-line peer reviews of all changes to the software. Two or more software engineers were required to review every source change that was submitted to the configuration management system.

The Software Automation Engineers also reviewed changes to the Test Engine and updated test scripts as necessary to accommodate changes to medical device software.

Each sub-team that reported to the author identified an individual to be the point of contact for identifying and sorting out build issues when software builds failed. The Software Automation Team Leader informed these points of contact whenever a build failed. As the Automated Build System came on-line, the points of contact system was replaced with an email notification system.

Because the medical device software was not stable, weekend test runs were conducted on five to seven medical devices to determine whether the system could remain operational for

24-hours. This was increased to 48-hours once the 24-hour goal was consistently met. Finally, 72-hour test runs became mandatory.

To minimize costs, the Software Automation Team and the Hardware Team used spare parts when medical devices required rebuilding. The Information Technology Department decommissioned three 19" racks that were immediately acquired by the Software Automation Team to start building the physical Software Assembly Line.

When doctors and nurses visited the development site, this author and the Software Automation Team Leader would interview them to gain an understanding of clinically relevant testing scenarios. This technique was also used with the Hardware Team to understand biomedical and service testing scenarios. The Hardware Team was helped by the device manufacturer and the field service organization to define further testing scenarios.

The Software Automation Team was responsible for developing Test Engines and all Test Scripts automated by the Software Assembly Line. An offshore team was responsible for writing Test Scripts and executing them during formal verification and validation periods. The offshore team used the Test Engine produced by the Software Automation Team.

Formats for Presenting Results and Deliverables

The Software Assembly Line produced five functional prototypes designed by this author and implemented by the Software Automation Team, which reported directly to the author. The prototypes were built at the author's direction and then demonstrated.

Review of the Deliverables

The demonstration time-frame of the five functional prototypes as approved by the author was:

- October 2005: The first prototype demonstrated that changes made to a software module and submitted to the configuration management system are automatically detected, built, and prepared for manually run smoke tests.
- February 2006: The second prototype demonstrated that the output from the first prototype was automatically loaded onto a Test Fixture, and that the corresponding smoke tests were automatically run.
- April 2006: The third prototype used nine Test Fixtures to demonstrate the following automated tests: Unit Testing, Clinical Scenarios, Biomed Scenarios, Record and Replay, Verification Tests, Validation Tests and Regression Tests. These tests were documented in Chapter 3, Research Methods To Be Used, as possible testing opportunities. The prototype demonstrated that these tests ran around the clock and proved the Software Assembly Line was capable of producing repeatable results.
- June 2006: The fourth prototype demonstrated that a single Test Engine could control and manage four different medical devices simultaneously. This proved that the Software Assembly Line was scaleable.
- October 2006: The fifth prototype demonstrated a Test Engine that was configured as a primary/secondary workflow controller. This demonstration proved that the Software Assembly Line could direct a Test Engine to either test or skip the testing of a binary based on the overall pass/fail criteria at a given station.

Once the prototype was approved, it became an operational component of the Software Assembly Line. By November 2006, the operational Software Assembly Line consisted of the following components that ran every day all day:

- Automated Build System with electronic email notification and integration into the configuration management system was achieved. This integration included the synchronization with a Primary Test Fixture to deposit a new binary image for an automatic smoke test.
- A single Primary Test Fixture and nine Secondary Test Fixtures all integrated with the configuration management system to obtain the latest test scripts. The Secondary Test Fixtures automatically downloaded the latest binary image provided by the Primary Test Fixture.
- Test Scripts written and deployed for the following tests: Smoke Tests, Random- -Button-Pushers Tests, Unit Tests, Clinical Scenarios, Biomed Scenarios, and Record-and-Replay Tests.

Resource Requirements

The author was the Software Team Lead for the medical device software. He provided direct supervision and was responsible for both the software engineers and software automation engineers. During the project, the team expanded to a maximum of 65 people, then tapered down to 35. The author was the architect of the Software Assembly Line and defined the requirements for the system and the functional prototypes.

The author also directed and coordinated the weekly activities of each sub-team.

