

Elicitation Practices That Can Decrease Vulnerability to Off-Nominal Behaviors: Lessons from using the Causal Component Model

Daniel Aceituna
DISTek Integration Inc.

ABSTRACT

When specifying an embedded system-to-be, a key consideration is how the embedded system will interact with its operating environment. Of particular concern is the system's vulnerability to Off-Nominal Behaviors (ONB) from human interaction. ONB vulnerability can result in human operators placing the system in an undesired state through an unforeseen sequence of events. This, in turn, can have an adverse effect on the system's quality. Reducing ONB vulnerability can be challenging because human behavior can be unpredictable and stakeholders have a natural tendency to assume the system will be used in a predictable, nominal, manner. One approach to reducing ONB vulnerability is to specify the system as "fool-proof" as possible, during the requirements phase, where access to domain experts is at its most convenient. This also raises awareness of potential ONB problems prior to the design phase, rather than after implementation where, quite often, ONBs are addressed through off-nominal testing. This paper presents a checklist of requirement elicitation questions that can result in lower ONB vulnerability. The checklist is derived from lessons learned from using a newly developed requirements model, the Causal Component Model, on several sets of requirements. The checklist, along with an introduction to the Causal Component Model, and how it can be a useful addition to a model based design methodology, is presented in this paper.

CITATION: Aceituna, D., "Elicitation Practices That Can Decrease Vulnerability to Off-Nominal Behaviors: Lessons from using the Causal Component Model," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 10(1):2017, doi:10.4271/2016-01-8109.

INTRODUCTION

We view a reactive system as a collection of components, whose behaviors are causally interrelated with one another. These components collectively respond to stimulus external to the system, with transitions to the next system state depending on both the system's present state and the external stimulus. In reactive systems, where there are considerable interactions with the system's operating environment, it is not unusual for the environment to behave in a manner unaccounted for by the system specification. Of particular concern are those interactions that come from the system's human operators. Human behavior can be unpredictable and off-nominal, contrary to what the designers of a system consider normal behavior for the operator [1].

Stakeholders typically make assumptions when specifying a set of system requirements. Key to these assumptions is the way in which the system and its operating environment are intended to behave; these intended behaviors are treated as nominal. Quite often, unintended, unusual behaviors are not accounted for; therefore, not every contingency the system might have to address is included in the requirements [2]. We refer to these unintended behaviors as Off-Nominal Behaviors (ONBs) [3]. In some cases, ONBs can result in a safety hazard or damage to the equipment being controlled. We refer to the lack of contingencies for ONBs as the ONB vulnerability problem. It is generally recognized that the sooner potential problems, such as ONB vulnerabilities, are exposed, the less expensive and easier these problems are to address [4]. Typically the

earliest phase in the development of a system is the requirement's phase, in particular the requirement elicitation activity, which in some cases is the only requirements activity that a company may perform. The primary question addressed in this paper is: How do we reduce a system's vulnerability to ONBs during the elicitation phase? In addressing that question two contributions are presented here. First there is the requirement modeling technique called CCM [5, 6]. CCM can be used to analyze a system's vulnerability to ONBs.

One goal of a system model is to abstract the characteristics of interest, so as to facilitate the analysis of those characteristics [7]. In the case of CCM, the abstracted characteristics are the system's state space, the transitions between states within that state space, and the distinction between transitions caused by the system's operating environment and transitions caused by the system in reaction to the environment. Abstracting these particular characteristics help address the ONB problem, since 1) ONBs are often induced on a system by the environment (e.g. human interaction). 2) A system's reaction to the environment is a product of the possible transitions within its state space.

The second contribution comes from applying the CCM to various sets of requirements and observing correlations between the exposed vulnerability to ONBs and elicitation practices. These correlations were compiled into a checklist of questions (according to recommended practices [8]) designed to expose potential ONB related problems during elicitation to help minimize system

vulnerability to ONBs that are due to ambiguities and incompleteness. The checklist can then be applied during the elicitation process, or as part of a requirements document review. The goal of this research is not to eliminate the need for off-nominal testing, and human behavior modification, but rather offer alternatives that can be used to enhance the reduction and elimination of ONB vulnerabilities before a system is designed, rather than after the system's implementation. It also provides an alternative to correcting human behavior by addressing the ONB problem from the system's perspective; namely, making the system less vulnerable to unintended human behavior. Another goal of this research is to raise awareness that ambiguity in requirements can result in greater vulnerability to ONBs. We will walk the reader through an example of ONB related questions that can arise in a growing set of requirements. The reader will then see how CCM can raise tough questions, which in turn can be used to compile a checklist.

