Software Engineering Institute | Carnegie Mellon University

The Influence of System Properties on Software Assurance and Project Management

Robert J. Ellison

February 2006

ABSTRACT: Certain characteristics of software being developed and of the development environment influence how software assurance should be managed. The scope and size of the system are obvious attributes to consider. A large system is built by first decomposing it into pieces that are more easily managed. Project management decisions often influence how that decomposition is done.

DEVELOPMENT CONTEXT

Certain characteristics of software being developed and of the development environment influence how software assurance should be managed. The scope and size of the system are obvious attributes to consider. We build a large system by first decomposing it into pieces that are more easily managed. Project management decisions often influence how we do that decomposition. A decomposition might be constrained by a need to integrate legacy systems, a requirement to use commercial products, or a desire to reduce costs by reusing available software. The challenge is to decompose the system in such a way that those individual pieces can be individually built and that the composition of those components meets system requirements.

Software assurance is strongly influenced by how we integrate components and by the knowledge and control that we have with respect to the behavior of those components. We consider the consequences for software assurance and project management as the development context moves from general-purpose software such as commercial products to organization-specific applications and integrated systems and then to system of systems that may span organizational boundaries.

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-2612

Phone: 412-268-5800
Toll-free: 1-888-201-4479

www.sei.cmu.edu

GENERAL GUIDANCE

Risk analysis should be threaded through the development process. A series of risk assessments provides a measure of how well the previously identified risks have been addressed.

There should be a close tie between the risk analysis and requirements as risk analysis helps to define the scope for security in terms of the threats to be considered, the responses desired, and the assurance levels desired.

GENERAL PURPOSE COMPONENTS AND COMMERCIAL PRODUCTS

The complexity associated with product development may be a consequence of tight component integration to meet market demands for functionality or performance. Products typically have extensibility requirements so that they can be tailored for a specific customer’s operating environment. The complexity induced by those product requirements also raise the risk that those features could be exploited.

Cost reduction is a frequent business driver that encourages the development of software packages that are used in multiple systems within an organization. Often cost reduction leads to sharing infrastructure services among multiple applications.

While the size and scope of commercial products can be significantly larger than that for libraries of shared business services, these two development contexts share traits that affect software assurance.

- The developer for commercial software typically can change any of the product software and has detailed knowledge its structure. Shared business components are typically small, with the developer having control over most aspects of their development.
- General-purpose business software can be used in applications with differing assurance requirements, although such software is typically avoided for systems with high-assurance requirements. The developer for a commercial product has limited or no knowledge of the criticality of the services that may be implemented with that product, the business impact of a product failure, or the operational risks associated with any specific usage.

Example

There is increasing interest in using service-oriented architectures (SOA) and Web Services to satisfy the escalating integration requirements among distributed business systems. That approach raises several software assurance issues for applications and systems. There are multiple definitions of SOA. This note considers it as an architecture in which the business logic of the application is organized in modules (services). Each module is a discrete service, and its internal design is independent of the nature and purpose of the requester (i.e., loosely coupled). The objective is to develop reusable services (components) that can be easily composed to implement multiple and frequently changing business work processes. The difficulty of assuring the behavior of a large, tightly coupled application has been replaced by the difficulty of

- assuring not only a specific business service but also assuring those services as used in multiple business processes with potentially differing performance, security, or reliability requirements. For example, a business process may use stronger authentication methods or limit data access with a remote business process when the process participants are not on a trusted network.
- assuring that those services can be composed to support complex and ever changing business processes

Selected Project Management Issues for Products and Shared Services

1. Code analysis, both manual and tool-based, can be applied to all software to find and correct many of the vulnerabilities associated with coding.
2. Design reviews should reflect the breadth of potential usage. There should be sufficient access to all component software to enable correcting the design or implementation faults identified.
3. Threat modeling and risk analysis should assume a spectrum of attacker profiles, from a novice using a readily available attack kit to a sophisticated attacker with ample resources who can tailor an attack for a specific system.
4. Shared services typically aggregate risks. A failure in shared software or infrastructure services could affect multiple systems. The level of software assurance required for the shared components should be higher than that required than for the systems in which they are deployed. The higher assurance and aggregation of risks implies that the risks for shared services should include the full spectrum of integrity, confidentiality, and availability issues.

Microsoft’s Trustworthy Computing Security Development Lifecycle provides a detailed description of their experience for incorporating software assurance into development life cycle for software products [Lipner 05].

