Workshop on the State Of the Practice in Dependably Upgrading Critical Systems April 16-17, 1997

David P. Gluch
Charles B. Weinstock
(Editors)
August 1997

SPECIAL REPORT
CMU/SEI-97-SR-014

Special Report
CMU/SEI-97-SR-014
August 1997

Workshop on the State of the Practice in Dependably Upgrading Critical Systems April 16-17, 1997

David P. Gluch

Charles B. Weinstock

(Editors)

Dependable System Upgrade

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through SAIC/ASSET: 1350 Earl L. Core Road; PO Box 3305; Morgantown, West Virginia 26505 / Phone: (304) 284-9000 / FAX: (304) 284-9001 / World Wide Web: http://www.asset.com/sei.html / e-mail: webmaster@www.asset.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / Attn: BRR / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218. Phone: (703) 767-8274 or toll-free in the U.S. — 1-800 225-3842).

Table of Contents

Acknowledgments

The workshop was a success due to the efforts of many. We'd especially like to thank Lorraine Nemeth and Leona Kass for their assistance in organizing and coordinating the workshop activities.

The quality of the workshop was greatly enhanced through the efforts of the working group chairs. Jack Goldberg, now retired from SRI International, Walt Heimerdinger, from the Honeywell Technology Center, and Charlie Westerfield, of the Harris Corporation, all did a marvelous job of running their subgroups. Additionally, in each group an SEI attendee was designated to take detailed notes. These notes were to serve two purposes: to help the chairs produce their out briefs during the workshop; and to serve as source material for this report. We'd like to thank Peter Feiler, Mike Gagliardi, and Mark Klein for their efforts.

While the editors had the privilege of assembling it, this report is the work of all of the workshop participants. It captures their comments, concerns, and ideas and represents their collective wisdom. The editors would like to thank the participants for the opportunity to work with them in addressing this important engineering problem.

Workshop on the State of the Practice in Dependably Upgrading Critical Systems

Abstract: This report describes the results of the *Workshop on the State of the Practice in Dependably Upgrading Critical Systems* held April 16-17, 1997 at the Software Engineering Institute. The workshop addressed a broad spectrum of issues associated with dependably and cost-effectively upgrading systems, primarily those with reliability or real-time requirements.

1. Introduction

On April 16 and 17, 1997 a Workshop on the State of the Practice in Dependably Upgrading Critical Systems was held at the Software Engineering Institute (SEI) in Pittsburgh, Pennsylvania. A total of 27 technical professionals representing 20 U.S. and international organizations participated in the working sessions. Included in the group of participants from industry, academia, and government were eight members of the technical staff at the SEI. Participation was by invitation only. A complete list of participants can be found in Appendix 0.

The purpose of the workshop was to explore the need for a discipline of dependable systems upgrade and assess the nature and scope of the upgrade problem. A series of questions and a preliminary framework (shown in 0 Framework) describing the upgrade problem were distributed to participants before the workshop. These were intended as guides in establishing a common vocabulary and context, while not unduly constraining the discussion. Changes in the direction, content, and nature of the outcome based upon the professional judgment of the team were encouraged and expected. An open exchange of ideas was evident throughout all of the sessions.

1.1 Background

Upgrade problems exist throughout the software engineering community. These problems are having a significant impact on the cost and capabilities of systems. The following examples illustrate these problems:

- At 39 seconds after launch, the Ariane 5 self-destruct mechanism activated, obliterating the rocket. (The estimated cost of the 10-year Ariane 5 program was on the order of 7 billion dollars.) The Ariane 5 was an upgrade of the Ariane 4. The upgraded software, based in part on the Ariane 4 software, could not handle the higher velocities of the Ariane 5 [Ariane 96].
- America Online's computer systems went down at 4am EDT on 7 Aug 1996. Service was reportedly restored sporadically 19 hours later, around 11pm EDT. The crash was caused by new software installed during a scheduled maintenance update [AOL 96].

Beyond the dramatic impacts of unsuccessful upgrades, there are basic economic factors motivating the need to address upgrade issues. Studies have shown that from 40 to 70% of the total life-cycle costs associated with a software system are in maintenance activities, i.e., changes made to the software [Ostrand 88]. One estimate of the cost of software maintenance shows that as much as two-thirds of software production costs is in maintenance [Leung 90].

This backdrop provided the impetus for the workshop and a foundation for exploring the complex area of dependably upgrading critical systems.

1.2 Goals and Objectives

The goals of this workshop were to

- get the best opinions and ideas from all participants
- formulate an accurate and comprehensive perspective on the problem space
- gain some insight into the directions for solutions

The goals involved both broad issues affecting the software engineering community and SEI-specific needs. The broad issues included explorations into the

- *nature of the problem*: defining the nature and scope of the problem, considering technical, operational, and managerial perspectives
- *critical issues*: identifying the critical issues, considering their importance and inter-relationships across application domains
- *framework*: establishing a technical, operational, and programmatic structure for system upgrade

From the SEI perspective, the workshop provided a basis upon which to define the nature and extent of the efforts of the SEI's Dependable System Upgrade initiative. The results of the workshop are being used to help establish the detailed technical areas and strategy for supporting SEI clients as they address the problems associated with dependably upgrading systems.

1.3 Questions to Answer

Participants were given a set of questions in advance to stimulate their thinking. Specific questions are covered in the sections to follow, but in general they encompassed the following:

- What are the critical issues to be addressed?
- What can a framework look like?
- What other issues should be identified?