BACKGROUND AND RELATED WORK

The background pertaining to this research along with related work falls into three categories. The Off-nominal behavior problem, research pertaining to the inspection of requirements, and the various techniques used to model requirements. We begin with the ONB problem.

Off-Nominal Behavior

ONBs can occur for various reasons, including, but not limited to, the following three situations [9]. 1) A system's human operator is confronted with an unexpected scenario, forcing the operator to react in an abnormal way. 2) The system's sensor data can be unexpectedly outside the range or of a different data type. 3) The assumed preconditions for the system's operation could be different than anticipated or even non-existent. Ideally, a system should be specified with the ability to contend with the three afore mentioned categories of ONBs. However, that is not always the case for various reasons, including the following.

Ignorance of the Problem Domain

This is the lack of anticipating everything that can occur in the operating environment and subsequently ignoring how the system will react to them. This is tied to performing an incomplete analysis of the problem domain, which translates into incompleteness in the requirements [28].

Over Reliance on Tacit Knowledge

In short, tacit knowledge is knowledge that is taken for granted. It can be background knowledge that seems so obvious to a stakeholder that he/she fails to mention it. Tacit knowledge contributes to ambiguous statements and the first item listed above: Ignorance of the problem domain.

System 1 Thinking

A mode of thinking that relies more on intuition and prior experience than on analytical analysis of a problem [10]. System 1 thinking and can result in hasty decision making. It is often used as a means of rapidly addressing problems, and can manifest itself by focusing on an immediate solution to a problem, and ignoring the possibility of unintended consequences.

The ONB problem is often addressed from the environmental standpoint, while focusing on the Human Machine Interface (HMI), because human behavior cannot be fully predictable and is prone to unexpected behaviors [11]. Many efforts have also been in the area of off-nominal testing which typically occurs at the end of the development cycle during software testing. These tests can involve scenarios designed to expose the software's response to an off-nominal case [12]. One study utilizes off-nominal test cases for airplane runway operations to expose possible ONBs [11]. As with testing in general, off-nominal testing can have limitations in exposing defects because exhaustive testing is not practical [13]. It is not unusual for a reactive system's user to find a system bug, after the system's release into the field, by interacting with the system in a way other than the "happy paths" [14]. "Happy paths" refer to test scenarios that represent the nominal behaviors the system is expected to produce. However, software systems tend to have a lot more transition paths than are typically tested, and there are many scenarios that are not accounted for, in particular those that are off-nominal [1]. NASA has recognized the importance of performing off-nominal testing as a means to assure that its software can deal with unpredictable contingencies [12].

The aviation industry has tried to address ONB problems [9] within the requirement phase, focusing on anticipating and specifying contingencies (in the form of scenarios) that address potential pilot errors [11]. Some researchers have used simulation-based models to create off-nominal conditions for air-traffic procedures [3]. Using scenarios and simulation-based models emphasizes the need to try to anticipate what can go wrong in the field and assures that the system can handle such occurrences [27].

Inspections and Reviews

Inspection is one of the most widely accepted and used verification techniques because it is relatively easy to apply and because it does not require technical knowledge for stakeholders to use it, especially when using a checklist as a guide. Inspections are also a form of manual static testing that some consider more cost effective than automated testing [8]. Other inspection variations include walkthroughs that involve the key stakeholders getting together to review a requirement document on a line-by-line basis [15].

Despite its popularity and merits, inspection can be error prone due to human limitations for finding faults in documents describing concurrent reactive systems [16]. Inspection lists tend to focus the user's attention on one requirement at a time. This narrow focus can make the verification of a reactive system difficult because a reactive system can involve many state transitions that often occur concurrently. Thus, it can be problematic when trying to verify all the possible functional interactions when inspecting one requirement at a time, which is the way inspectors typically review requirements. With large, manual inspection, methods can fall under three categories: Ad Hoc (reviewing a document with no well-defined criteria), Checklist (using a well-defined criteria), and Scenario-based (compiling system-to-be scenarios and verifying that those scenarios can be enacted by the requirements under inspection) [8]. Experiments have been conducted to determine the relative effectiveness of Ad Hoc, Checklist, and Scenario-based inspection approaches, and have pointed to the scenario-based approach as being the most effective of the three [17], however, even with scenarios, there is still the challenge of verifying a requirement set that specifies concurrent behavior.