PRODUCTS/COMPONENTS TO APPLICATIONS/INTEGRATED SYSTEMS

The difficulty of demonstrating software assurance is compounded by two problems as we move to applications and integrated systems. Commercial products and shared components are used to build applications and integrated systems. Compared to the original developer, the application or system developer has limited knowledge of their internal structure and control of their behavior and cannot provide as strong an assurance argument as that provided by the product developer. While the product developer should demonstrate software assurance for a spectrum of uses, the system developer only has to demonstrate assurance for how products or shared components are used in that system.

Component Assurance Issues

The section on requirements and scope included examples of medium and high assurance systems. Although there are many studies on the problems raised by low assurance, medium assurance is more of a grey area in terms of the development practices that should be used. Some will argue that we cannot compose a medium-assurance system from low-assurance components, and that medium assurance has to be built in from the start of development. Medium- and high-assurance requirements are often in conflict with other development strategies such as the reuse of existing code to lower costs and shorten development time. Reuse can lead to software being employed in an environment where it was not designed to be secure or to a combination of components that were not designed to manage failure. The addition of extensibility mechanisms so that a product can be customized for specific usage usually raises the security risks. Hence, medium- and high-assurance systems may need to use more special purpose mechanisms than the more flexible and general-purpose techniques that are used for low-assurance systems.

System Assurance Issues

There is frequently a tendency when considering system security to focus on the security of the components. Unfortunately, the whole is not the sum of the parts. Security is an emergent system attribute and depends on the collective behavior of the components. Although many security failures have been caused by a component error, security vulnerabilities also arise from unexpected interactions among components and from inconsistencies in the design and operational assumptions among the subsystems.

While the problems associated with networked computing systems may be receiving the most attention, the difficulties associated with managing large systems have a long history. John Gall first published Systemantics in 1975. The third edition was published in 2002 and renamed The Systems Bible [Gall 02]. While the wry humor in The Systems Bible is entertaining, the discussions should be a reminder that many aspects of large system assembly and integration are not well understood and that the methods that were successful for components and relatively simple systems do not necessarily scale to large systems.

Grady Booch listed some of his favorite Gall axioms in his Web log on Feb 15, 2005 [Booch 05]. Several of those suggest the software assurance and project management issues that arise with systems and systems of systems.

- A large system that is produced by expanding the dimensions of a smaller system does not behave like the smaller system.
- A complex system that works is invariably found to have evolved from a simple system that worked.

Both safety and security are emergent system properties. Nancy Leveson's discussion of the consequences for safety analysis of that emergent behavior also applies to security [Leveson 05].

Safety is an emergent property of systems. Determining whether a plant is acceptably safe is not possible by examining a single valve in the plant. In fact, statements about the "safety of the valve," without information about the context in which that valve is used, are meaningless. Conclusions can be reached, however, about the reliability of the valve, where reliability is defined as "the ability of a system or component to perform its required functions under stated conditions for a specified period of time," i.e., that the behavior of the valve will satisfy its specification over time and under given conditions. This is one of the basic distinctions between safety and reliability: Safety can only be determined by the relationship between the valve and the other plant components—that is, in the context of the whole. Therefore it is not possible to take a single system component, like a software module, in isolation and assess its safety. A component that is perfectly safe in one system may not be when used in another. Attempts to assign safety levels to software components in isolation from a particular use, as is currently the approach in some international safety standards, is misguided.

The vulnerabilities generated by interactions among multiple system components are much more difficult to locate and predict than the vulnerabilities associated with a specific component or technology. The issue is not whether a component is secure or insecure but whether that component can be securely used in a system with specific threats. Security issues often arise with the use of commercial components. Such components may need to be incorporated into a system with stronger surety requirements than those satisfied by the individual components.

Selected Project Management Issues for Application and System Development

Security is a system property. Whereas the implementation of a specific application feature might be contained in a single component, a security feature may require coordinated actions by multiple components or may depend on design or coding guidelines that must be followed for all components. A well-designed system might not depend on a single component to prevent an attack but instead would implement multiple defenses so that the failure of a single mechanism would not expose the system to attack.

Integration introduces dependencies and feedback among components that may lead to an unintentional system or operational error or allow a maliciously induced fault. Individual component design decisions on how to recover from and report faults, either accidental or malicious, can lead to undesirable system behavior. A safe shutdown from a component perspective may induce a significant system failure.