- The Clinical sub-team was responsible for all software that performed clinical therapies and interacted with the motor and power subsystems of the medical device.
- The Service sub-team was responsible for all software that configured and performed Biomed setup.
- The User Interface sub-team was responsible for implementing all workflows on a 6” x 9” touch screen.
- The Board Support team was responsible for all software activating the hardware. This included managing the physical infrastructure needed to mange the interfaces with the power supply sub-system, motor controller sub-system, and communication engine sub-system.
- The Software Automation sub-team was responsible for the development of the Test Engine software and Test Fixture setup, and for writing the Test Scripts identified in the deliverable section.
- An Automation Verification and Validation (Auto-V&V) sub-team was a consumer of the Test Engine. That group performed semi-automated testing based on requirements. The work was done offshore using an earlier version of the Test Engine that did not include the features listed in the deliverable section.

Outcomes

The desired outcomes were to have by November 2006 a functional Software Assembly Line that would operate everyday, all day, and a medical device that would be stable and ready for commercial launch in December.

The medical device was tested with the Software Assembly Line and was commercially available by December 2006. Between October 2005 and September 2006, the Software Assembly Line produced 2,061 software builds and automatically tested 750 test cases resulting in 1,545,750 test case executions which yielded an overall test case pass rate of 98.75%.

Summary

Because the medical device development cycle was using a "code-and-fix" model, proactive measures were needed to resolve problems. Incremental improvements were needed to be made daily while simultaneously minimizing the impact on both software development and Auto-V&V efforts. Research showed that no "*silver bullet"* was available. Software development efforts were redirected to support and meet the needs of both the Software Automation and Auto-V&V teams. Spare parts and decommissioned equipment were used to build the physical Software Assembly Line. Internal resources were used to resolve all software problems. A rapid prototyping system was used to mitigate risk and adjust priorities daily as necessary.

Chapter Four

Project History

This project began because the software project for which the author was responsible was not meeting program commitments with repeatability or reliability. In short, the software being developed was not stable. Hardware, software, and automation efforts had been ongoing for two and one-half years. Management could not rely on schedule predictions that were made by the Software Team. The need to find a solution to this problem was what led the author to start the project, provide architectural direction, and oversee the construction. This chapter details events and milestones and describes how the project evolved over time.

How the Project Began

The project started on July 18, 2005, the day the author assumed the role as Software Team Leader for the software development effort. The author found that development practices were not operating optimally. Successful compilations and links were occurring only randomly. The usefulness, stability, and reliability of the latest binaries were suspect. The development organization was using a "code-and-fix" model that was not predictable. The software development organization was not meeting schedules it published; employees' roles and responsibilities were confusing; and software project management and architectural leadership was challenged to organize work efforts and schedule predictions. Stress levels within the organization were high and even higher within the development group. Each individual was doing his or her, but were not adding incremental value to the project each day.

The software engineers had resorted to sending emails to the entire team when a build had failed to compile correctly. An attempt to resolve the build issues was to require each developer to integrate code onto their workstation and make all builds before checking their

work. This reduced the developers' cycle time to two check-ins per day, but it did not increase the likelihood that a build would actually work correctly. Nor did it guarantee that the build could actually be loaded and tested on the medical device. Another ineffective solution had been to assign to a single software test engineer the responsibility of manually producing the builds, manually loading them on to a medical device, and manually performing a smoke test. Overall, the software organization was not working effectively and efficiently together as a team.

How the Project Was Managed

As previously mentioned, as the Software Project Team Leader, the author was responsible for all medical device software development and software automation development. In addition, he was the architect for the Software Assembly Line. In this role, he defined abstract concepts that, when implemented, would benefit the company. He worked with the Software Automation Team Leader to convert ideas to actionable tasks that could be assigned to Software Automation Team members.

The author held weekly meetings with the entire Software Development Team, which included the Software Automation, Clinical, Service, and Communications Engine sub-teams. The Software Automation Team considered the Clinical, Service and Communications Engine teams as their customer. The Software Automation Team provided all the services needed to automatically build and test the software produced by the Clinical, Service and Communications Engine teams.

Weekly one-on-one meetings with the Software Automation Team Leader were held to review accomplishments, work-in-progress, and action plans, and to make adjustments as necessary. When significant work was completed, the author witnessed prototype demonstrations

and made suggestions which either reduced or increased the scopes of the prototypes. Approved changes were incorporated into the Software Assembly Line.

Management was provided demonstrations throughout 2006 as significant accomplishments were made.

Significant Events and Milestones in the Project

Table 3 below summarizes each significant event that occurred during the project. More detailed descriptions of the events follow the table.