There are two areas of concern that should be addressed while inspecting a requirements document: operational context and pre/post conditions [14]. The operational context addresses the fact that a system cannot be accurately specified without considering the context, the operating environment, in which it will be used [18]. Reactive systems, such as embedded systems, must be able to function correctly within the environment that they are meant to operate. The pre/post conditions are also an important aspect to consider because reactive systems act deterministically [19], given a user-input set or external asynchronous events (such as unpredictable interrupts). Their deterministic response is also affected by preconditions. Post conditions are important when trying to assess how the system responds to a negative situation. The two aforementioned areas of concern can be more easily addressed if one takes the approach of converting the NL requirements into a model. CCM was conceived with the idea of abstracting concurrent behavior, operational context and pre/post conditions

Requirement Modeling

Another area that is relevant to our work is system modeling, and much research has been performed in this area. Most definitions for a system share the concept of components interacting with one another in an interacting relationship [20]. Systems are viewed as having structure, interconnectivity, behavior, and a boundary between themselves and the environment where the system is meant to operate [21]. There are various ways to model a system, including a black-box approach, a state-theory approach, a component-integration approach, and a decision-process approach [7]. Systems can be modeled with various views which fall under the structural and behavioral categories. Other modeling approaches, such as graphs [22], are geared toward design modeling and require a greater amount of detail. Efforts to model embedded systems include the Embedded System Modeling Language (ESML) [23], which enables the modeling of components, interactions, and configuration. ESML uses a UML class diagram formalism and was conceived to overcome some inherent shortcoming in UML which hinder its ability to model Embedded systems.

Modeling is not limited to a graphical representation, but can be achieved mathematically, and with the use of scenarios [24]. Sutcliffe et al. [23], have proposed a method that converts use cases into scenarios semi-automatically and validates scenarios using rule-based frames that detect incomplete/incorrect event patterns. There is also petri-nets, which allow for the modeling of concurrency [25]. CCM utilizes a modeling scheme which captures the causal relationship, and concurrent behavior, between the system components. The CCM was conceived as a high-level model that abstracts what a system is supposed to do, not how it is supposed to do it. This high abstraction makes the CCM conducive to modeling requirements, whereas many other modeling schemes were conceived with design in mind.

OVERVIEW OF THE CAUSAL COMPONENT MODEL

The focus of this paper is the lessons learned from examining requirements that have been modeled using CCM and its various views. Thus, it is not within the scope of this paper to provide a complete, detailed, and formal definition of CCM; for such information the reader should refer to [5, 6]. We will, however, provide a summary of the different CCM concepts that are pertinent to the explanation of how the elicitation/review lessons were derived.

The CCM View

CCM is a model that represents a set of NL requirements as a set of inter-related State Transition Diagrams (STD), where each STD represents the behavior of a system component. Figure 1, section [C] shows the CCM of the motor controller described in the set of requirements in section [A] of the same figure. Note the three STDs, defining the behaviors of a switch, a temperature sensor, and an electric motor. Section [B] of Figure 1, shows a set of CS rules that were derived by translating the requirements [A], using a four step process consisting of: 1) Determining the system components, 2) Determining the states of those components (known henceforth as Component-States (CS)), 3) Determining the transition between components states, and 4) Determining the cause of those transitions. The formal definition of these four artifacts is beyond the scope of this paper [5, 6], so we will suffice with a summary of some key definitions.

Components

A component is defined as a subunit of a system which can assume a state, and exhibit a set of one or more behaviors. In Figure 1, the system is divided into three components: switch, temperature sensor and motor.

Component States

A component can assume one or more states. A Component State (CS) is defined as the state of a given component at any given time. Component states are denoted as Component(State). For example Motor(on), means the Motor is in an "on" state.

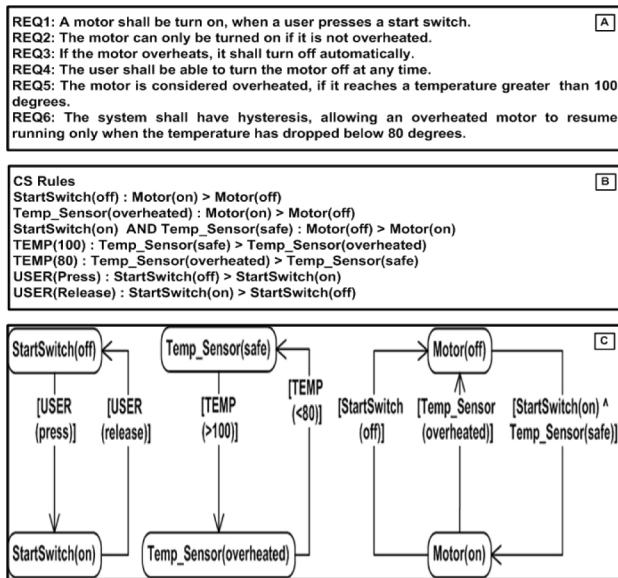


Figure 1. An example of CCM [C], the source requirements [A], and the rules that derived from the requirements [B]

Component State Transitions

These are the defined transitions between the component states (CS) of a given component. In other words, transition between the CS of different components is not allowed.

