Developing An Integrated Process Methodology For Interim Software Releases

Tim Woods
Southern Methodist University
420 Deer Run
Keller, TX 76248
Phone: 817.777.5238
twoods@smu.edu

Dr. Jerrell Stracener
Southern Methodist University
P.O. Box 750123
Dallas, TX 75275-0123
Phone: 214.768.1535
jerrell@smu.edu

Abstract

This paper covers factors that make production software releases successful – design complete, requirements complete, testing complete, customer expectations set, etc., and how a production software release process may not be fitting for an interim software release due to the state of the program during an interim release – one or all of the production release success factors possibly being incomplete, open system problem reports, interactions with other systems – hardware, communications and software interfaces, schedule interactions, resource constraints, etc. A discussion will follow on the need for an integrated process methodology that factors the incomplete nature of a program during an interim software release into the release decision methodology. The goals for the integrated process methodology will be discussed along with the next steps in developing the integrated process methodology for interim software releases.

Introduction

Today's complex systems are becoming more and more integrated, as evidence by the growing field of Systems of Systems (SoS). Consequently, software is being integrated with other processors within its own system and across interfaces within the total system itself, increasing the complexity and integration required for software releases.

SoS Adds Complexity

SoS, as the name implies, is a system comprised of other systems. Creating a system composed of other systems adds additional complexity and integration challenges. For instance, cars today may have 50 microprocessors controlling everything from the engine to the air bag [1]. Every microprocessor runs its own software and probably interfaces with additional microprocessors, driving additional complexity and integration pains. The Drive By Wire [2] technology for future cars, will only increase the complexity and integration challenges. In the past cars could be serviced by mechanically inclined individuals who did not mind getting their hands dirty. Today, one practically needs a degree in software engineering to service cars.

Aircraft have always been complex, integrated systems, but today, as more systems integrate with each other, the aircraft is becoming more complex and tightly integrated. Not long ago, the flight control software (commands surfaces to keep the aircraft flying) could be released with minimal to no integration testing with avionic software (controls the mission). Today the flight control and avionic software are tightly integrated providing advanced functions. This integration demands that even the smallest of software changes drives system and integrated testing to insure the software changes did not detrimentally affect the aircraft in some unforeseen manner.

The complexity and integration requirements of a SoS affects the system's software and its safety implications. As Leveson [3] points out:

> Today we are building systems – and using computers to control them – that have the potential for large-scale destruction of life and the environment: Even a single accident may be disastrous.

Today's added complexity, additional requirements, and criticality of software, means the decision of when to release software is becoming as complex as the software itself. This paper will explore a software release methodology that considers the complexity, integrational aspects, and criticality of today's software.

Software Complexity

Software is complex and becoming more complex daily. As an example, take the Joint Strike Fighter (JSF) program. In March of 2006 in a report from the General Accounting Office (GAO) to Congressional Committees, it was reported that the JSF program would develop 19 million lines of code [4]. In March 2007, the GAO reported the program would develop 22 million lines of code [5]. In one year the estimate increased by 3 million lines of code or 16%. Just think of the complexity added in those previously unaccounted for 3 million lines of code. The JSF software was to be delivered in 5 different blocks, but the number of actual software releases was not given.   It can be assumed that the software releases will number more than the delivery blocks.

Looking at the number of lines of code can give an idea of software complexity, the more lines of code, the more complex the software. Looking at Microsoft's LOC count shows an interesting trend [6]:

> Real systems show no signs of becoming less complex. In fact, they are becoming more complex faster and faster. Microsoft Windows is a poster child for this trend to complexity. Windows 3.1, released in 1992, had 3 million lines of code; Windows 95 has 15 million and Windows 98 has 18

> million. The original Windows NT (also 1992) had 4 million lines of code; NT 4.0 (1996) has 16.5 million. In 1998, Windows NT 5.0 was estimated to have 20 million lines of code; by the time it was renamed Windows 2000 (in 1999) it had between 35 million and 60 million lines of code, depending on who you believe.