Item#	Date	Description
1	09-2005	A Continuous Build System was created to automatically build the software whenever source code was submitted to the configuration management system.
2	02-2006	Assembled 19" racks with 7 test engines. Automated testing began.
3	03-2006	Merged different test engines into a common solution.
4	03-2006	Update test servers to support four Test Engines from a single server.
5	04-2006	Nightly test suites relative to Biomed and Service mode were enabled.
6	04-2006	Stable communications between Test Engine, Communication Engine and User Interface Controller achieved.
7	05-2006	Random-button pusher scripts were enabled to detect random errors in user gestures.
8	06-2006	Test Engine was updated to automatically download the latest binary image for testing purposes.
9	06-2006	Hourly Smoke Tests that were specific to Biomed, Service and Clinical mode were enabled.
10	07-2006	The Software Automation Team automated 32 unit tests the Software Engineers had been running manually as part of formal verification.
11	08-2006	Download Drug Libraries were added to test suites.
12	08-2006	The Communication Engines were configured to automatically upload status and event messages to a Company Server.
13	09-2006	Proto-type a master / slave workflow configuration with the Test Engines enabled.

Table 3: Dates significant events were completed.

A Continuous Build System was integrated into the Configuration Management System, to monitor source code and test script changes. Source code was refreshed on the build machine, and the Continuous Build system invoked the software product's standard compilation and link system to produce a new binary image. The Continuous Build System monitored build output and sent out email notifications. Email notifications were sent for both passed and failed builds. This was completed by September 2005.

Three 19" racks were assembled. These racks housed seven Test Engines controlled, by seven dedicated servers and associated network gear. The Test Engines automatically test software changes all day, every day. The test platforms execute various test scripts, which have been configured to test different aspects of the infuser. In addition, the test platforms are used to test "special" engineering development builds dedicated to characterizing troublesome bugs. This was completed by February 2006.

The Test Engine had morphed into three distinct engines developed by different teams between July 2003 and July 2005. A consolidated Test Engine was achieved by March 2006. This Test Engine was revalidated with an existing Test Engine Validation Plan.

A Test Engine enhancement was made to allow four Test Engines to coexist on a single Microsoft® Windows XP Server. Before this change, a single Microsoft® Windows XP Server could control only a single medical device. This was accomplished by making software modifications to the Test Engine and purchasing an Ethernet to Serial port converter that multiplexed one Ethernet connection input port into four serial output ports. The capacity of the

19” racks increased from seven medical devices to 28. A reduction in cost of $2,000 per Test Engine was realized. This enhancement was completed by March 2006.

The Software Assembly Line made nightly runs of Test Scripts that were specific to Service and Biomed operational modes of the medical device. The nightly runs were in place by April 2006.

The Test Engine relied heavily on a stable telnet session between the Test Engine and the Communications Engine. The telnet session was not reliable causing an unacceptably large number of aborted test runs and test failures. The telnet dropout issue was resolved to provide a stable communications pathway by April 2006.

The Software Automation Team implemented a Random-Button-Push Test Script that randomly pushed buttons available on the current medical device. Button presses were captured and replayed so as to duplicate found bugs. This Test Script discovered several navigational bugs and hard to reproduce navigational issues. This was completed by May 2006.

The Test Engine was modified to automatically download binary images from the Continuous Build System to the medical device. Three binary images were downloaded, namely: User Interface Controller, Pump Motor Controller, and Power System Controller. This was completed by June 2006.

The Software Assembly Line runs Test Scripts hourly that are specific to the Clinical, Service, and Biomed modes of the medical device. The hourly Test Scripts run in a simulated environment of the medical device. Hourly test runs occur between 8:00 am and 5:00 pm daily. These test runs are considered a one-hour Smoke Tests. The Test Engine automatically downloads the simulated environment from the Continuous Build System. This was completed by June 2006.

The Software Automation Team automated 32 requirements that previously had been conducted manually. The Software Assembly Line now tests these requirements with every new binary image. This was accomplished by July 2006.

The Test Engines were upgraded to exercise drug library download to the medical device. This accomplishment automated the stress testing of the built Extensible Markup Language (XML) parser used by the medical device. This came on line August 2006.

The Communication Engine on the medical device was configured to upload event and status data to the company server. Thus, actual data recorded by the medical device is reported to an external team as the result of the Test Engine executing automated test scripts. This was completed by August 2006.

