Stop the Flow
A Damage Mitigation Framework for Trusted Systems

Linda M. Laird, Stevens Institute of Technology
Jon P. Wade, Stevens Institute of Technology

Abstract. This article proposes a very high-level, abstract framework of Damage Mitigation to enable the architecture and design of trusted systems, those that dependably perform a mission while minimizing or eliminating the probability of significant, unintended damage. This framework is based upon the premise of system imperfection, consisting of a Trusted Systems Model and a Damage Process Model. The intent is that this systems approach to Damage Mitigation will improve the ability to analyze the Damage Mitigation capabilities of a system and encourage new solutions.

Introduction
A Trusted System is one that dependably performs its mission while minimizing or eliminating the probability of significant, unintended damage. The ability to develop, deploy, and maintain trusted systems, those that are safe, secure, dependable, and survivable, is an unsolved problem. There are calls for papers looking for new approaches to address these issues [1] [2] as radiation systems continue to occasionally kill people [3], cars occasionally refuse to stop, and rogue botnets are available for hire [4]. If anything, our experiences in the last 30 years of building software intensive systems have shown that software and systems without defects or vulnerabilities are, for all practical purposes, non-existent. This is not to say that the utmost should not be done to eliminate defects and vulnerabilities. It must. But time and effort are limited, as are human abilities and knowledge, while system complexities continue to increase. We must assume that defects and vulnerabilities will always exist.

This article proposes a very high-level, abstract framework of Damage Mitigation based upon this premise of imperfection. Damage will be defined as any significant negative consequence of a system’s operation. The intent is that this systems approach to Damage Mitigation will facilitate new ideas on how to improve the fundamental properties of trustability of systems and encourage the creation of trustable architectures and designs for critical systems. The central idea is that there exists a causal event chain that can lead to damage and a loss in system value. At each point within the chain, there are potential “chokepoints” where it may be possible to stop the flow from an instigating event to a damage event.

The broader framework and the different viewpoint of this article hopefully will encourage and suggest new solutions to the problem of building, engineering, and operating systems that need to be trusted. Alan Kay once remarked, “A change in perspective is worth 80 IQ points.” The experience to date has been that in using this framework, additional solutions and mitigation strategies to past accidents became apparent.

System Trustability Model
The Trustability Model shown in Figure 1 has three attributes: Trustability Properties, Threats to Trustability, and Means to Achieve Trustability. It is based upon the Dependability Model in [5]. This model shows that there is a strong relationship between the various properties of trustability, the threats to those properties, and the means to achieve those properties. For example, Threat Prevention has a positive impact on all of the properties of trustability, with perhaps the exception of resilience. Threat Tolerance has a positive impact on all of the properties. Being able to tolerate and eventually remove loss events improves resilience, availability, and security.

In this model, safety reflects the system’s ability to protect its users and environment from both physical and non-physical threats. Security describes the ability of the system to protect itself with respect to the confidentiality of its assets and its overall integrity. Dependability ensures that the system provides its services and supports its mission when it is required. The properties of availability, reliability, and maintainability are all elements which constitute dependability. Finally, survivability is a measure of the system’s capability to support the attributes noted above despite adverse environmental effects. The properties of robustness and resilience contribute to this capability.
System safety, security, dependability and survivability are proposed as the critical properties of trustability. The relative importance and need for each individual property differ by domain, application, and context of the system. The definitions of these terms and those for some of their constituent properties that will be used in this work are described below.

Safety is the ability of a system to perform predictably under normal and abnormal conditions, thereby preventing accidental injury, death, or damage. This definition has been adapted [6] with a key change being the addition of the damage concept. Although safety, in general use, typically has the connotation of physical safety, with software intensive systems, safety is the antithesis of dangerous and can relate to non-physical safety as well.

Security is the ability of a system to thwart attempts to exploit defects and vulnerabilities to disrupt the system or the value it produces. (Adapted from [7]) Security adds the concepts of perpetrators and malicious attackers.

Dependability is “the ability to deliver service that can justifiably be trusted.” [5] The dependability of a system is based upon its reliability, availability, and maintainability. The relative importance for each of these is determined by the context in which the system is being used.

