Changing Safety-Critical Software

Leanna K. Rierson
Federal Aviation Administration

ABSTRACT

Changing software to implement new features, correct problems, update functions, or replace obsolete hardware components (e.g., microprocessors) is a way of life in today's industry. A systematic software change process is essential for the maintenance and reuse of safety-critical software. This paper presents activities to consider when changing software in safety-critical systems. The focus is on the aviation industry; however, the concepts are also applicable to other safety-critical domains, such as medical or nuclear.

INTRODUCTION

Changes to software are a way of life in today's environment. A 1988 paper presented in the Proceedings of the Sixth Annual Pacific Northwest Software Quality Conference emphasized that 40 to 70% of the total life-cycle costs associated with a software intensive system are maintenance activities; i.e., changes made to software [1]. Not much has changed in twelve years - in fact, the need to change software has actually risen. At a recent conference with developers of aircraft engine controls, one manufacturer claimed that over 90% of their software development activity consists of changes to already approved software. In the aviation industry, derivative products are very common. These derivative products tend to reuse as much software as possible, adding new features or corrections, as needed.

When software changes occur in safety-critical systems, great care must be taken. The changes to in-service software must be carefully planned, analyzed, implemented, controlled, and verified/tested. Additionally, steps must be taken to assure that a change will not negatively impact other functions in the system or on the aircraft. This paper will address the typical steps and activities involved in a systematic software change process. The change process should be initially planned and then faithfully followed to assure the safety of the software application and system. This paper focuses on changes to in-service aviation software, but is also applicable to other safety-critical domains, such as medical or nuclear.

CHANGE CONSIDERATIONS

Change should be considered during the initial development. In order to successfully change safety-critical software, two things should be considered: 1) the utilization of design features and tools that ease the burden of change; and 2) a well-documented change process that is both systematic and executable.

The Software Engineering Institute (SEI) "Workshop on the State of the Practice in Dependably Upgrading Critical Systems" explored ways to dependably upgrade software. One area of focus at the workshop was the "design for upgrade" concept. The SEI workshop emphasized that "most 'new' systems are updates to existing (legacy) systems rather than completely new systems" [2]. Therefore, designing for upgrade can significantly reduce the safety and cost impact of future changes. The following should be considered when designing for upgrade [2]:

- Architectural design that includes ease of changes should be considered (e.g., design simplicity, modularity, high cohesion, and low coupling).
Automated tools for design and verification should be implemented.

Methods to identify and isolate the impact of an upgrade should be explored (e.g., keeping design data and traceability current).

Tools to identify the dependencies among different system components should be used (e.g., system modeling tools used to evaluate the dependencies of system components and changes to software).

Cost models should be used to anticipate changes and to evaluate their impact.

Figure 1 identifies a comprehensive change process. There are ten steps typically found in a change process. During the original development the change process and each step should be carefully planned and documented. Modifications to the change process should be incorporated into the planning documents, as needed, for specific project needs. When a software change actually occurs, the planned and documented change process should be followed. The ten steps of the change process are described below.

**Step 1: Propose Changes**

The first step of the change process is to identify the proposed change(s). Typically, a number of changes are proposed at the same time in order to optimize the change effort. These changes should be documented in a consistent manner (e.g., software change request, problem report, or similar document). The proposed changes may include requests to fix bugs, to add new features, to modify existing features, etc.

**Step 2: Perform Initial Change Impact Analysis**

Not all changes are technically possible, financially feasible, or even necessary to implement. Each change is initially assessed to determine whether it will be implemented, deferred, or closed with no action. Obviously, considerations for technical feasibility, safety, cost, and customer satisfaction weigh heavily in the decision. An initial change impact analysis is a vehicle to help with this assessment. Each change is analyzed to determine the potential impact. There are many components of the change impact analysis. Some of the analysis can be done roughly to initially determine the feasibility of the change, but should be repeated after the software change is implemented to assure the original performance, safety, and resource constraints have not been compromised.