Windows Vista, Microsoft's latest operating system, reportedly contains 50 million lines of code [7]. The number of software releases for a product with 50 million lines of code has to be large. Imagine performing only one software release for a product with 50 million lines of code.

Software Releases

Given today's integrated environment, releasing production software is an accomplishment in itself. With a production release, the design is complete, testing is complete, requirements are verified, outstanding problems are mitigated, contractual obligations have been met, the schedule no longer is a plan, it is the actuals for the program, and significant management oversight – sometimes known as help – is provided, making the path for a production release familiar and the process well known. Accompanying a production release is a sense of accomplishment for a job well done and possibly the end of the program.

With all that said, defining production release, as used in this paper, is required. A literature search will discover many terms and definitions related to software releases [8, 9, 10, 11]. For the purpose of this paper, production software release will be defined as a release to the end customer that is validated and verified to meet all the requirements. Along the same lines, an interim software release is a release that is not fully verified or validated to all the requirements. Customer, as used here, is defined as a user of the software. A customer could be internal or external to the company. An end customer is the customer that receives the software after all verification and validation activities are complete.

In today's integrated, SoS environment, it would be difficult, if not impossible, to proceed through a production software program of any size with only a production software release. The complexity and integrated nature of SoS almost requires interim releases before the production release.

If the path to production release is well known and familiar, does it necessarily follow that the production software release path/process is adequate for interim software releases? Production software releases benefit from the completeness of the design, testing, requirements and problem mitigations, interim releases usually do not have those luxuries. An interim release usually contains partial functionality and may even occur before the design is complete and may be used to complete requirement verification meaning requirements may not be verified. Because design may be on-going, testing may not be complete, requirements may still require verification, and outstanding high severity problems may not be mitigated, a production release process may not suffice for an interim release. Today's integrated SoS environment along with program schedule pressures add to the complexities of interim release decision making.

Integrated Process Methodology

An integrated process methodology is being developed for assisting in the decision making regarding when to release interim software. The integrated process methodology will consider the incomplete state of the program that exists for an interim release and additional factors that could affect a release such as interfaces, problem reports, resources, requirements, software criticality, and schedules. The integrated process methodology will assist system development programs in determining the optimal time to produce an interim software release that supports its intended purpose, given multiple release paths and multiple integrated software products, while considering the factors mentioned above.

The proposed integrated process methodology is not meant to replace software planning, but aid in the software release decision process. The software plan would be used as an input to the integrated process methodology decision matrix to assist in determining the optimal release path for a specific interim software release. Nor is the process methodology meant to solve the question of when to release the software, but to allow the decision makers to make better decisions regarding when to release software. The methodology's benefits will be especially useful as the decision of when to release software becomes more difficult. Keeney's [9] take on difficult decisions and analysis:

> More Difficult decision problems are naturally more difficult to analyze. This is true regardless of the degree to which formal analysis (i.e., use of models as a decision aid) or intuitive appraisal (i.e., in one's head) is used. However, as complexity increases, the efficacy of the intuitive appraisal decreases more rapidly than formal analysis.

Software release decisions are difficult by themselves, but when combined with the problems SoS introduces, there may be too much information required to properly process the decision. The decision maker may then use simplified mental strategies, without using decision analysis methods [10]. The integrated process methodology would be used to analyzed the information provided and aide in the decision making process, with the goal of replacing non-productive decision methodologies currently in use, like BOGSAT (Bunch Of Guys Sitting Around Table). Ideally, the integrated process methodology will provide an analytical methodology to aid in the software release decision process with the hopes of replacing multiple smaller software releases with fewer, more integrated releases.

The goal of the process methodology is to reduce software releases. That's a good thing, right? If it is just software, can't it be released anytime? While it is true that software can be released anytime, cost and schedule normally constrain the number of software releases for a particular program. Normally, releasing software incurs both a schedule and monetary cost. It takes a finite amount of time to make, build, release, document, and minimally test the release. During the release, the resources used (people, computers, labs, etc.) are not available to perform other tasks (incurring schedule costs) and must be paid for their time (incurring monetary cost). Consequently, the fewer software releases needed, the less the cost to the program.