Survivability is the ability of a system to function during and after a natural or man-made disturbance. (Adapted from [8]) For a given application, survivability must be qualified by specifying the range of conditions which the system will survive along with the minimum acceptable levels of safety, security, and dependability. Resilience and robustness are two important properties which determine survivability.

**Threats to Trusted Systems**

A Threat to a Trusted System is anything that can potentially cause the system to have significant unintended damage, including not being able to complete its mission successfully. This definition includes both malicious and unintended threats. In this model, the set of threats includes actors, faults, failures, hazards, and loss events.

The term Actor is borrowed from Unified Modeling Language (UML) and well suited to this use. Within this context, an Actor is anything that instigates execution of the system. Actors include humans, systems (external and internal), the physical world, and any other external object that can act upon the system. Actors instigate execution of the system, and in doing so provide inputs, friendly and malicious, planned and unexpected, that can cause failures, hazards, and ultimately, significant damage.

A fault is a defect or vulnerability in the system that may or may not cause a failure. A fault might be a memory leak that can lead to a system crash or incorrect execution. An exploitable buffer-overflow vulnerability is a fault. A requirements defect is a fault. For software, if the code that contains the faults is never executed, or never executed under the precise conditions that cause the fault to occur, then the system never fails due to this fault. Faults are defects in the system that may or may never be seen in operation.

A failure occurs when the object (component, human, system) can no longer create value (carry out its mission) or no longer delivers the correct service. Failures only occur during system execution.

A hazard is a state or set of conditions of a system that, together with other conditions in the environment of the system, will probabilistically lead to a loss event, be it an accident or incident. A hazard represents the possibility of a loss event. Hazards have the following attributes: severity, likelihood of occurrence, exposure (duration), and likelihood of a hazard leading to an accident.

A loss event, within the context of this work, is an accident, incident, or other unsuccessful completion of the mission of the system. Loss events vary in significance. An incident is considered to be a loss event that involves minor or no loss but with the potential for significant loss under different circumstances. A loss event can be mitigated to minimize damage.

**Means to Achieve Trusted Systems**

Trustability is achieved when loss events either do not occur or the unintended damage caused by them to the stakeholders is deemed to be acceptable. In order to achieve trustability, the threats to trustability need to be mitigated. There are a significant number of mitigation techniques categorized below. While there have been numerous attempts at this categorization, this work will use the following definitions:

1. **Threat Prevention:** the set of techniques to assure that the threat is not allowed into the system.
2. **Threat Removal:** the set of techniques to remove threats from the system.
3. **Threat Tolerance:** the set of techniques that prevent a threat from causing damage.
4. **Threat Management:** the set of techniques to minimize the potential damage.
5. **Threat Detection:** the set of techniques that allow threats to be observed. Threat detection could be included as an enabler to all of the other Means to Achieve. In this work, it is noted separately as it is required to demonstrate that the system is trustworthy. Monitoring allows the current threat patterns to be understood and potentially supports the prediction of future threat patterns.

All of these techniques have advantages, although threat prevention is obviously preferred. If malicious actors never contact the system, or vulnerabilities never exist, the damage that they might cause is totally mitigated. When threats cannot be prevented or removed, threat tolerance, which stops the propagation of potential damage, is useful. Threat tolerance includes basic techniques such as rollback and recovery. Threat management can be used once a threat condition is recognized and until the root cause can be addressed. For example, quarantining off parts of a system that have been compromised is a threat management technique.

**Damage Process Model**

A casual model of the Damage Process is shown in Figure 2, consistent with the definitions above. It is a set of steps, starting with an actor, and terminating with damage. If a fault is encountered, and it causes a failure, it can create a hazardous situation that under certain circumstances can cause a loss event, which, if not properly mitigated can cause unacceptable damage. It is important to emphasize that this damage process is one
RUGGED SOFTWARE

![Diagram of Damage Process Steps](Figure 2: Damage Process Steps)

scenario of a system-in-the-large in execution. No damage occurs unless the process is initiated. In some respects, it could be considered a high-level scenario or use case.