A workflow was incorporated into the Test Engine. To accomplish this, Test Engines were configured as either a primary or secondary. Primary Test Engines monitored the output from the Continuous Build System. The latest binary images were automatically downloaded and put through an automated Smoke Test. The Primary Test Engines copied the binary images to Secondary Test Engines that has been configured to operate in a particular testing scenario. This enhancement reduced the man-hours with correcting and detecting test system failures. Additionally, Secondary Test Engines would continuously re-run the last Test Scripts and binary images until directed to load a new binary image by the Primary Test Engine. Leaving Secondary Test Engines in a continuous operational state did uncover regression errors with the same binary images and Test Scripts. This proved useful in detecting hard to reproduce errors. This was completed by September 2006.

Changes to the Project Plan

When the project first began, the author proposed both building and validating the Software Assembly Line. In FDA terms, product verification is done against requirements and product validation is done as needed by the user. These are different views of the same set of requirement specifications. Validation follows verification and the two are never done in parallel, without obtaining process deviation permits. Complicating matters is the concept called "tools validation" which line "process validation" must "practically guarantee an outcome." (Hogo, 2004, p4)

For example, assume that project X is a medical device for which Mosteller, DuBois and DuBois, Haycock, Gehan and Georga, and Boyd formulas for Body Surface Area (BSA) as documented by (Halls, 2004) are required to be implemented. Upon examination, one discovers the formulas are deterministic:

Mosteller: $BSA\ (m^2) = ([Height(cm) * Weight(kg)] / 3600)^{1/2}$

DuBois: $BSA(m^2) = 0.20247 * Height(m)^{0.725} * Weight(kg)^{0.425}$

Haycock: $BSA(m^2) = 0.024265 * Height(m)^{0.3964} * Weight(kg)^{0.5378}$

Gehan: $BSA(m^2) = 0.0235 * Height(m)^{0.42246} * Weight(kg)^{0.51456}$

Boyd: $BSA(m^2) = 0.0003207 * Height(m)^{0.3} * Weight(kg)^{(0.7285 - (0.0188 * LOG(grams)))}$

Each calculation could be proved manually using paper and pen, but this would not be repeatable or scaleable. So this gives rise to the question of what tool to use and how much effort is required to validate the selected tool. Per regulation, commercial-off-the-shelf (COTS) and developed tools must be validated for the intended use. Microsoft® Office Excel 2003 (Microsoft, 2003), herein referred to as "Excel", is selected.

One must validate that Excel properly works and the formulas have been written correctly. An Installation Qualification (IQ), Operational Qualification (OQ), validation plan, validation protocol and validation report must be written. The IQ describes how Excel is installed. The OQ demonstrates Excel meets the intended requirements. The validation plan identifies the scope and approach. The validation protocol describes how the tool will be validated. The validation report summarizes the evidence and conclusion to report the tool is validated and acceptable for its intended use.

Tool validation regulation is intended to ensure that inputs and outputs from the tool meet the user's needs. In this case, the user is a software engineer who is making decisions based on the output. This author considers the Software Assembly Line a tool. A tool whose intended use is to verify and validate a product, continuously. The intended users are software, automation, verification, and validation, management, quality and configuration management personnel.

After careful examination of the requirements to build and validate a Software Assembly Line, the author realized these were two separate projects. Therefore, the scope of the project was reduced to building the Software Assembly Line and proving it could be validated. Thereafter, the scope of the Software Assembly Line was tightly controlled.

Briefly, the author considered expanding the scope and directly tying this work in to the Metrics database and a site-wide dashboard, which is a project summary portal. While creating seamless integrations between the Software Assembly Line and these two other systems remains a workable long-term goal, the author ultimately considered it to be outside the scope of the initial configuration for the Software Assembly Line. However, integrations with these two systems were included in the recommendations for future work.

The Record-and-Replay function has constantly taken a back seat to other program demands. Work in this area has progressed, but a complete record-and-replay solution has not been delivered yet.

A special Bitmap Smoke Test had to be written to record and compare all screen shots as part of the smoke test. This increased the number of stations needed to run a smoke test from 1 to 2. The standard Smoke Test was designed to complete within one hour. Changing bitmaps and screen shots results in false-negatives reported by the Test Fixtures. The medical device software changed to output and the object hierarchy for each object represented on a screen. Test Scripts were updated to key off this output as opposed to being tightly coupled with the physical layout for any given screen. The Bitmap Smoke Test completes within approximately 90 minutes. It was used to detect screen changes and provides a list of differences between what was considered the "gold standard" screen shots and current screen shots.

