SEI Insights

SEI Blog

The Latest Research in Software Engineering and Cybersecurity

The Need to Specify Requirements for Off-Nominal Behavior

POSTED ON JANUARY 16, 2012 BY **DONALD FIRESMITH** [/AUTHOR/DONALD-FIRESMITH] IN **ACQUISITION** [HTTPS://INSIGHTS.SEI.CMU.EDU/SEI_BLOG/ACQUISITION/]

In our work with **acquisition** [http://www.sei.cmu.edu/solutions/acquisition/] programs, we've often observed a major problem: requirements specifications that are incomplete, with many functional requirements missing. Whereas requirements specifications typically specify normal system behavior, they are often woefully incomplete when it comes to off-nominal behavior, which deals with abnormal events and situations the system must detect and how the system must react when it detects that these events have occurred or situations exist. Thus, although requirements typically specify how the system must behave under normal conditions, they often do not adequately specify how the system must behave if it cannot or should not behave as normally expected. This blog post examines requirements engineering for off-nominal behavior.

Examples of off-nominal behavior that are inadequately addressed by requirements specifications include how robust (i.e., error, fault, and failure tolerant) must the system be, how the system must behave when hardware fails or software defects are executed, how the system must react when incorrect data (e.g., out of range or incorrect data type) is input, and what should happen if the system detects that it is in an improper mode or inconsistent state. This lack of requirements

specification can lead to the following omissions and questions that must be asked as a result.

- **All credible conditions and events.** How must the system behave under off-nominal sets of preconditions and trigger events that are unlikely and/or infrequent? When these conditions occur--as they invariably will--there is a risk that the system either does not handle them or the developers have been forced to guess (often incorrectly) how the system must behave. The requirements therefore need to specify how the system shall behave under all *credible* combinations of conditions and trigger events. Moreover, how are combinations of rare conditions and events determined to be not credible? Users and requirements engineers often underestimate the probability of rare occurrences, so they are surprised when they occur and the system reacts improperly. If these off-nominal conditions and the desired behavior of the system to them are not identified and documented early in the lifecycle, the decisions about what error/fault conditions should be handled by the system are left to individuals who may not have the proper expertise to identify such conditions, but who nevertheless feel compelled to make such decisions.
- **Detecting off-nominal situations.** How will the system recognize off-nominal combinations of conditions and events? Does the system need sensors to determine the existence of these states or occurrence of these events? How available, reliable, accurate, and precise must these sensors and inputs be?
- **Reacting to off-nominal situations.** How must the system react when it recognizes an off-nominal combination of conditions (possibly when a specific, associated event occurs)? Must it notify users or operators by providing warnings, cautions, or advisories? Must it do something to ensure that the system remains in a safe or secure state? Must the system be able to shut down in a safe and secure state or must it automatically restart? Must it record abnormal situations and the responses of the users/operators?
- **Incomplete use case models.** Use case modeling is the most common requirements identification and analysis method for functional requirements. Each use case has one or more normal (so-called "sunny day") paths (a.k.a., courses and flows) as well as several exceptional ("rainy day") paths. Unfortunately, requirements engineers often concentrate so heavily on normal paths that there is inadequate time and staffing to properly address the credible exceptional paths. This omission leads to incomplete requirements specifications that do not adequately address necessary robustness (e.g., error, fault, and failure tolerance), reliability, safety, and security.
- **Coding standards.** Programming languages typically include features and reusable code (e.g., base classes that come with the language) that are inherently unreliable, unsafe, and insecure. Because language features may not be well-defined in the language specification, their behavior

may be inconsistent. For example, the use of **concurrency** and **automated garbage collection** can lead to common defects, such as **race conditions**, **starvation**, **deadlock**, **livelock**, and **priority inversion**. Likewise, certain language features may well be used in an incomplete manner. For example, an if/then/else clause may not contain an else clause stating what to do if the if clause precondition is not true. Similarly, a *do X* followed by *do Y* may not say what to do if *X* fails to complete. There are other cases such as *divide by zero situations, taking the square root of negative numbers,* and *a lack of strong typing*, as well as no verification of inputs, preconditions, invariants, post conditions, and outputs. These implementation coding defects typically start as requirements defects: incomplete requirements that do not mandate the use of reliable, safe, and secure subsets of the language, safe base classes, and automatically verified coding standards.

- **Lack of subject matter expertise.** Exception handling is often left to the programmers, who must ensure their software is error, fault, and failure tolerant and meets its requirements. Programmers will be blamed if defects prevent the system from being available, reliable, robust, safe, and secure, even if there are no relevant requirements. Unfortunately, programmers often make assumptions as to what the software's off-nominal behavior should be. Without adequate domain expertise and sufficient contact with subject matter experts, programmers will incorporate defects and safety/security vulnerabilities. Likewise, poor quality requirements specifications show how requirement engineers struggle to address mandatory off-nominal requirements since they lack sufficient domain expertise and training to determine, analyze, and specify adequate availability, reliability, robustness, safety, and security requirements. Ultimately, the engineering of these quality requirements requires subject matter expertise that is rarely combined in any one developer.