1.4 Outcomes

The outcomes of the workshop were expected to include some of the following:

- a record of questions, responses, and discussions
- a listing of critical issues (technical, management, and business)
- a compilation of solutions, models, and approaches
- a presentation of results (set of slides)
- broadened personal knowledge by the attendees
- an informal compilation of workshop notes
- this report

1.5 Approach of the Workshop

The problem area was divided into three subproblems, with a subgroup for each. The three sub-groups were

- design for upgrade
- commercial off-the-shelf (COTS) upgrade
- online upgrade

Most of the workshop time was devoted to working group sessions. The agenda for the meeting is shown in 0 Agenda.

The remainder of this report summarizes the results of the three subgroups, plus a summary discussion.

1.6 Contents of This Report

This report is a compendium of the discussions and outcomes of the workshop. It represents the collective perspectives of all of the attendees, not the perspective of any single individual. As editors, we coalesced the information into a coherent format. No attempt was made to provide a single interpretation of the information presented.

2. Design for Upgrade

This chapter presents the results of the efforts of the working group that focused on the issues associated with design for upgrade.

2.1 Objectives and Goals

It was conjectured that the cost effectiveness of upgrading systems can be improved if the issue of upgrade is considered during design and continues to be considered throughout the product life cycle.

To help stimulate and guide the discussion, the participants in this session were asked the following questions:

- How important is design for upgradability?
- What design approaches should be applied to help make upgrade more cost effective?
- Can the benefits of this approach be quantified to support management decisions?

2.2 Participants

The participants in this working group were

- Chair: Jack Goldberg (SRI International, retired)
- Mario Barbacci (SEI)
- Lynn Elliott (Guidant Medical Electronics)
- Dave Gluch (SEI)
- Connie Heitmeyer (Naval Research Laboratory)
- George Shoemaker (Naval Undersea Warfare Center)
- Bill Wood (SEI)

2.3 Issues and Findings

This section summarizes the issues and problem areas associated with designing a system for upgradability. In this summary, we grouped the issues that were identified and discussed into subsections. Within each subsection we grouped specific issues that were discussed into separate paragraphs.

2.3.1 General Issues

Currently system upgrade efforts are often difficult, expensive, and ad hoc, rather than being recognized as an important consideration in the development process or the product

concept. Although many systems are upgraded during their lifetime, design for upgradability is not usually a standard part of current software and system engineering practices.

There is a need to identify upgradability as a key feature of a system development effort and of the system itself— to generate an awareness for the need to upgrade. Upgradability must be addressed throughout the life cycle. It is integral to the system development process.

During upgrades, complex problems often arise that require substantial redesigns of parts of the system. In many cases, a change in one part of the system may necessitate changes in other components, just to maintain integrity of the system. For example, increasing the scale of some service may overload a limiting resource that is difficult to augment or replace. Such limitations are, in general, easier to overcome if they are anticipated during design and provisions are made in the design that would simplify upgrade actions.

Software cannot be considered in isolation from the system of which it is a part. A broad system approach must be taken that addresses software, hardware, and system-wide issues.

Broader management awareness of the need for and benefits of design for upgradability must be achieved to provide adequate funding. This involves conveying to management both the problems that can happen without considering upgradability (e.g., systems that are expensive to maintain and evolve), and the advantages of designing upgradability into a system (e.g., longer useful lifetime, lower life-cycle cost).

The general paradigm shift for upgradability in system and software design must address how to

- anticipate the nature and scale of environmental and behavioral changes over the system life cycle
- achieve high levels of orthogonality among subsystems so as to allow independent subsystem changes
- identify components requiring change to achieve desired new functionality
- assess changes in system complexity that might result from a proposed upgrade
- estimate practical limits to the upgradability of a design, to support change-or-replace decisions
- identify tradeoffs among upgrade objectives, such as increased scale, new functionality, and new non-functional services
- specify designs so as to simplify upgrade plans
- track dependencies among components and upgrades

2.3.2 Guidelines for Engineering for Upgrade

While there are notable technical challenges associated with designing for upgrade, simply incorporating many of the well-understood software engineering concepts into practice will

significantly improve the effectiveness of upgrades. However, these concepts, by themselves, will not solve the problem entirely. It is also important to integrate upgradability into the design and design process and establish a paradigm for designers that includes guidelines for "engineering choices" and tradeoffs in engineering upgrades. Specialized methods may be needed for managing complexity, evaluation, and tradeoff analysis. It may be possible to gain leverage from existing system capabilities to improve the quality of the upgrade (e.g., using existing fault-tolerance functions within a system).

Some aspects of software engineering approaches that may be required or be useful in designing for upgrade include considerations of modularity, composeability, scalability, and integrability of designs. A design for upgrade may include controllability, observability, and testability built into the artifact.

Issues that must be addressed in designing for upgrade include consideration of

- previous knowledge about earlier versions in the upgrade
- how to verify and validate designs
- how to assess the viability and feasibility of a potential upgrade
- human factors and human computer interaction (HCI)

Most "new" systems are updates to existing (legacy) systems rather than completely new systems. This has the effect of making design for upgrade more difficult as most such systems were built without any consideration of the need for future upgrades. In general the better documented and better structured such systems are, the easier it is to upgrade or transform them into new systems.

Designing a system for upgrade from the start can be facilitated with

- architectures that include ease of changes as a primary consideration
- automated tools for design and verification
- methods to identify and isolate the impact of an upgrade
- tools to make it possible to recognize the dependencies among system components
- cost models for defining anticipated changes and their impact

Commercial upgrade successes may be useful as a basis for effective upgrade design practices. Other considerations include design approaches for discardability, approaches for implementing upgrades, strategies based upon product lines, and the impact of an upgrade on operational procedures. This includes both real changes and implicit changes.