The Federal Aviation Administration (FAA) has recently completed policy on change impact analysis in a document entitled, “Guidelines for the Oversight of Software Change Impact Analyses Used to Classify Software Changes as Major or Minor” (FAA Notice N8100.85). The purpose of the FAA policy is to help developers of safety-critical software and the FAA to assess the extent and impact of software changes. This assessment then determines the data to be updated and the effort needed for regression testing, reviews, analyses, inspections, etc. for the software change.

The FAA determined that the items below should be addressed by the change impact analysis, as applicable. The items fall roughly into two categories: performance-related analysis and safety-related analysis. The following ten items are generally related to the overall performance characteristics of the system [3]:

- 1) **Traceability analysis** to identify the requirements, design elements, code, and test cases and procedures that may be either directly or indirectly affected by the change. Forward traceability identifies design components affected by the change. Backwards traceability helps determine other design features and requirements that may be inadvertently affected by the change. Overall, the requirements traceability helps determine the impact of change on the software project.
• 2) Memory margin analysis to assure that the memory allocation requirements are not altered, the original memory map is maintained, and adequate memory margins are maintained.

• 3) Timing margin analysis to assure that the original timing requirements, central processing unit (CPU) task scheduling requirements, system resource contention characteristics, and interface timing requirements are met and that adequate timing margins are maintained.

• 4) Data flow analysis to identify any adverse affects due to changes in data flow and coupling between and within components. In order to perform the data flow analysis, each variable and interface affected by the change should be analyzed to assure that the original initialization of that variable is still valid, that the change was made consistently, and that the change does not affect any other usage of that data element. This can be a costly and difficult process, if large global sections are used.

• 5) Control flow analysis to identify any adverse effects due to changes to the control flow and coupling of components. Task scheduling, execution flow, prioritization, and interrupt structure are examples of the items that should be considered.

• 6) Input/output analysis to assure that the changes have not adversely impacted the input and output requirements of the product. Things such as bus loading, memory access, throughput, hardware input, and device interfaces should be considered.

• 7) Development environment and process analyses to identify any change(s) which may adversely impact the software product (e.g., compiler options or versions and optimization change; linker, assembler, and loader instructions or options change; or software tool change).

• 8) Operational characteristics analysis, such as evaluation of changes to displays and symbols, performance parameters, gains, filters, limits, data validation, interrupt and exception handling, and fault mitigation to assure that there are no adverse affects.

• 9) Certification maintenance requirements (CMR) analysis to determine whether new or changed CMRs are necessitated by the software change. During the original certification of an aviation product, CMRs are identified. For example, the brakes may need to be inspected after 100 landings. If a change to software affects a CMR, it should be addressed during the change process.

• 10) Partitioning analysis to assure that the changes do not impact any protective mechanisms incorporated in the design. If architectural means are employed as part of the partitioning scheme, the change must not affect those strategies. For example, data should not be passed from a less critical partition to a more critical partition.

The FAA also determined that the change impact analysis should assess whether the change could adversely affect safe operation of the system or product. The following are examples of areas that could have an adverse impact on safety or operation (specific examples of each can be found in the FAA Notice) [3]:

• 1) Safety-related information is changed.

• 2) Operational or procedural characteristics of the aircraft are changed in a manner that could adversely affect flight safety as a result of the software change.

• 3) New functions or features are added to the existing system functions that could adversely impact flight safety or could affect the basis for the original certification approval.

• 4) Processors, interfaces, and other hardware components of the operational environment are changed in such a way that safety or the software could be adversely affected. An international Certification Authorities Software Team (CAST) paper entitled, “Guidance for Assuring the Software Aspects of Certification When Replacing Obsolete Electronic Parts Used in Airborne Systems and Equipment,” provides additional guidance for assessing hardware changes that could affect software [4].

• 5) Software life cycle data (e.g., requirements, design, code, architecture, test cases/procedures) is significantly changed in such a way that it could adversely affect safety. This analysis is facilitated when safety-related software requirements are identified and a safety assessment is maintained.

Lastly, the FAA determined that the change impact analysis should document software life cycle data (e.g., requirements, design, architecture, source and object code, test cases and procedures) affected by the change and verification activities (e.g., reviews analyses, inspections, tests) needed to assure that no adverse affects on the system are introduced during the change.