Transition Causes

The STDs in a CCM are inter-related by the causal effect that the components have on one another. That is, the state transition of a given component is allowed or disallowed by the present state of another component. The model is primary designed for synchronous systems, where all component transitions occur on a system's given clock cycle. So while the actual cause of a CS transition is the system clock, the transition itself is allowed or disallowed by either the state of another component or by an environmental event. Keeping the clock cycle transparent, we refer to a CS allowed transition as a T-cause, and an environmental event as an E-cause. This allows for the modelling of operational context.

CS Rules

The CCM is rule based. The graphical representation of a CCM is derived from a set of rules of the form: T/E-cause : CS_p → CS_n. That is, a transition from a present component state (CS_p) to a next component state (CS_n) is allowed by either a T-cause or an E-cause. For example, the E-cause USER(pressesSwitch), means the user interacting with system has pressed a switch and caused a transition between two component states. For example, given a switch as a component, we can express the rule: USER(pressesSwitch) : Switch(off) → Switch(on), which defines that the USER has caused the Switch to turn on. Figure 1, section [B] shows the set of CS rules that act as the intermediate step between the set of Natural Language (NL) requirements [A] and the CCM [C]. System causes can be single terms, or an ANDed combination of two or more terms. For example, Switch(on) ^ Temp_Sensor(safe) : Motor(off) → Motor(on), means the motor will transition from an off state to an on state if the

Switch is in an on state AND the Temp_Sensor is in a safe state. The CCM of Figure 1, section [C] has its own usefulness as a requirement model [6]. The CCM has also been used as a means of creating a model checking script [5].

The CCMk View

The CS rules, of Figure 2 [B], can be further translated into a set of Global System State (GSS) rules (Figure 2: GSS Zero Rules, section [C] and GSS Expanded Rules, section [E]). GSS rules map the defined transitions between Global System States (discussed later). GSS Expanded Rules can be used to generated the Labeled Transition System, resembling a Kripke structure, shown in Figure 2, section [F]. This LTS is also known as a CCMk (the k stands for Kripke). The CCMk has been used as a means of analyzing the ONB problem. Some of the lessons learned, compiled in this paper, were derived from examining the CCMk view of the requirements.

Global System States

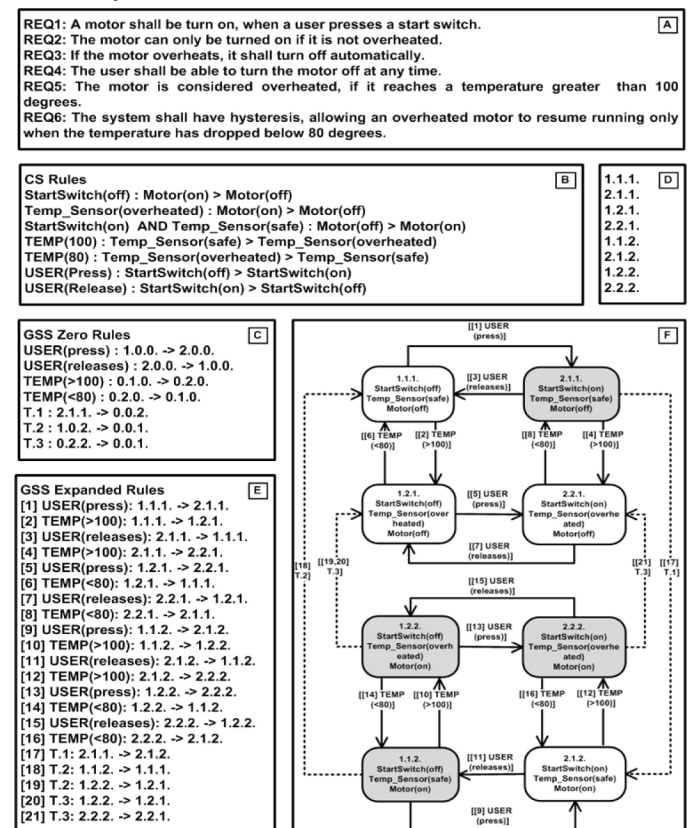


Figure 2. A CCMk [F] representing requirements [A], and the intermediary steps [B] to [E].

The system state (known as a Global System State (GSS)) is a snapshot of the system's state at any given time. The GSS is derived by combining component states that occur concurrently. An example of a GSS is {Switch(off), Temp_Sensor(safe), Motor(off)}, which signifies that when the Switch is off, the Temp_Sensor is concurrently in the safe state, and the Motor is in the off state. The states that comprise the CCMk of Figure 2, section [F] are Global System States. The white represents idle states, whose transition is caused by an environmental cause, such as waiting for the user to do something.