There are often many more off-nominal (and rare) combinations of conditions and events than the common nominal ones. There are also often many more ways that a system can fail. Requirements specifications are typically incomplete with regard to the previous problems, often only including 10 to 30 percent of the necessary requirements. This level of incompleteness can result in systems that fail to meet their true availability, reliability, robustness, **safety, and security requirements** [https://insights.sei.cmu.edu/sei_blog/2011/06/the-importance-of-safety--and-security-related-requirements-first-of-a-three-part-series.html]

.

It is insufficient for requirements specifications to state that the system shall be highly available, reliable, robust, safe, and secure, or that it has no single points of failure. The requirements must specify all credible off-nominal combinations of conditions and events. Otherwise, software developers will make incorrect guesses, have incorrect assumptions, and ignore important off-

nominal situations. Without complete requirements, verification will not catch these defects, and the resulting defective system will be fielded with highly unfortunate, if predictable, results. Because program offices cannot safely assume that the contractor will automatically address these issues, programs cannot safely leave it up to their contractors. Off-nominal situations must be properly addressed in the requirements.

Studies (**Knight** [http://dl.acm.org/citation.cfm?id=647400.724997], **Weiss** [http://sunnyday.mit.edu/accidents/soho.doc], **Leveson** [http://sunnyday.mit.edu/papers/sae.pdf]) have shown that the vast majority of accidents (safety) and many common software vulnerabilities (security) result at least partially from incomplete requirements. Many availability and reliability defects due to software also result, at least partially, from incomplete requirements.

Recommended Solutions

To address the problems described above, acquisition program offices should consider the following steps:

- **Address off-nominal requirements in the contract.** The program office should contractually mandate all significant off-nominal behavior. The contract should also mandate that contractors address all credible off-nominal conditions and events affecting mission, safety, and security functionality.
- **To address all credible conditions and events,** the program office should ensure that the contractor's requirements engineering plan explicitly states that all credible combinations of conditions and events are to be addressed, even very rare ones, if the corresponding function is mission, safety, and/or security critical. The program office should verify that the requirements engineers collaborate with reliability, safety, and security engineers to ensure than no significant combinations of states and events be overlooked. The program office should also ensure that the proper testing of the associated software in terms of test completion criteria and test case generation criteria is explicitly addressed in the software test plans.
- **To detect off-nominal situations,** the program office should ensure that the contractors have properly addressed the detection of off-nominal situations. Specifically, this includes verifying that the **system engineering management plan (SEMP)** as well as the system requirements, architecture, and design address situational awareness in terms of both sensors and the input of necessary data concerning off-nominal situations.
- **When reacting to off-nominal situations,** the program office should ensure that the requirements address how the system must behave if it cannot behave in the nominal manner. This process includes ensuring that the system either remains in a safe and secure state or shuts

down safely and securely (i.e., is fail-safe). It also includes notifications (warnings, cautions, and advisories) as well as logging any associated error, fault, or failure information.

- **To ensure against incomplete use case models,** the program office should ensure that they include all credible normal and exceptional use case paths, and also ensure that adequate project schedule, budget, and staffing are allocated to complete the models. Verification of the requirements and their associated models should explicitly address exceptional as well as normal use case paths.
- **With respect to contractor coding standards,** the program office should ensure they explicitly address eliminating common design and coding defects that make the software less available, reliable, robust, safe, and secure. The program office should also ensure these coding standards are properly followed including, where practical, automatic verification via static and dynamic code checking.
- **With respect to subject matter expertise,** the program office should ensure that associated quality requirements mandating adequate availability, reliability, robustness, safety, and security are engineered by cross-functional teams of closely collaborating requirements engineers, subject matter experts, stakeholders, and engineers specializing in reliability, safety, and security. This team must identify the appropriate credible off-nominal situations and decide which of these situations should be analyzed and turned into associated requirements given programmatic constraints such as cost, schedule, available development staffing, and critical functionality. The program office should also ensure that the contractors and subcontractors use appropriate coding standards and associated foundational software (e.g., a safe and secure subset of C++, including safe and secure base classes).

Most acquisition programs suffer from incomplete requirements, especially with regard to dealing with rare combinations of states and events, detecting and reacting to off-nominal situations, use-case models that are incomplete due to missing exceptional use case paths, and either inadequate coding standards or coding standards not being followed. The engineers who actually develop the software often lack adequate expertise in availability, reliability, robustness, safety, and security requirements, which yields systems that do not meet their associated requirements. While the problems are well known, so are their answers. Program offices, therefore, must ensure that these answers are implemented, enforced, and verified to be effective and efficient.

Additional Resources:

To read Don Firesmith's series on The Importance of Safety- & Security-Related Requirements, please visit
http://insights.sei.cmu.edu/archives.cfm/category/safety-related-requirements

[http://insights.sei.cmu.edu/archives.cfm/category/safety-related-requirements]

About the Author

Donald Firesmith

Contact Donald Firesmith [https://www.sei.cmu.edu/contact.cfm]
Visit the SEI Digital Library for **other publications by Donald**
[http://resources.sei.cmu.edu/library/author.cfm?authorID=4637]
View **other blog posts by Donald Firesmith**
[/author/donald-firesmith]