Step 3: Develop an Implementation Strategy

Once software changes to be implemented into a previously-developed baseline are identified, an implementation strategy should be developed and executed. The implementation strategy should specify the change process, software life cycle data
to be updated, activities to be implemented, resources needed, schedules to be applied, etc.

A software modification can be handled like a mini-development." Plans should be established for development, software quality assurance, software configuration management, and verification/testing. Once the plans are established, the changes should be implemented following those plans.

The implementation strategy should also describe the verification activities that will be needed to verify the changes and to verify that there are no adverse effects on the system. The change impact analysis should address how changes which could adversely affect performance or safe operation of the system or aircraft will be verified, such that the changed and unchanged software will continue to satisfy their requirements for safe operation. These verification activities may include reviews, analyses, regression testing, requirements-based testing, bench testing, ground and flight testing on the aircraft, etc., including re-evaluation of existing analyses, re-execution of existing tests, and new test procedures and cases (for added functionality or previously deficient testing).

Additionally, the implementation strategy should describe the resources needed to perform and assure the changes, the schedule for changes and verification activities, the software life cycle data to be updated, and any other programmatic details needed for success.

Step 4: Implement Changes

The implementation of software changes should follow the documented plans. During the actual implementation of changes, there may be additional issues identified. These might require modification to the change impact analysis or plans.

Many software developers desire to implement multiple software changes at the same time. It is difficult to determine the correctness of each change and to perform a thorough change impact analysis, if concurrent changes are made to the baseline. However, software changes can be implemented simultaneously (provided they do not affect the same areas of the code), as long as each software change is incorporated individually into an existing baseline. To assure correctness of each change, the change should be verified both individually and as a part of the overall system.

Step 5: Verify Changes

Once the changes have been implemented, they should be verified. For purposes of this paper, "verification" includes reviews, inspections, walkthroughs, analyses, and tests of software.

Figure 1 divides the verification process into three tasks (5A, 5B, and 5C). These tasks are detailed below:

Task 5A: Inspect, Review, or Analyze Changes

This task includes many of the non-testing aspects of the verification process (i.e., reviews, analyses, inspections, and walkthroughs). Edward Kit describes these as the "human examination or review of the work product" [5]. In this task, the software life cycle data (e.g., requirements, design, architecture, code, test cases and procedures) are reviewed for accuracy and consistency.

Task 5B: Perform Regression Testing

Regression testing is another aspect of the verification process that must be addressed when software changes. Webster’s dictionary defines regression as "a trend or shift toward a lower or less perfect state.” Boris Beizer defines regression testing as "any repetition of tests (usually after software or data change) intended to show that the software's behavior is unchanged except insofar as required by the change to the software or data” [6]. Software progresses through several versions before one is ready for release. Regression testing is performed on each version of software. The biggest issue is how to determine which software tests in Version N need to be run, if they were already tested in Version N-1. Any specific change can (a) fix only the problem that was reported, (b) fail to fix the problem, (c) fix the problem but adversely affect some other function or aspect that was previously working, or (d) fail to fix the problem and adversely affect something else [7].

It is typically not possible to re-run every test on every version of software, so an analysis should be used to determine which tests should be run on the interim versions.

Joy Shafer’s paper entitled, “Regression Testing Basics” provides some insight that is useful for planning regression testing. She writes that "regression testing finds many bugs. Studies have shown that changes and error corrections tend to be much more error prone than the original code in the program, in much the same way that most copy errors that make it to print were introduced during edits rather than in the original draft” [8]. Shafer goes on to list some of the most common types of regression test [8]:

• Bug verification tests – run to verify that the fix for a bug addresses the problem and does not introduce additional problems.
• Build acceptance tests – tests run to make sure that a build is ready to go to the test team.
Regression test pass with a regression test suite – running regressions tests that have been automated.

Regression test pass on closed bugs – rerunning the regression tests even after the bugs have been “fixed.”

Regression test pass without a test suite – manually running regression tests.

Shafer recommends that the most important tests be run first in order to quickly validate operation and assess risks. She also encourages the use of automated test suites to help reduce the time factor in re-running regression tests [8].