Evaluation of Whether or Not the Project Met Project Goals

The Software Assembly Line exceeded the author's expectations and proved that it would be worth be worthwhile to expand efforts to validate the final output. The project did improve the overall effectiveness of the software development efforts by reducing the number of defects released to either the verification or the validation teams. Further, the Software Assembly Line was operational around the clock every day detecting software changes, building and testing them. The Software Engineering and Software Automation Teams relied on daily email notifications stating whether software builds and automated tests had competed correctly.

The work began with a single medical device that was being used to manually smoke test the latest software build. When the project was completed, seven workstations were deployed controlling nine test stations, and three continuous build servers were operational. The company

budgeted to expand capacity to 30 medical devices in 2007. The company also plans to reuse the Software Assembly Line on the next project, which will be built on the same hardware and software system architecture.

Discussion of What Went Right and What Went Wrong in the Project

Rapidly prototyping the system was instrumental to the success of the project. Significant milestones were achieved monthly and incremental success was observed almost daily. It turned out that small incremental steps allowed the project to add value continuously while minimizing risk and making course corrections as needed.

On the other hand, record-and-replay proved to be more difficult than expected. The ability to record-and-replay was achieved, but the desired ability to compensate for time was not met. Recording keystrokes and touch screen events throughout a 24-hour scenario was not difficult, but there were problems replaying them. This was because two different time adjustment algorithms were needed: one to adjust for playback rate and one to adjust for programmed elapsed time. The following example illustrates the problem.

First, the playback speed could not be adjusted, so a simple delay timer was used. This did not have the desired effect because system failures had been uncovered when user interface gestures were to fast or slow.

Second, a programmed time algorithm adjustment did not yield satisfactory results. A medical device might be programmed to last for 1 to 96 hours, but this is not practical for automation purposes. Typical automation scripts cycle medical devices between 1 and 30 minutes.

Since a satisfactory algorithm was not identified to address these problems, record-and-replay was sparingly and only used within its known limitations. Oddly enough, the results of

combining record-and-replay with the random-button-pusher tests prove useful. Some very unrealistic and difficult to reproduce user gestures were identified and corrected.

There were also problems integrating the Software Assembly Line with the Metrics database. The Metrics database was designed for low volumes of data and the Software Assembly Line created high volumes of data. This integration effort was dropped.

Discussion of Project Variables and Their Impact on the Project

The Software Automation Team's primary assignment was to write test scripts and execute them manually to assist the Software Engineering Team with Unit Tests. A rudimentary method was invented to execute one or more scripts throughout the day. At one point, almost the entire software development efforts halted so that the software engineers could assist the automation execute some 1,700 test cases manually. These test cases did not have any test scripts. As a result, one month was lost to the Software Assembly Line effort.

Findings and Analysis Results

Large systems integration efforts must be carefully planned so that deliverables arrive at the right place and according to an established schedule. Ad-hoc efforts with cursory, simplified planning might work for small projects, but with large projects they are likely to be excessively time consuming, inefficient, and costly. Sub-system interaction is paramount in large-scale system development, and sub-system interface contracts must be established and honored. It is management's responsibility to ensure that deviations are not permitted. Likewise, it is clear that sub-teams must meet and communicate frequently.

Automated Testing does not just happen. The software system under test must be designed and developed with testing in mind. Nor is Automated Testing cheap. It is

management's responsibility to provide adequate budget and supply resources for testing and building software that can be automatically tested.

The author's Software Assembly Line operates all day every day and detects problems within a few hours of detecting software changes. The Software Assembly Line concept should be implemented for all future software projects.

Summary of Results

The Software Assembly Line was a successful project. It conclusion of this project, it operated on an actual medical device and in a simulated environment continuously 24 hours a day, every day without human intervention.

Chapter Five

Lessons Learned and Next Evolution of the Project

This project started with a simple goal: to construct a system that would build and test the hardware and software of a medical device and to demonstrate predictable and repeatable results through automation. In addition, it was expected that the project would improve the operational efficiency and effectiveness of the software development organization.

What You Learned from the Project Experience

Ideally, this project should have been planned and budgeted at the same time the medical device software project began. Without budget and resources, prototypes and demonstrations had to be developed so as to keep the work effort moving forward and to establish credibility for both the software engineers and the software automation engineers.