2.3.3 Classification System

To support the design for upgrade efforts, there is a need to classify systems. The exact classification appropriate to any particular problem may involve classification by industry, attribute, resources required, or life-cycle model. It is not clear exactly which approach would be most effective in all situations or any particular situation. Multiple classifications

may be needed to deal effectively with upgrade issues. One scheme may address issues across an entire industry (e.g., one technique may be appropriate for aircraft control but not for telecommunications) while other schemes may deal with systems based upon individual characteristics (e.g., one technique may be appropriate for single processors but not distributed systems).

2.3.4 System Issues

An upgrade decision is subject to a number of engineering and economic issues. Managers need guidelines and models to aid in the evaluation of upgrade plans. Important questions include the following:

- What is the minimum size of a justifiable upgrade; would it be better to build a new system?
- What is the feasibility of a proposed upgrade; what is the cost and the benefit, and is the required time consistent with other business objectives?
- Will technology developments make the upgrade obsolete when it is accomplished?
- How many future upgrades to the system will be practical, and what frequency may be expected?

Such questions can be included as part of a new design approach for system upgradability that is an integral part of system specification, design, and maintenance throughout a system's life cycle.

2.4 Summary of Key Outcomes

Designing for upgradability must be a key design criterion for all system designers; it should be a part of the design culture.

Designing for upgrade involves an extension of current design methodologies, but with special problems. There is a need for establishing appropriate (new) system engineering methods, constructive design techniques, architectures, and tools specifically aimed at improved system upgradability. Some of the open issues involve guidelines for systems developers and testing and verification strategies for upgrading systems.

While there is a general need for guidelines for design for upgrade, special needs and opportunities exist in various types of systems (e.g., control, distributed, networks, memory-intensive systems). These differences result in unique design requirements and constraints for effective system upgrade.

3. COTS Upgrade

3.1 Objectives and Goals

As COTS software becomes an increasingly significant part of systems, developers and customers are faced with the need to respond to upgrades in the COTS components. In general, these upgrades are driven by the suppliers—sometimes without regard to the impact on systems that use the components.

To help stimulate and guide the discussion, the COTS group was asked the following questions prior to the workshop:

- What is the impact of the problem on currently deployed software?
- What is the anticipated impact as more systems rely on COTS components?
- What are the specific techniques that can be used to alleviate the problems associated with supplier-driven COTS upgrade?
- What other issues should be identified?

3.2 Participants

The participants in this working group were

- Chair: Charlie Westerfield (Harris)
- Stephen Barnett (National Security Agency)
- Peter Feiler (SEI)
- Kathryn Kemp (National Aeronautics and Space Administration)
- Mike Lane (DERA, UK)
- Lui Sha (SEI)
- Kevin Sullivan (University of Virginia)
- Jeffrey Voas (Reliable Software Technologies)
- Chris Walter (WW Group)

3.3 Issues and Findings

3.3.1 Participants' Expectations

The participants in this subgroup came from diverse backgrounds and had diverse expectations. Specific topics of interest included the following:

- how to use COTS effectively in diverse system types including federated, mission-critical, safety-critical, and ultra-reliable systems

- how to ensure system security in the presence of COTS components as those COTS components evolve
- how to evolve COTS-based systems effectively
- tools for COTS development and testing, including package-oriented programming, fault-injection, analysis tools, etc.
- management issues surrounding the use and upgrading of COTS components

3.3.2 The Upgrade Cycle

It is important to recognize that, in general, the user has no control over the evolution and release cycle of COTS components. Thus it is important to understand the COTS vendors' product release schedule, the market, the technology, and the trend towards standards.

Some COTS components are treated like appliances by the user. They do not plan to upgrade the component, but replace it when it wears out. Other COTS components are upgraded only to fix problems or to meet new needs. Finally, some users continually upgrade their COTS components as new versions are introduced. This can be to add new functionality, to keep up with (or ahead of) the competition, or to take advantage of new hardware capabilities.

Whichever of the above upgrade strategies the customer uses, the changes from one version to the next may be anything from minor to major in scope as shown below:

- maintenance releases: These typically involve bug fixes with no new functionality added. They are usually backward compatible.
- minor upgrades: These are new releases of the system that add some functionality. They are usually backward compatible.
- technology refresh or major upgrades: This is a complete new version of a system with new functionality. It is typically backward compatible.
- technology insertion: This is a product swap out with new system-level functionality. Backward compatibility is not assured.

As you move down the list, the upgrade cost to the customer usually increases.

3.3.3 To Upgrade or Not to Upgrade?

When a vendor announces a new version of a COTS component, the user must decide whether to upgrade or continue to use the old version. If the upgrade fixes a problem that affects the user, or if it adds new functionality that is important to the application, the decision to upgrade may be obvious.

There is a tendency to want the latest version of anything. In the absence of overwhelming reasons to upgrade, this desire must be balanced against both the risk of upgrading and the cost of upgrading. There may be risks associated with both upgrading (e.g., the unknown reliability or security of the new component) and not upgrading (e.g., the old component is

aging and there is no longer any vendor support available). Costs include the obvious (purchasing the upgraded component), and the not so obvious (increased operational costs, retraining operators, cost of spares, cost of recertification). Without a proper risk/reward and cost/benefit analysis, it is impossible to determine whether it makes sense to upgrade the component.