1. Risk analysis occurs at multiple levels of the system. A general system risk analysis and threat model is an essential input into the design of the overall system software architecture. That architecture should describe the response to the identified risks in terms of the responsibilities of the software components, users, and operations. Risk-based testing should be applied to both components and to the system. (See White Box Testing and Security Testing content.)
2. The design and implementation of components may generate system risks. The outcome of the component risk analysis should be input for the periodic system risk analyses.
3. The user's risk analysis for commercially available software may not have to consider as wide a spectrum of threats as the analysis done by the vendor. A user's risk analysis only has to consider how that software is used in the specific system with the identified threats for that context.
4. Poor coordination and communications are often listed among the reasons for project failures. System integration has to resolve any mismatches with both internal and outsourced development. One mechanism to encourage better integration might be to specify the software assurance criteria for each component such as completed code analysis for all delivered software. There are likely differences in the software assurance requirements among components developed in-house and those commercially available.
5. Systems typically have a shared infrastructure; hence, item four on the list of project management issues for components and shared services is a critical requirement for system development.

6. The system risk assessment has to consider the functional design decisions as well as the technologies that are used. Functional design decisions may have significant security implications. A design that uses a central data server has very different risks than a design that is based on peer-to-peer information sharing.
7. Just as for other aspects of the software, the early phases of development are a learning curve for the security requirements associated with the desired functionality. Reports of possible vulnerabilities must be treated differently from other defects, as they represent a threat to users and other stakeholders. There will be a significant number of unknowns when the business application or supporting technology is on the leading edge. A serious mismatch between the functional architecture and the security requirements that is only recognized late in the development process is typically both difficult and expensive to resolve.
8. Many quality requirements such as those for security are known only abstractly in the early life-cycle phases. An early risk analysis might identify general threats such as insider attacks for a financial institution. A vulnerability analysis model with more detailed attacker actions and possible responses requires a more detailed description of the software such as that provided by the software architecture or a detailed design. The refinement of these requirements could lead to changes in the project plan.

SYSTEMS TO SYSTEMS OF SYSTEMS

The expanding role of Web-based information exchange and technologies such as Web Services is leading to the deployment of more systems of systems. We use Mark W. Maier’s criteria to distinguish systems of systems from large-scale monolithic systems [Maier 96]. A system of systems characterized by

- operational independence: If the system of systems is disassembled into its component systems those component systems must be able to effectively operate independently. The system of systems is composed of systems which are independent and useful in their own right.
- managerial independence: The component systems not only can operate independently, they do operate independently. The component systems are separately acquired and integrated but maintain a continuing operational existence independent of the system of systems.
- evolutionary development: The system of systems does not appear fully formed. Its development and existence are evolutionary with functions and purposes added.

- emergent behavior: The system performs functions and carries out purposes that do not reside in any component system. These behaviors are emergent properties of the entire system of systems and cannot be localized to any component system.
- geographic distribution: The geographic extent of the component systems is large. Large is a nebulous and relative concept as communication capabilities increase, but at a minimum it means that the components can readily exchange only information and not substantial quantities of mass or energy.

Example

An event-driven architecture is an alternative to implementing a business-to-business transaction with an online application. In an event-driven architecture, a business transaction corresponds to an asynchronous message sent from the buyer to a supplier. An acknowledgement or a later shipping notice would be a message from the supplier to the seller. The arrival of such messages can be thought of as events. The advantage of asynchronous communications is that neither party ties up computing resources waiting on a response from the other.

Such a purchasing process represents a simple system of systems and is a good example of the differences in design guidance between an integrated system or application and a system of systems. Both parties in the system of systems implementation have to operate without full control and visibility of the total process.

In an IT application, authentication and authorization can be checked whenever data are accessed. For an event-driven architecture, the authentication and authorization might be implemented by a Web Services protocol. The message not only contains the data but must also satisfy the authorization and authentication requirements for both parties. Encryption might be used by the sender to restrict access to the information. Signing might be used to help in identifying the source of an order. The authentication and authorization of an individual who submitted an order would typically have been verified on the purchaser’s system. The message may contain information that describes the details of that authentication that could be reviewed by the supplier.

Business to business transactions may be supported by a contractual agreement that defines the responsibilities and liabilities of both parties. Such an agreement may impose software assurance requirements on both parties.

Some Project Management Issues for Application and System Development

1. The threats for any single system in a system of systems are propagated to the other participants.
2. Whereas prevention is a frequent response to a component or application risk, a system participating in a system of systems cannot control how the risks are mitigated by the other systems. There will be a greater spectrum of errors, and it is difficult to distinguish malicious errors from normal operational errors. The architect for a participant in a system of systems now has to demonstrate software assurance for that system's behavior in the context of the system of systems faults.
3. Many of the design guidelines that support component or application development should be carefully revised for a system of systems. The analysis of a system used in a system of systems is done with incomplete information about the other participants. The design of a system-of-systems interface has to reflect the complexity of error handling in that context.
4. Static analysis is often effective for a component or application, but the analysis of a system of systems can depend more on run-time analysis of behavior because of the limited knowledge of other participants. The difficulty of analysis is compounded by the evolutionary nature of a system of systems.