Another paper on regression testing by Christian Molnar discusses what should be tested and when it should be tested. Molnar recommends that regression testing be run in parallel with other development activities. This approach helps to address errors early. However, this approach also poses the question of “How many tests need to be re-run?” That is, as the project matures, do all regressions tests need to be re-run? Molnar offers these “general rules” for applying regression testing [9]:

- A test that has passed twice should be considered as regressed, unless the code has been changed.
- A test that has failed once should not be re-executed unless the developer informs the test team that the defect has been fixed.
- For tests that have already passed once, the second execution should be reserved for the final regression pass, unless frequent changes to the code indicate otherwise.
- The final regression test should not consist of 30% to 40% of the total number of tests in the suite.

Although Molnar’s rules provide a good foundation for implementing regression testing, they are not entirely applicable for the safety-critical domain. For a safety-critical system, the change impact analysis should lead to a set of regression tests that are unique to the set of changes being proposed. All of these identified regression tests should be run on the final version of the software prior to release.

Task 5C: Perform Other Verification

In addition to the inspections, reviews, analyses, and regression testing, there will be other types of verification activities to be performed. Some of these activities will be performed at the software level, some at the system level, some at the integrated system level, and some at the aircraft level. For example, requirements-based tests, acceptance tests, bench tests, ground and flight tests on the aircraft, structural coverage analysis, etc., may need to be performed. These additional verification activities will vary from project to project, depending on the extent of the change and the function(s) affected. These tests should be planned after the change impact analysis is performed and agreed upon early in the project.

Step 6: Finalize Change Impact Analysis

Step 2 described the initial performance of the software change impact analysis. Early in the change process it is impossible to fully evaluate the impact of the software changes. Therefore, once the changes have been implemented and verified, the change impact analysis should be finalized.

Step 7: Report and Analyze Problems

Each problem discovered during the previous steps should be documented in a problem report. In his paper, “The Bug Reporting Process,” Matt Baskett emphasized the importance of quality problem reports. He stated that the report should include a title, severity, description, steps to reproduce a bug, actual results of the test, expected results of the test [10]. Additionally, the problem report should include a place to identify affected documents, problem analyses, and proposed resolution.

Step 8: Address Problems

Problems documented in problem reports should be addressed in some manner. The problem can be fixed or determined to have no safety, performance, or operational affects. Many projects finish with open problem reports; however, there needs to be some method to determine which problems should be addressed and which can be deferred to a subsequent software release. In safety-critical systems, the software engineer, the systems engineer, and a safety expert should assess each open problem report to determine if it affects the safety, performance, or operational capability of the system. These problems should be addressed prior to release of the product.

Step 9: Fix Problems, As Required

As mentioned above, not all problems are fixed; however, those problems that are fixed should once again go through the change control and verification processes.

Step 10: Release Software

Once the necessary changes have been implemented, problems have been addressed, software has been re-verified, and documentation has been updated, the software is ready for release.
CONFIGURATION MANAGEMENT AND QUALITY ASSURANCE

Throughout the change process, software configuration management and software quality assurance should be applied to the project. It is essential to know exactly what problems have been fixed, what version of software was tested, what configuration of tests were run, and what version of software and data was approved for release. Hence, good software configuration management is essential. Software quality assurance also provides a valuable monitoring function by having independent reviewers overseeing the process to assure compliance to the approved plans.

SUMMARY

It is important to have a defined, structured, and rigorous process, when changing software in safety-critical systems. Some minimal steps needed for successful software changes have been discussed in this paper. A change process should include, at a minimum, change impact analysis, change planning, change implementation and verification, problem reporting and analysis, a process to assess and address problems, software quality assurance, and software configuration management.

ACKNOWLEDGEMENTS

Many thanks to the following people for their review of and input into this paper: Cheryl Dorsey (Digital Flight); Will Struck (Federal Aviation Administration); Mike DeWalt (Certification Services, Inc.); Kelly Hayhurst (NASA Langley); Uma Ferrell (Ferrell and Associates Consulting); and Jozef van Baal (RLD, Netherlands).

REFERENCES