One component of risk is the degree of uncertainty as to whether the upgrade will be successful. There are several ways to alleviate, at least partially, this uncertainty. Before deciding to upgrade, the following factors should be considered: the qualification process used by the vendor, the number of customers already using the upgraded component with success, the vendor's past performance and upgrade history, the logistical support that the vendor is willing to provide, and the severity of the upgrade (major versus bug fix).

3.3.4 Planning for Upgrade

To use COTS components effectively, the system must have been designed for change. This was the main topic of another subgroup. The key observation is that upgrading COTS components requires planning. Concepts such as layering, packaging, and information hiding can make upgrading easier. Even so, interfaces to the components will evolve over time. Frameworks in specific domains can accommodate this evolution. Different domains will have different sensitivity to change and upgrade.

Not all upgrades will be successful, so there must be a plan to deal with the failure of the COTS upgrade. The system will either have to be repaired or returned to its pre-upgrade configuration.

Modifying the development and validation process to require that the effects of an upgrade (what parts of the system the upgrade will affect and how it will do so) be known in advance should lead to less complex designs.

3.3.5 Testing Upgrades

No software is perfect, but some is better than others. Before deciding to use an upgraded COTS component, there should be some proof that the component works and will not cause a failure in a critical function. The COTS component must have no more than an acceptable level of faults, and the larger system must be able to deal with those faults. Dealing with the faults may involve simply tolerating them or rolling back to a previous version of the component.

In certain environments (e.g., safety critical, secure), upgrading the COTS component will require recertifying the system against some criterion. However, most COTS components must be treated as black boxes, since their manufacturer does not make the internal structure visible. This makes certification and even testing of upgrades difficult.

3.3.6 Architectural Implications

The assumption that complete system retest and validation is impractical, especially in the face of COTS components, which are typically black (or at least gray) boxes, implies that evolving most systems will require that they be designed for change. Designing a system for change requires

- facilitation of change through partitioning, modularization, dynamic binding, and dynamic configuration
- scoping of change through encapsulation, documentation, simplifying and semantically enriching the interface, and simplification of connectivity
- reducing the impact of change through the ability to perform analysis at the architectural/system level and of the system itself

This leads to a "specification sheet" approach for components that provides functional and non-functional properties, architectural descriptions, and configurations as a basis for impact analysis. Some questions to be answered include the following: What are the relevant facts that should go into a specification sheet? What aspects need to be validated? Do the facts have a linearity property (i.e., checking of endpoints vs. discrete state space)?

The analysis will have to be performed with imperfect and incomplete information. This leads to

- analysis of the architecture based on specification information, including propagation and the impact of change to a component specification
- analysis and validation of the implementation of the component against the specification

Off-line impact analysis and validation [e.g., simulation, dependency analysis, supplier (release) testing, and consumer (acceptance) testing] may be incomplete. The result is that we need to be prepared to deal with a component violating its specification online through detection, containment, and mitigation.

Detection mechanisms include runtime monitoring, watchdog timers, and assertion testing (i.e., built-in test) for components and for the users of the components (since it is also useful to protect against misuse of a component.) Sometimes we will be able to detect a problem through direct observation of the fault or violation. Other times we will only be able to infer the existence of a fault condition by observing the effects.

Containment mechanisms include firewalls, wrappers, timing enforcement, and the testing and validation of firewalls. Firewalls can be in the form of runtime mechanisms or in the form of "safe" languages (i.e., language concepts enforced by compilers and their runtime system). Incomplete firewalling is also useful. For instance non-critical components might be outside of the firewall, or the firewall may only be in place with respect to properties deemed important, (e.g., security or reliability).

Mitigation mechanisms include forward and backward recovery at runtime and rollback to the configuration in operation before the upgrade.

Thus, in addition to the analysis of components against their specifications, and the impact of changes in those specifications, we also must analyze or validate a fail-safe minimal application core and the detection-containment-migration infrastructure. From an architectural perspective, this leads to a strategy that tolerates faults generated by COTS upgrades resulting in "COTS-assisted operational capabilities."

3.3.7 Recertifying and Revalidating Upgrades

We take as given the current certification process. Given this, and that it is burdensome, we pose the following question: Is there a way of limiting the scope of recertification/validation to something less than the entire system?

A problem is that we need to verify all changes and side effects of those changes—not only the advertised changes, but also unannounced changes or even defects that may have been introduced. The supplier presumably does white-box testing on the component, presumably following the supplier's own process. The customer presumably is only able to do black-box testing.

The impact of recertification can be reduced by reducing the connectivity of the component. If all effects are funneled through a small number of connections, they can be more easily tested.

Another technique that can sometimes be employed is hardware-based partitioning. This is a common practice in high-reliability contexts (e.g., see DO-178B), but it is too costly in some other contexts. Hardware sharing and the soundness of shared resources becomes an issue.

Other approaches that might hold some promise include scoping changes in terms of the product, system, and operation environment and statistical approaches for determining which components need to be tested. It may also be possible to scale recertification by using a combination of techniques and move towards incremental approaches to keep the effort in line with change.

3.4 Summary of Key Outcomes

There are risks associated with upgrading COTS components. There are also risks associated with not upgrading when a new version is released. Proper risk/reward and cost/benefit analyses can provide a logical basis for the decision whether or not to upgrade.

Upgrading COTS components requires planning. Program structuring techniques can make this easier by minimizing the impact of a change, but planning will always be necessary.

In a similar vein, the impact of recertification can be reduced by reducing the connectivity of the component. If all effects are funneled through a small number of connections, they can be more easily tested.