There was resistance to change. The development culture was slow to accept new ideas. Skeptics did not initially understand the potential value of the Software Assembly Line concept. Because continuous builds and testing was done around the clock, became fearful there jobs would be eliminated.

Within the first month after the first prototype was deployed, these skepticisms and fears were abated. The software engineers began to rely on the continuous builds and the email notifications indicating builds and tests had occurred, and the configuration management team was able to predictably and reliably complete formal software builds.

Ultimately, the Software Assembly Line was so successful that it was funded throughout the remainder of the medical device project. It was realized that the Software Assembly Line increased the repeatability and reliability of the medical device project. Future projects are

expected to continue to use the Software Assembly Line and to tailor it to meet their specific needs.

What You Would Have Done Differently in the Project

As previously mentioned an offshore team was responsible for writing test scripts and performing automated tests during formal verification and validation periods. This author would have sent the Software Automation Team Lead to that location earlier in the project life cycle to better understand the difficulties the offshore team was having with the Test Engine.

Because of this trip, it was determined that 60% of all automation could be achieved within 24 hours. All obstacles that prevented 100% automation within 24 hours were identified and plans were made to resolve them.

Software automation relies heavily on white-box testing methods. To build automated scripts, intimate knowledge of the interfaces and inner workings of the medical device is needed. This author believes this can be achieved only if the software development and automation engineers can work closely together to develop testable systems.

The Software Assembly Line needs a budget. Because it operates on real hardware, sufficient products and spare parts are needed to keep it operational.

Because the Software Assembly Line required significant hardware and software upgrades, and configuration changes, the project would have benefited from a hired a laboratory technician to maintain them.

Discussion of Whether or Not the Project Met Initial Project Expectations

The author's expectations were met. The Software Assembly Line detected numerous problems that previously were difficult to reproduce and isolate but were easier to detect and resolve after the Software Assembly Line was implemented.

The project was also useful to the Software Engineering teams. Because they knew their software was being tested round the clock, they gained confidence in the automated testing. As a result, they provided guidance to software automation engineers who were testing certain interactions and combinations that were either difficult to reproduce or were susceptible to human error because the exact testing steps had to be repeated exactly to solicit a particular system behavior.

The Software Automation Team was able to take over formal white-box unit testing from the Software Engineering Teams. Test Scripts were written to replace the manual unit tests. These tests were executed daily.

What the Next Stage of Evolution for the Project Would Be If It Continued

Incorporating the changes outlined in this section would augment the Software Assembly Line in the following ways.

1. Remove all OCR (Optical Character Recognition) tests. The medical device software was modified to send all objects, locations, and text messages to the serial port. Performing an OCR text comparisons is time consuming, prone to error, and is tied to specific screen coordinates, screen color and font sizes.
2. Remove all Bitmap comparisons during operational tests. As with OCR tests, these tests are sensitive to change, and they mask real software problems when bitmaps changed resulting in false-negatives.
3. Implement button location testing so that the location of each button will be identified during the test and the test engine is updated to include current button locations.

4. Complete the record-and-replay feature. Define two time-compensation algorithms, one to compensate for touch screen entry rate, and the other to compensate for programmed therapy time.
5. Reorganize test teams into two distinct groups, one supporting early development testing and the other supporting Auto-V&V testing. Establish a head count for each team. Use lessons learned from the medical device project to define the test strategy for the development group. Test script development done early in the project life-cycle must focus on providing unit test overage. Auto-V&V test script development must focus on providing test coverage that demonstrates requirements have been met.
6. Define the test platform for future development projects so as to determine whether the existing test platform should be enhanced to accommodate new projects or a new test platform should be chosen. The new platform could be built from scratch or built with a purchased tool.
7. Decide if the strategy would include both a hard (real hardware) and soft (simulated hardware) target interface.
8. Design a new "universal" test engine, and determine whether it should be based on an existing test engine code or built from scratch.
9. Merge soft and hard test engines. If the current test platform is used on future development projects, this will reduce the amount of maintenance that would be required for separate test engines.
10. Integrate the Software Assembly Line with a Metrics database mentioned in Chapter 3 and implement the sample reports below.