As an example, consider the St. Vincent’s radiation tragedy [3], in which at least two patients were killed by incorrectly applied radiation dosage, apparently due, in part, to defects in the fault management software and system design. “... as (the technician) was trying to save her work, the computer began seizing up, displaying an error message. An error message asked (the technician) if she wanted to save her changes before the program aborted. She answered yes. ... (the Doctor) approved the new plan.” Two rounds of radiation treatment were given. The technician ran a test to verify the settings before the third round. “What she saw was horrifying: the multileaf collimator, which was supposed to focus the beam precisely on his tumor, was wide open.” [3]

The patient subsequently died from the incorrect and excessive radiation. The equipment provider subsequently “distributed new software, with a fail-safe provision.” [3]

Each step leading up to the final damage result is considered a potential threat to the trustability of the system. At each step, there is the possibility of mitigating the threat to prevent or minimize the final damage. There are possible intervention points, or chokepoints, after each step of the process model, as shown below in Figure 3.

At each chokepoint, techniques could be applied to both monitor and mitigate potential threat conditions. Using this radiation example, the flow to the significant damage event could have been stopped if:

- a. The technician did not save her work when the system was crashing. Additional training or documentation may have worked.
- b. The system had improved fault handling that either stored the work correctly or did not allow potentially compromised configurations to be stored.
- c. The display of the information to the doctor, who had to sign off on it, may have been complex and difficult to verify. A visual display/simulation may have improved it significantly.
- d. The technician was required to run a test to verify the settings before first use (which she did before the third use).
- e. The system had safety checks (either internal to the algorithms or even on the radiation devices) that prevented extremely high dosage radiation or forced the operator to repeatedly confirm that this high level was intended.
- f. The overall system was designed to first test the radiation settings on a radiation “dummy,” with safe dosage confirmation before continuing with a live patient.
- g. There were new medicines that could reverse the effects of over-radiation.

This is an example of different solutions that are relatively easy to uncover by considering the damage event as a process with many possible intervention points, rather than identifying the problem as a bug in the code. These solutions are examples of possible “safe(r) designs.” From a systems viewpoint, solution f is especially interesting because it stops incorrect radiation in almost all cases, no matter the cause. It would “Stop the Flow” before any loss event could occur.

### Discussion of Mitigation Techniques

It is important to realize that the mitigation techniques of focus here are for the execution process, not the development process. Preventing faults and vulnerabilities from entering the system during development is critically important and is a relatively well developed field, but in this model, we assume that regardless of the quality of the developers and the development process, faults and vulnerabilities will exist.

To illustrate the ease of uncovering mitigation techniques, examples of generic techniques for each chokepoint are listed. These generic techniques could be easily expanded and used as a brainstorming tool or checklist during the design of critical systems.

The first chokepoint focuses on actors and fault prevention. Some possible issues might arise from unauthorized users, incorrect usage, or system overload. Example mitigation techniques include denying access to unauthorized users of the system, denying access to “untrusted” external systems, deliberate shedding of users in overload conditions, and input and interface validation.

At the second chokepoint, typical examples would be traditional
techniques such as robust fault handling, fault tolerance, fault tolerant middleware and software rejuvenation, or the use of error correction codes in hardware storage or communication systems.

At the third chokepoint, between failures and hazards, typical techniques would include failure management capabilities, such as recovery, redundancy, an overarching failure management strategy, as well as fail-safe and fail-operational capabilities.

At the fourth chokepoint, once a hazardous state occurs it may still be possible to recognize that a hazardous condition exists, and prevent an accident by implementing accident avoidance capabilities. Techniques for survivability, including system resilience and the concept of degraded modes of operation, such as in [9] and [10] are likely to be applicable at this chokepoint as well.

At the fifth chokepoint, even after a loss-event occurs, there are mitigation techniques that can be applied. For example in an automobile, if an accident occurs, air bags and guard rails are mitigation techniques to minimize the resultant damage. Examples of generic techniques are more difficult to define at this chokepoint; they seem to be more system dependent, although analysis techniques such as Failure Modes and Effects Analysis are applicable.