4. Online Upgrade

4.1 Objectives and Goals

Some systems provide critical services to their users and cannot be shut down for purposes of software upgrade. The usual solution to this problem is not to upgrade unless major system maintenance is required. Often the result is that such software becomes obsolete.

To help stimulate and guide the discussion, the online upgrade group was asked the following questions before the workshop:

- How pervasive is this problem?
- What are some examples?
- What technologies are appropriate to solving these problems?
- What additional technologies are needed?
- What other issues should be identified?

4.2 Participants

The participants in this working group were

- Chair: Walter Heimerdinger (Honeywell)
- Felix Bachmann (Robert Bosch)
- Mike Gagliardi (SEI)
- Mike Hinchey (21st Century Systems)
- Mark Klein (SEI)
- Marc Pitarys (Wright Laboratory)
- Sampath Rangarajan (Lucent Technologies)
- Therese Smith (MIT/Lincoln Laboratory)
- Dolores Wallace (National Institute of Standards and Technology)

4.3 Issues and Findings

Successful online system upgrade requires several steps including preparation, resource identification, selection of upgrade techniques, certification or recertification of the upgrade, a plan for rollback or recovery from a botched upgrade, and ideally a criteria for accepting the upgrade.

4.3.1 Preparing for the Upgrade

Preparation for an online upgrade requires that the customer and user concerns be identified and dealt with. There is no point to doing the upgrade, online or otherwise, if it meets no pressing needs. It is ultimately the customer who decides what the criteria are for upgrade acceptance.

Since this is an online upgrade, preparation will also include training the user and operator on how to use the changed system. For some systems, this will not require any preparation as the changes will all be behind the scenes. For other systems, this will require extensive preparation since the way that the system operates will be affected.

An online upgrade potentially requires additional resources over more traditional upgrade approaches. It must be possible to support both the old and new versions of the system simultaneously. This may require additional memory, disk space, computer cycles, etc. Being able to support multiple versions of the system places additional requirements on the architecture and runtime system mechanisms as well.

One way to determine if there are enough resources to support online upgrade is through the use of tools. For instance online measurement analysis tools can characterize the resource utilization and timing characteristics of the existing application and can also be used to characterize the new version of the application in the test environment. Other types of tools that may be of interest include program-flow tracing tools to analyze program structure, and program-slicing tools to isolate program sections for modification.

4.3.2 Techniques for Online Upgrades

Online upgrade requires its own infrastructure. The nature of this infrastructure will vary with the implementation of the application to be upgraded but should include the use of authenticated upgrade agents. The purpose of these agents is to instantiate a replacement upgrade unit, transfer state and data from the old version to the new version, enable the outputs for the upgraded version, and disable and ultimately remove the old version.

Depending on the implementation of the application, there may be different problems to be solved when performing an online upgrade. Object-oriented systems present their own set of problems. Specifically there is a need to be able to activate and deactivate threads within objects, to allow multiple versions (as well as multiple instances) of the same object to co-exist, to deal with transactions during replacement, and to re-route object requests/messages after an upgrade.

4.3.3 Tools for Online Upgrade

Specialized tools can make the online upgrade process better. Examples of tools that would be useful to any online upgrade effort would include tools to aid in certification and

recertification of the software. Regression testing tools can be used to ensure that the upgrade retains important functionality. Incremental testing techniques can be used to limit the scope of testing, but this requires knowledge of which portions of the software are affected by the change. This points to the need for tools to analyze the scope of an upgrade and rules for using them.

4.3.4 Recovery / Fallback from Unsuccessful Upgrades

Upgrades can fail. There must be a way to back-out an unsuccessful upgrade. For online upgrades, the challenge is to do this without causing the system to crash. In addition to support for runtime detection of failures and containment of faults, this requires tools and processes to map the updated system state and data back to something that the original version is able to deal with. Techniques such as checkpoint/rollback and tools that do reverse mapping should prove useful here.

4.3.5 Commitment to Upgrades

At some point, trust is developed in the upgraded system. This can happen over the course of time as experience is gained in the system, or it can be helped along through the use of acceptance tests and stress tests (e.g., fault injection). Once the new version is trusted, it may be possible to remove the old version and to free up the resources it consumes. Fallback ceases to be a viable option.

4.3.6 Other Upgrade Issues

The online upgrade group identified other issues worthy of study. These included semantic dependencies (when to switch and what to switch), upgrades within a single address space (e.g., no new process), configuration, and understanding what the old and new versions do and how they do it.

4.4 Summary of Key Outcomes

Online upgrade requires additional resources (e.g., CPU [central processing unit] cycles, memory, networking bandwidth) over more traditional upgrade approaches. Because of this, its usefulness is restricted to those applications that cannot afford downtime—for dependability, safety, or economical reasons.

Especially because online upgrade techniques are used in dependable and safety-critical environments, close attention must be paid to details of fault containment and upgrade rollback. Special attention must be paid to verifying and testing the proposed upgrade, or at least the recovery mechanism, before letting it go live.

Many of the points made throughout this section apply to the upgrade problem in general.

5. Discussion

Dependable system upgrade is a problem faced by a variety of domains, ranging from embedded medical electronics to real-time command, control, communication, and intelligence (C3I) systems. Although the problem is pervasive, the recognition that it requires special attention is not. This is evidenced by the fact that most systems are not designed with the need for upgrade in mind.