INFLUENCE OF SYSTEM PROPERTIES ON RESOURCES

Estimates

Shared infrastructure can reduce component development costs, but those shared services typically aggregate risks. Estimates should reflect the increased assurance that be applied to the shared services.

Systems and systems of systems can raise havoc with estimates. As noted by Leveson, it is very difficult to identify vulnerabilities that arise from the integration of components or systems [Leveson 05]. Unanticipated behavior can appear during integration testing, and the resolution of such problems may require component or architectural changes. Systems of systems requirements typically increase the costs for the development of a participating system. The implementation of dynamic (run-time) analyses of system-of-systems interfaces and the system's response to adverse system-of-systems events is more expensive than the static analyses and preventive measures frequently used for an integrated system or application.

Facilities and Staffing

The development context—component, system, or system of systems—also influences the skills required. An analysis of the emergent behavior of a system or system of systems is quite different from the vulnerability analysis of a component. The assurance of a system and system of systems likely requires the assurance of any shared infrastructure services. A risk assessment requires experience with risk analysis and with applying it in the corresponding development context.

Other Related Estimates Such as Size and Defects

A number of Gall's axioms suggest the difficulties with estimating defects with large systems [Gall 02].

- Any large system is going to be operating most of the time in failure mode.
- The mode of failure of a complex system cannot ordinarily be determined from its structure.
- One does not know all the expected effects of known bugs.

Vulnerabilities arise as we put the pieces together. The sources of potential errors now include

- specific interface: An interface controls access to a service. Interfaces that fail to validate the input stream are frequent members of published vulnerability lists.
- component-specific integration: Assembly problems often arise because of conflicts in the design assumptions for the components. Project constraints may require using components, COTS software, or legacy systems that were not designed for the operating environment, which raises the likelihood of mismatches. The increasing importance of business integration requirements compounds the component integration problems.
- architecture integration mechanisms: Commercial software tool vendors often provide the capability for the purchaser to integrate the tool into their systems and tailor its functionality for their specific needs. However, the capability to reconfigure a system rapidly is matched by the increased probability of component inconsistencies generated by the more frequently changing component base, as well as the increased risk that the dynamic integration mechanisms could be misused or exploited. These mechanisms represent another interface that must be properly constrained.

- system behavior — component interactions: The behavior of a system is not the simple sum of the behavior of the individual components. System behavior is strongly influenced by the interactions of its components. Components may individually meet all specifications, but when they are composed into a system the unanticipated feedback among components can lead to unacceptable system behavior. Security and safety are system rather than component requirements. We can build a reliable system out of unreliable components by appropriate use of redundancy. Components that are not secure as standalone components in an operating environment may be secure when used within the constraints maintained by a system.

Each source of errors requires its own analysis. (See the Assembly, Integration & Evolution content area for more detail.) The errors associated with system behavior challenge the traditional approach to failure analysis. The assumption for almost all causal analysis for engineered systems today is a model of accidents that assumes they result from a chain (or tree) of failure events and human errors. From an observed error, the analysis backward chains and eventually stops at an event that is designated as the cause [Leveson 05].

Event-based models of accidents, with their relatively simple cause-effect links, were created in an era of mechanical systems and then adapted for electro-mechanical systems. The use of software in engineered systems has removed many of the physical constraints that limit complexity and has allowed engineers to incorporate greatly increased complexity and coupling in systems containing large numbers of dynamically interacting components. In the simpler systems of the past, where all the interactions between components could be predicted and handled, component failure was the primary cause of accidents. In today's complex systems, made possible by the use of software, this is no longer the case. The same applies to security and other system properties: While some vulnerabilities may be related to a single component only, a more interesting class of vulnerability emerges in the interactions among multiple system components. Vulnerabilities of this type are system vulnerabilities and are much more difficult to locate and predict [Leveson 05].

BIBLIOGRAPHY

[Booch 05]	Booch, Grady. Architecture Web Log. http://www.booch.com/architecture/blog.jsp (2005).
[Gall 02]	Gall, John. The Systems Bible. Walker, MN: The General Systemmantics Press, 2002.
[Leveson 05]	Leveson, Nancy. “A Systems-Theoretic Approach to Safety in Software-Intensive Systems.” IEEE Transactions on Dependable and Secure Computing 1, 1 (January-March 2004): 66-86.
[Lipner 05]	Lipner, Steve & Howard, Michael. The Trustworthy Computing Security Development Lifecycle. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsecure/html/sdl.asp (March 2005).
[Maier 96]	Maier, Mark W. Architecting Principles for Systems-of-Systems. http://www.infoed.com/Open/PAPERS/systems.htm (1996).