The damage mitigation techniques for software-intensive systems naturally have been focused more at the beginning of the damage process with less emphasis towards the end. Much is known and published about fault avoidance and fault tolerance. Less is known about failure management. Little has been written about the recognition and management of hazardous conditions in software-intensive systems. Loss-event avoidance appears to be relatively unexplored, with the exception of Sha’s work on a Simplex Architecture [11], [12] on safety-critical systems, and the previously mentioned work on defining survivable core capabilities [10] and [9].

Conclusion and Future Work

The objective of this article is to provide a conceptual framework for damage mitigation and trusted systems to enable the research, design, and operation of mission-critical systems. While it has been used successfully in a graduate level software engineering course, this work is still in an embryonic stage.

Future work includes:

• Conducting case studies of actual mission-critical systems to determine the utility of the proposed framework, the classification of existing damage mitigation techniques and architectures, and exploring novel techniques which the framework suggests.
• Analyzing techniques that may be used to recognize hazardous conditions during runtime as a means to deploy damage mitigation.
• Investigating the use of control system theory as a means to monitor and control the damage mitigation process.
• Researching the relationships between the attributes of safety, security, dependability, and survivability. 

U.S. Department of Defense Systems Engineering

THE U.S. DEPARTMENT OF DEFENSE APPLIES BEST ENGINEERING PRACTICES TO
• SUPPORT WARRIORFIGHTER OPERATIONS; MANAGE RISK WITH DISCIPLINE
• GROW ENGINEERING CAPABILITIES TO ADDRESS EMERGING CHALLENGES
• CHAMPION SYSTEMS ENGINEERING AS A TOOL TO IMPROVE ACQUISITION QUALITY
• DEVELOP FUTURE TECHNICAL LEADERS ACROSS THE ACQUISITION ENTERPRISE

The U.S. Department of Defense seeks experienced engineers dedicated to delivering technical acquisition excellence. See www.usajobs.gov

Director of Systems Engineering - Office of the Director, Defense Research and Engineering
3040 Defense Pentagon • Washington, DC 20301-3040 • http://www.acq.osd.mil/se

Homeland Security

The Department of Homeland Security, Office of Cybersecurity and Communications, is seeking dynamic individuals to fill several positions in the areas of software assurance, information technology, network engineering, telecommunications, electrical engineering, program management and analysis, budget and finance, research and development, and public affairs. These positions are located in the Washington, DC metropolitan area.

To learn more about the DHS Office of Cybersecurity and Communications and to find out how to apply for a vacant position, please go to USAJOBS at www.usajobs.gov or visit us at www.DHS.gov; follow the link Find Career Opportunities, and then select Cybersecurity under Featured Mission Areas.
REFERENCES


ABOUT THE AUTHORS

Linda Laird is the Director of the Software Engineering Program at Stevens Institute of Technology. Professor Laird is the author of Software Estimation and Measurement: A Practical Approach. Her research interests include system dependability, architecture and design for software reliability, and software estimation. She is a graduate of Carnegie-Mellon University and the University of Michigan. Professor Laird managed large software development projects at Bell Laboratories for over 25 years.

Jon Wade, Ph.D. is the Associate Dean of Research at the School of Systems and Enterprises at the Stevens Institute of Technology. Dr. Wade’s research interests include the transformation of systems engineering, Enterprise Systems and Systems of Systems, and the use of technology in technical workforce development. He has over 20 years of experience in the research and development of Enterprise systems at IGT, Sun Microsystems and Thinking Machines Corporation. Dr. Wade is a graduate of the Massachusetts Institute of Technology.

Linda M. Laird, Jon P. Wade
Stevens Institute of Technology
School of Systems and Enterprises
Castle Point on Hudson
Hoboken, NJ 07030

WANTED

Electrical Engineers and Computer Scientists
Be on the Cutting Edge of Software Development

The Software Maintenance Group at Hill Air Force Base is recruiting civilian positions (U.S. Citizenship Required). Benefits include paid vacation, health care plans, matching retirement fund, tuition assistance and time off for fitness activities. Become part of the best and brightest!

Hill Air Force Base is located close to the Wasatch and Uinta mountains with many recreational opportunities available.

Send resumes to: phil.coumans@hill.af.mil
or call (801) 586-5325
Visit us at: http://www.309SMXG.hill.af.mil