Improvement in the quality of upgrades can be realized by integrating upgrade considerations into the design process–instituting a culture of design for upgrade. Approaches for designing for upgrade approaches can be built upon established practices and technologies for system and software engineering, but with specialized adaptations to address upgrade capabilities. These adaptations may include specialized system engineering methods, constructive design techniques, architectures, and tools specifically aimed at improved system upgradability.

The current push to use COTS in an increasing number of system designs is aggravating the upgrade problem. Users of COTS components are often confronted with the reality that a vendor can dictate both the nature and the timing of upgrades, while allowing users or customers to provide input into upgrade decisions. Yet changes in newer versions of a COTS component may be difficult to integrate into an operational system and may have adverse affects. On the other hand, if a customer does not integrate an upgraded version of a component into their system, a vendor may either stop supporting earlier versions or require a costly customer support contract. Vendors of COTS components and their customers need to work together to ensure that cost-effective customer upgrade paths exist and are simple to implement.

Particular problems exist for mission-critical systems with high-dependability requirements where online upgrade may be advantageous. Additional resources are required for online upgrade and special attention must be paid to the details of fault containment and rollback. In this case, it is especially important to verify and test the proposed upgrade, or at least the recovery mechanism, before enabling the execution of the upgrade.

There is an important technology transition component to the upgrade problem. While technologies exist to ease the upgrade problem, designers are often not aware of them. Further, even when techniques for designing systems for upgrade are known, designers are often not trained in their use, made aware of their need and effectiveness, or encouraged to use them.

Dependable system upgrade is a ubiquitous problem characterized by numerous challenges and demanding requirements. All phases of a system's evolution are affected, including system concept, design, verification, implementation, testing, and operation. Comprehensive solutions to the problems associated with dependable upgrades will involve bringing together and enhancing a variety of system and software engineering disciplines.

Efforts to address these problems will involve identifying and, as appropriate, developing innovative approaches and integrating these approaches into routine systems and software management and engineering practice.

The Software Engineering Institute is actively working on the problems associated with dependable system upgrade. As part of the Dependable System Upgrade initiative, there are efforts investigating technologies and practices for online real-time upgrade (the Simplex architecture), continuous verification and test for upgrades, and specific awareness and technology transition issues associated with dependably upgrading systems.

References

[Ariane 96] *ARIANE 5: Flight 501 Failure* [online]. Available WWW <URL: http://sspg1.bnsc.rl.ac.uk/Share/ISTP/ariane5rep.htm> (19 July 1996).

[AOL 96] "America Off-Line." *San Francisco Chronicle* (8 Aug 1996): p. A1.

[DO 178B] *Software Considerations in Airborne Systems and Equipment Certification* (Document No. RCTA/DO-178B). Washington DC: RTCA, Inc., December 1, 1992.

[Leung 90] Leung, H. K. N. & White, L. J. "A Study of Integration Testing and Software Regression Testing," 290-300. *Proceedings of the Conference on Software Maintenance*, 1990. Los Alamitos, CA: IEEE Computer Society. Nov. 1990.

[Ostrand 88] Ostrand, T. J. & Weyuker, E. J. "Using Data Flow Analysis for Regression Testing," 233-247. *Proceedings of the Sixth Annual Pacific Northwest Software Quality Conference*, 19-20 Sept., 1988. Portland, OR: Lawrence & Craig, 1988.

[Purtilo 91] Purtilo, James M. & Hofmeister, Christine R. "Dynamic Reconfiguration of Distributed Programs," 560-571. *Proceedings of the 11th International Conference on Distributed Computing Systems*, Arlington, Texas. Los Alamitos, CA: IEEE Computer Society, 1991.

Appendix A: Framework

Introduction

Purpose

This document is a seed position paper that provides an overview of our current thoughts on the practice of upgrading software-dependent systems.

Synonyms for Upgrade

The term system upgrade is used here to encompass any change made to software that modifies its capabilities or attributes. The term upgrade often overlaps similar terms. In various contexts upgrade is also referred to as

- maintenance
- reconfiguration
- enhancement

While in certain contexts these terms refer to upgrade, in many cases they denote very different ideas than upgrade. For instance reconfiguration encompasses operational aspects, like mode changes in flight control systems.

System Upgrade

A system upgrade as used here encompasses changes in the [Purtilo 91]

- functional characteristics of individual components
- logical structure of the system (interrelationship of components)
- allocation of functions among system resources

Dependable Upgrade

The discipline of dependable software upgrade (DSU) melds three areas of software technology:

- dependability
- maintenance
- reconfiguration

No specific development or life cycle model is assumed and the issues relating to dependable software upgrade extend across the entire product life cycle.

Practice of Dependable System Upgrade

Dependable system upgrade practices encompass engineering processes, methods, tools, and technologies for the support of dependable system upgrade. Collectively, these are used to develop and support software systems that can be readily modified and improved.

Problem Space

Critical Systems

Organizations rely increasingly on software-dependent systems as critical components of their operation and as a vital part of their infrastructure.

Upgrade Issues

- Long System Life: These systems generally evolve over very long life times.
- Periodic Upgrades: Rapid evolution in software technology coupled with frequent changes in the operational environment not controlled by the organization require systems to be upgraded periodically to avoid obsolescence.
- Need for Improvement: Current upgrade approaches are often difficult, costly, and ad-hoc and lack tolerance to faults that may have been introduced by the upgrade.

Reliability Constraints

Dependable upgrades to deployed systems can be viewed as upgrades with reliability constraints.

Reliability constraints can range from the basic need to have a quality (reliable) upgrade to a requirement for safe uninterrupted service of an online system, even in the event of an error in the upgraded software, hardware, or process.