- A Dashboard metric report showing the Test Fixture, Testing Role, Build Number, Pass Rate and Elapsed time as depicted below:

Test Fixture	Testing Role	Build Number	Pass Rate	Time At Fixture (min)	Elapsed Time (min)
1	Build System	92.1.8.110	100.00%	37	37
2	Smoke Test	92.1.8.109	100.00%	28	65
3	Random Button Pusher	92.1.8.108	99.99%	45	110
4	Unit Tests	92.1.8.107	98.75%	57	167
5	Clinical Scenarios	92.1.8.106	94.32%	28	195
6	Biomed Scenarios	92.1.8.105	89.28%	15	210
7	Record and Replay	92.1.8.104	100.00%	58	268
8	Verification Tests	92.1.8.103	95.25%	73	341
9	Validation Tests	92.1.8.102	97.24%	53	394
10	Acceptance Tests	92.1.8.01	99.58%	52	446
	Last Released Build	92.1.8.100	99.58%	446	446
	Last Rejected Build	92.1.8.105	89.28%	210	210

Figure 5. Sample Dashboard Report

- A Build Comparison Report comparing build-over-build results as shown below:

Test Fixture	Testing Role	Build 92.8.1.105	Build 92.8.1.100	Delta
1	Build System	100.00%	100.00%	0.00%
2	Smoke Test	100.00%	100.00%	0.00%
3	Random Button Pusher	99.99%	99.98%	0.01%
4	Unit Tests	98.74%	98.75%	-0.01%
5	Clinical Scenarios	94.32%	94.32%	0.00%
6	Biomed Scenarios	89.28%	100.00%	10.72%
7	Record and Replay	100.00%	100.00%	0.00%
8	Verification Tests	95.27%	95.25%	0.02%
9	Validation Tests	97.25%	97.24%	0.01%
10	Acceptance Tests	99.59%	99.58%	0.01%
	Elapsed Time	505min	485min	20min

Figure 6. Sample Build Comparison Report

11. Validate the Software Assembly Line.

Conclusions and Recommendations

The conclusion of this work is to validate the Software Assembly Line in accordance with regulation 21CFR820.75, Process Validation. Applying the rigor needed to obtain process validation provides the conclusive proof the Software Assembly Line has provided results that are predictable, repeatable, and have met predefined acceptance standards. A validated Software Assembly Line reduces the manpower and cost needed to manually verify and validate a medical product. This occurs because many manual process steps are eliminated by the Software Assembly Line.

The author recommends that a Software Automation Department be created as part of R&D organization. This new department should report to R&D site director. It is recommended that this new department be staffed, equipped and funded no later than the fourth quarter of 2007. This department would have three responsibilities. (1) They would manage and operate the Software Assembly Line. (2) They would enhance the Software Assembly to fit on-going automation needs. (3) They would control, monitor, review, and document the Software Assembly Line so it could achieve 21CFR820.75, Process Validation.

Summary

The Software Assembly Line successfully met the objects defined by the author. It was delivered by November 2006. It improved the quality and predictability of the software development efforts during 2006 by continuously building the software and automatically testing it on multiple hardware configurations. As a result, the number of cycles required for formal verification and validation was reduced. The Software Assembly Line produced 2,160 software builds and automatically tested 750 test cases resulting in 1,545,750 test case executions which yield an overall test case pass rate of 98.75%.

References

Beck, K. (2002). *Test-Driven Development By Example* (1st ed.). Boston, MA: Addison-Wesley.

Brooks, F.P. JR. (1995). *The Mythical Man-Month Essays on Software Engineering Anniversary Edition* (8th ed.). Chapel Hill, NC: Addison-Wesley. (Original work published 1975)

Burnstein, I. (2002). *Practical Software Testing: a process-oriented approach*. New York, NY: Springer.

Business Communications Company. (2006). U.S. Market for IV Therapy and Vein Access. In *Strategic Report* (BCC040, p. 1). London, United Kingdom: Business Communications Company. Retrieved February 9, 2007, from Piribo Online Business Intelligence for the BioParama Industry Web site: http://www.piribo.com/publications/technology/us_market_for_iv_therapy_vein_access.html.