Future work includes defining a generic interim software release process, developing the integrated process methodology, verifying the process' decision matrix, verifying the integrated process methodology, and optimizing the process methodology.

References:

[1] Karim Nice (2001, April 11) How Car Computers Work, Retrieved April 09, 2007 from http://auto.howstuffworks.com/car-computer.htm .

[2] Forte, Rob (2005, May 9) Driving By Wire *Autonet.ca*, Retrieved September 30, 2007, from http://www6.autonet.ca/Parts/Systems/story.cfm?story=/Parts/Systems/2005/05/10/1033945.html.

[3] Leveson, Nancy G., Safeware System Safety and Computers, Addison-Wesley Publishing Company, Inc., New York, New York, ISBN 0-201-11972-2, © 1995.

[4] Joint Strike Fighter: DOD Plans to Enter Production before Testing Demonstrates Acceptable Performance, GAO-06-356, March 15, 2006.

[5] Joint Strike Fighter: Progress Made and Challenges Remain, GAO-07-360, March 15, 2007.

[6] Schneier, Bruce (2000, March, 15). Software Complexity and Security. *Crypto-Gram Newsletter*, Retrieved September 29, 2007, from http://www.schneier.com/crypto-gram-0003.html.

[7] Lohr, Steve and Markoff, John (2006, March, 15). Windows Is So Slow, but Why? *New York Times* , Retrieved September 29, 2007, from http://www.nytimes.com/2006/03/27/technology/27soft.html?_r=1&oref=slogin#.

[8] Summerville, Ian, Software Engineering, Seventh Edition, Pearson Education Limited, Essex England, ISBN 0-321-21026-3, © 2004

[9] Bays, Michael E., Software Release Methodology, Prentice Hall PTR, Upper Saddle River, New Jersey, ISBN 0-13-636564-7, © 1999.

[10] RTCA/DO-178B. "Softare Considerations in Airborne Systems and Equipment Certification", December 1, 1992.

[11] IEEE Software Engineering Coordinating Committee,(SWECC). 2001. Software Engineering Book of Knowledge. http://www.swebok.org/.

[12] Keeney, Ralph L., "Decision Analysis: An Overview" Operations Research, Vol. 30, Iss. 5, pp. 803-838, September 1982.

[13] Goodwin, Paul and Wright George, Decision Analysis for Management Judgment, Third Edition, John Wiley & Sons Ltd., West Sussex, England, ISBN 0-470-86108-8, © 2004.

Author Biographies:

Tim Woods is currently a PhD in Applied Science candidate in the Southern Methodist University (SMU) Systems Engineering Program. He is currently leading a project to assist in developing a PhD in Systems Engineering program for the SMU Systems Engineering Program, while finishing the necessary course work for his PhD.

Currently he is working in the defense industry as a Systems Engineer and has supported several major fighter aircraft programs and worked commercial aircraft programs. Prior to re-joining the defense industry, Tim spent three years working for a systems engineering software tool company and performed systems engineering consulting for customers across the commercial and defense industries. Tim is a current member of INCOSE.

Mr. Woods earned MS degrees in Engineering Management and Systems Engineering from SMU and a BS in Electrical Engineering Michigan Technological University.

Jerrell T. Stracener is the founding Director of the SMU Systems Engineering Program. He also teaches graduate-level courses in Probability & Statistics, Reliability Engineering, Statistical Quality Control & Systems Analysis and conducts systems engineering research, consulting and training.

Prior to joining SMU full-time in January 2000, he was employed with Vought/Northrop Grumman for 31 years. Jerrell was a reliability engineer and ILS program manager on many advanced aircraft programs including the Lockheed Martin Joint Strike Fighter and the B-2.

Jerrell has actively promoted systems engineering & analysis, and reliability, maintainability, supportability & logistics through his leadership as founding member & chairman of the SAE RMSL Division and through participation INCOSE, AIAA & SOLE.

Dr. Stracener earned PhD & MS degrees in Statistics from SMU and a BS in Math from Arlington State College (now UTA).