Implementation

The implementation of an upgrade can be either

- static (take the system off-line during reconfiguration) or
- dynamic (make changes on-line while the system is operating)

Views of Dependable System Upgrade

Life Cycle View

The dimensions of the problem span the entire product life-cycle and can be divided into practices and technologies.

Practice

The practices of dependable system upgrade encompass principles, process, methods, and tools that addresses the broad set of

- Management
- Business and Economic
- Design and Development (Engineering)

issues relating to the development, operation, and maintenance of software-dependent systems.

Domains

Some of the domains to be addressed include:

- real-time control
- transaction processing
- C^3I
- Telecommunications
- database
- manufacturing

Scale

Upgrades may differ in scale. We hypothesize that the smaller the upgrade the less difficult it is to upgrade dependably. A measure of scale might include:

- Minor (small number of lines of code, usually confined to one module.)
- Modular (replacement/modification of a whole module.)
- Subsystem (replacement/modification of a subsystem.)
- Version Change (major modifications to the complete system.)
- Replacement (completely new system with similar intended functionality).

These categories differ in the amount of the system which is "touched" by the upgrade process, ranging from minuscule (<.01%) to total (100%).

Motivation

The primary goal of upgrading a system is to improve it. The improvement may result in a system that:

- Operates more correctly. (e.g., a bug is fixed)
- Operates more efficiently. (e.g., performance is improved, it is cheaper to deploy, it computes more precise answers, etc.)
- Has additional capabilities. (e.g., the system has been enhanced.)

Upgrade Paradigm

There are several strategies for accomplishing an upgrade

- Direct Transfer: Turn off the old version, turn on the new one.
- Parallel Operation: Run the old one and new one in parallel, switching to the new one only after experience dictates it will be safe.
- Legacy Backup: Running the old and new versions in parallel, using the new version as long as it continues to work properly and reverting to the old version when the newer version fails.

For software installed in multiple sites working together, another set of strategies dictates how to accomplish the upgrade of those sites.

- Total System: upgrade all sites simultaneously.
- Phased: upgrade sites one at a time, making sure that everything is working before moving on to the next site.

The former is necessary when the sites inter-operate and the new version cannot work with the old version on different sites. The latter can work if the old version and new version inter-operate with each other.

COTS

Introducing COTS software into a system further complicates the dependable software upgrade process.

- Often the COTS component must be treated as a black box. There is no visibility into its structure or internal behavior. Successful integration of the COTS software is dependent on the manufacturer following documented specifications. If the specifications change in an undocumented manner interfaces may fail unexpectedly or the system may demonstrate unanticipated behavior.
- The vendor of the COTS system usually determines when and how to release upgrades. Completion of a system upgrade depends on the vendor's timely completion and release of their product. Consequently, upgrade schedules may be dictated and/or adversely effected by the vendor. The system upgrade schedule must be coordinated with the vendor's.
- The vendor determines which versions to continue supporting. A system using COTS components that is not upgraded based upon the vendor's product support and maintenance policies may contain unsupported (obsolete) components.

Other Thoughts on DSU Issues

Although our tendency is to focus on systems where high dependability as a requirement, it is also important to look at the problem in the broader sense of overall system quality.

Perhaps this broader focus can be described as identifying the practices and technologies that can provide quality upgrades that are cost-effective, reliable, and efficient.

These efforts should address the goal of defining a broad practice of dependable system upgrade and integrating that practice into software and systems engineering

Appendix B: Workshop Participants

Felix H. Bachmann
Robert Bosch GmbH
Kleyerstr. 94
Frankfurt, Germany 60326
Phone: 412-268-6194
Fax: 412-268-5758
E-Mail: fb@sei.cmu.edu

Mario Barbacci
Software Engineering Institute
4500 Fifth Ave
Pittsburgh, PA 15213-3890
Phone: 412-268-7704
Fax: 412-268-5758
E-Mail: mrb@sei.cmu.edu

Stephen Barnett
National Computer Security Center
10209 Bristol Channel
Ellicott City, MD 21042
Phone: 410-859-4371
Fax: 410-859-4375
E-Mail: sbarnett@romulus.ncsc.mil

Jon Dehn
Lockheed Martin
9231 Corporate Blvd.
Rockville, MD 20850
Phone: 301-640-2912
Fax: 301-640-3103
E-Mail: Jon.Dehn@lmco.com

Lynn Elliott
Guidant Medical Electronics
4100 Hamlin Ave N
St Paul, MN 55112
Phone: 612-582-2842
Fax: 612-582-7484
E-Mail: lynn.elliott@guidant.com

Peter Feiler
Software Engineering Institute
4500 Fifth Ave
Pittsburgh, PA 15213-3890
Phone: 412-268-7790
Fax: 412-268-5758
E-Mail: phf@sei.cmu.edu

Mike Gagliardi
Software Engineering Institute
4500 Fifth Ave
Pittsburgh, PA 15213-3890
Phone: 412-268-7738
Fax: 412-268-5758
E-Mail: mjg@sei.cmu.edu

Dave Gluch
Software Engineering Institute
4500 Fifth Ave
Pittsburgh, PA 15213-3890
Phone: 412-268-7069
Fax: 412-268-5758
E-Mail: dpg@sei.cmu.edu

Jack Goldberg
(Independent Consultant)
3373 Cowper Street
Palo Alto, CA 94306
Phone: 415-493-3390
Fax: 415-493-3758
E-Mail: goldberg@csl.sri.com