Center for Devices and Radiological Health, Food and Drug Administration. (2002). *General Principles of Software Validation; Final Guidance for Industry and FDA Staff* (Version 2) [This guidance outlines general validation principles that the Food and Drug Administration (FDA) considers to be applicable to the validation of medical device software or the validation of software used to design, develop, or manufacture medical devices]. Available from http://www.fda.gov/cdrh/comp/guidance/938.html

Chang, T., Danylyzsn, A., Norimatsu, S., Riveria, J., Shepard, D., Lattanze, A., Tomayko, J. (1997). *"Continuous Verification" in Mission Critical Software Development.* Hicss, p.273, 30th, Hawaii International Conference on Systems Sciences (HICSS) Volume 5: Advanced Technology Track. Retrieved February 13, 2007, from IEEE Computer Society Web site: http://csdl2.computer.org/persagen/DLAbsToc.jsp?

resourcePath=/dl/proceedings/&toc=comp/proceedings/hicss/1997/7734/05/7734toc.xml&DOI=10.1109/HICSS.1997.663184

Christensen, M. (Series Ed.), & Plummer, D. (Vol. Ed.). (2001). *The Project Manager's Guide to Softwaer Engineering's Best Practices*. Los Alamitos, CA: Angela Burgess.

Expect. (2007). In *Wikipedia: The Fee Online Encyclopedia.* Retrieved February, 10, 2007, from http://en.wikiperdia.org/wiki/Expect

Food, & Drug Administration, Department of Health and Human Services. (2001). Code of Federal Regulations. In FDA (Ed.), *Title 21--Food and Drugs* (21CFR820.75). Washington, DC: Author.

Fowler, M. (2000, September). *Continuous Integration (original version)*. (Available from Martin Fowler, http://www.martinfowler.com/) Retrieved February 9, 2007, from Martin Fowler Web site: http://www.martinfowler.com/articles/originalContinuousIntegration.html

Fowler, M. (2006, May). *Continuous Integration.* (Available from Martin Fowler, http://www.martinfowler.com/) Retrieved February 9, 2007, from Martin Fowler Web site: http://www.martinfowler.com/articles/ContinuousIntegration.html

Glossary. (2007). *Glossary of Computerized System Software Development Terminology.* (Available from U.S. Food and Drug Administration, http://www.fda.gov) Retrieved February, 10, 2007, from http://www.fda.gov/ora/inspect_ref/igs/gloss.html

Halls, S., B. (2006, May). *References and formulas used by the Body Surface Area Calculator.* (Available from Steven B. Halls Professional Corporation, http://www.halls.md/) Retrieved February 9, 2007, from Halls MD Web site http://www.halls.md/body-surface-area/refs.htm

Hogo, T. (2004). *Quality Management System – Process Validation Guidance* (pp. 3-4). The Global Harmonization Task Force: Study Group 3. Retrieved February 9, 2007, from Global Harmonization Task Force Web site: http://www.ghtf.org/sg3/inventorysg3/ sg3_fd_n99-10_edition2.pdf

Jacobson, I. (Series Ed.), & Booch, G. (Vol. Ed.). (1998). *The Unified Software Development Process* (1st ed.). Boston: Addison-Wesley.

JUnit. (2007). In *Wikipedia: The Fee Online Encyclopedia.* Retrieved February, 10, 2007, from http://en.wikiperdia.org/wiki/JUnit

McConnell, S. (1996). *RAPID Development: Taming Wild Software Schedules* (1st ed.). Redmond, WA: Microsoft Press.

Microsoft. (2003). Microsoft® Excel 2003. (Available from Microsoft®, http://www.microsoft.com) Retrieved February 19, 2007, from http://www.microsoft.com/products/info/ product.aspx?view=22&pcid=ec28c9b6-60c3-4e85-82a6-d1be1e0848dc&crumb=all

Prevent. (2007). *Coverity Products.* (Available from Coverity, http://www.coverity.com) Retrieved February, 10, 2007, from http://www.coverity.com/html/products.html

RS-232. (2007). In *Wikipedia: The Fee Online Encyclopedia.* Retrieved February, 10, 2007, from http://en.wikiperdia.org/wiki/RS-232

Standish Group. (1994). *The Chaos Report* (Software development projects are in chaos, and we can no longer imitate the three monkeys -- hear no failures, see no failures, speak no failures, pp. 1-4). West Yarmouth, Massachusetts: The Standish Group Report. Retrieved February 9, 2007, from Project Smart Web site: http://www.projectsmart.co.ui/docs/ chaos_report.pdf

Tcl. (2007). In *Wikipedia: The Fee Online Encyclopedia.* Retrieved February, 10, 2007, from http://en.wikiperdia.org/wiki/Tcl

Wysocki, R. K. (2003). *Effective Project Management* (3rd ed.). Indianapolis, ID: Wiley Publishing.

Yourdon, E. (1989). *Modern Structured Analysis* (S. Papanikolaou, Ed.). Englewood Cliffs, NJ: Prentice-Hall.