Walter Heimerdinger
Honeywell Technology Center MN65-2200
3660 Technology Drive
Minneapolis, MN 55418
Phone: 612-951-7333
Fax: 612-951-7438
E-Mail: walt@src.honeywell.com

Constance Heitmeyer
Naval Research Laboratories
Code 5546
Washington, DC 20375
Phone: 202-767-3596
Fax: 202-767-9197
E-Mail:
constance.l.heitmeyer@nrl.navy.mil

Mike Hinchey
21st Century Systems, Inc.
420 Hardscrabble Road
Chappaqua, NY 10514-3030
Phone: 201-596-5750
Fax: 201-596-5777
E-Mail: hinchey@homer.njit.edu

Kathryn Kemp
NASA SW IV&V Facility
100 University Drive
Fairmont, WV 26554
Phone: 304-367-8238
E-Mail: kemp@ivv.nasa.gov

Mark Klein
Software Engineering Institute
4500 Fifth Ave
Pittsburgh, PA 15213-3890
Phone: 412-268-7615
Fax: 412-268-5758
E-Mail: mk@sei.cmu.edu

Mike Lane
DERA (UK)
Probert Building
Farnborough, Hampshire
England GU14 OLX
E-Mail: mjlane@dra.hmg.gb

Marc Pitarys
Wright Laboratory
WL/AASH, Bldg. 620
Wright-Patterson AFB, OH 45387-6543
Phone: 937 255-6548
Fax: 937 656-4277
E-Mail: pitarymj@aa.wpafb.af.mil

Sampath Rangarajan
Lucent Technologies Bell Laboratories
Room 2A-229
600 Mountain Avenue
Murray Hill, NJ 07974
Phone: 908 582-6687
E-Mail: sampath@research.bell-labs.com

Lui Sha
Software Engineering Institute
4500 Fifth Ave
Pittsburgh, PA 15213-3890
Phone: 412-268-5875
Fax: 412-268-5758
E-Mail: lrs@sei.cmu.edu

George Shoemaker
Naval Undersea Warfare Center
Code 3822, Bldg. 104
1176 Howell Street
Newport, RI 02841-1708
Phone: 412-268-3420
Fax: 412-268-5758
E-Mail: gts@sei.cmu.edu

Therese Smith
Lincoln Laboratory
244 Wood Street
Lexington, MA 02173
Phone: 617-981-4179
Fax: 617-981-3220
E-Mail: tmsmith@ll.mit.edu

Kevin Sullivan
University of Virginia
School of Engineering and Applied Science
222 Olsson Hall
Charlottesville, VA 22903-2442
E-Mail: sullivan@cs.virginia.edu

Jeffrey Voas
Reliable Software Technologies Corporation
21515 Ridgetop Circle, Suite 250
Sterling, VA 20166
Phone: 703-404-9293
Fax: 703-404-9295
E-Mail: jmvoas@rstcorp.com

Dolores Wallace
National Institute of Standards and Technology
NIST North, Bldg 820, RM 517
Gaithersburg, MD 20899
Phone: 301-975-3340
Fax: 301-926-3696
E-Mail: wallace@nist.gov

Chris Walter
WW Group
4519 Mustering Drum
Ellicott City, MD 21042
Phone: 410-418-4353
Fax: 410-418-4355
E-Mail: cwalter@blaze.cs.jhu.edu

Chuck Weinstock
Software Engineering Institute
4500 Fifth Ave
Pittsburgh, PA 15213-3890
Phone: 412-268-7719
Fax: 412-268-5758
E-Mail: weinstock@sei.cmu.edu

Charles Westerfield
Harris ISD
P.O. Box 98000
MS W3-7755
Melbourne, FL 32902
Phone: 407-984-6281
Fax: 407-984-6323
E-Mail: cwesterf@harris.com

Bill Wood
Software Engineering Institute
4500 Fifth Ave.
Pittsburgh, PA 15213-3890
Phone: 412-268-7723
Fax: 412-268-5758
E-Mail: wgw@sei.cmu.edu

Appendix C: Agenda

Wednesday, April 16

8:00 - 8:30	Registration
8:30 - 9:15	Plenary
9:15 - 12:00	Working Groups
1:00 - 1:30	Plenary
1:30 - 5:00	Working Groups

Thursday, April 17

8:00 - 10:00	Working Groups
10:15 - 12:00	SEI demos/presentations
	Group chairs prepare slides/reports
1:00 - 3:00	Presentations and discussion of results

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (LEAVE BLANK)	**2.** REPORT DATE **August 1997**	**3.** REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE **Workshop on the State of the Practice in Dependably Upgrading Critical Systems**		**5.** FUNDING NUMBERS C — F19628-95-C-0003
6. AUTHOR(S) **David P. Gluch and Charles B. Weinstock (Editors)**		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		**8.** PERFORMING ORGANIZATION REPORT NUMBER **CMU/SEI-97-SR-014**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/AXS 5 Eglin Street Hanscom AFB, MA 01731-2116		**10.** SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12.A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		**12.B** DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

This report describes the results of the Workshop on Dependably Upgrading Critical Systems held April 16-17, 1997 at the Software Engineering Institute. The workshop addressed a broad spectrum of issues associated with dependably and cost-effectively upgrading systems, primarily those with reliability or real-time requirements.

14. SUBJECT TERMS: **commercial off-the-shelf (COTS) components, critical systems, dependable systems, design for upgrade, online upgrade, system upgrade**			**15.** NUMBER OF PAGES **40**
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	**18.** SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	**19.** SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	**20.** LIMITATION OF ABSTRACT UL