SAND REPORT
SAND2001-3092
Unlimited Release
Printed November 2001

Source Code Assurance Tool: Preliminary Functional Description

Richard L. Craft, Philip L. Campbell, Juan Espinoza Jr.

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/ordering.htm

SAND 2001-3092
Unlimited Release
Printed November 2001

Source Code Assurance Tool: Preliminary Functional Description

Richard L. Craft
Advanced Concepts Group

Philip L. Campbell
Networked Systems Survivability and Assurance

Juan Espinoza Jr.
Cryptography and Information Systems Surety

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0785

Abstract

This report provides a preliminary functional description of a novel software application, the Source Code Assurance Tool, which would assist a system analyst in the software assessment process. An overview is given of the tool's functionality and design; and how the analyst would use it to assess a body of source code. This work was done as part of a Laboratory Directed Research and Development project.

Contents

Source Code Assurance Tool: Preliminary Functional Description

1. TOOL OVERVIEW

It has been said that information system security is really information *system* security – the point being that in the assessment of an information system, it is never enough to focus only on the bits and bytes that flow through the system. Processes external to the information system, human machine interactions, the information system's operating location, the lifecycle of its components, and a range of other concerns must all be addressed in order to assure that a system that incorporates information technology is safe, secure, and reliable.

Because any system of reasonable size can contain a range of technologies, the assessment of such a system typically requires a team of analysts who each bring to the table a unique body of expertise. For efficiency sake, responsibility for various assessment activities is often parceled out among the team members. One member has responsibility for the mechanical aspects of the system, another for physical security issues, a third addresses the software, a fourth analyzes the networking, another considers electrical engineering problems, and yet another handles system issues. While this arrangement is efficient from a project management viewpoint, it introduces problems in the assessment of the system. In a sense, this arrangement encourages each analyst to take a parochial view of the system. While each may analyst may understand his own portion of the system very well, he may not understand how behaviors associated with his portion of the system play together with the rest of the system to deliver undesirable outcomes. For this reason, the analyst must typically ask two types of questions of the other analysts on the team:

- What happens if my portion of the system delivers this kind of event to your portion of the system?
- How could your part of the system deliver this kind of event to my part?

As practiced today, assessments of this sort are typically fragmented and often performed manually. Where automated tools do exist to support a specific aspect of assessment, these tools almost always operate in isolation from one another. What would help is some sort of unifying framework that permits the tools and products of different analysts to be integrated into a common assessment environment. Showing how this can be done is one goal of the Source Code Assurance Tool LDRD project.

1.1 Problems in Source Code Assessment

In the development of high consequence systems, one of the perennially difficult problems is the assurance of software used in these systems. Achieving this assurance invariably rests on human inspection and testing of the software[1]. This process is extremely labor-intensive and, therefore, can be

1 While much research has gone into the use of formal methods to design in assurance, the methods have yet to gain widespread acceptance or to be proven on large-scale engineering projects. Even where they are used (e.g., in the design of security kernels for high security computers, the assurance that the implementation matches the formally-proven design is based on human inspection.

time-consuming and expensive. Given this, safety- and security-critical software projects are often forced into one of two unacceptable outcomes – to slip delivery dates to finish manual inspections or to deliver code that has not been full assessed[2]. The quality of the assessment is also highly dependent on the analyst. Analyst biases and the sheer volume of things to be considered in an assessment can lead to critical problems being overlooked by the analyst. For these reasons, a tool that increases the human analyst's level of performance in software assessment – both in terms of time invested and accuracy – would be of significant benefit.

Effective software assessment in these systems requires that the analyst take a holistic view of the software system and not just focus on an assessment of the software itself. In addition to determining if there are weaknesses/vulnerabilities within the software that could lead to system failure or compromise, the analyst must also ask:

- Can software fail due to vulnerabilities in the platform (computing device and operating system) on which the software runs?
- Can interactions between the system's hardware elements (other than the computing platform) and software lead to failure of the system?

To answer these questions, the analyst needs to understand the various causal relationships that exist (a) within the software, (b) between the software and the computing platform, and (c) between the software and the rest of the system. The analyst must also know the various failure mechanisms / vulnerabilities that exist in the computing platform.

These two tasks each contribute in their own way to the time-intensiveness of assessment. First, identifying the causal relationships within a system is currently a manual process. While tools are available within *integrated development environments* that help the analyst browse the software, they are usually limited in their capabilities. The analyst typically ends up tracing by hand through the series of function calls and equations that contribute to the state of a variable in question or that depend on this variable. Once these causal chains are identified, it is then up to the analyst to decide whether or not given undesirable states can be reached via those chains. While this latter task requires intelligence, the former is essentially mechanistic and, if automated, would significantly reduce the time involved in this portion of the assessment process.

The second source of time-intensiveness is the diffuse nature of the knowledge base regarding the vulnerabilities of various computing platforms. Rather than being centralized so as to be readily usable by the analyst, the vulnerabilities are typically scattered in a divergent set of repositories, many of which may not be known to the analyst. For this reason, the analyst may invest significant amounts of time simply tracking down the vulnerabilities on the Internet, on bulletin boards, in magazines and newsletters, and in a range of other locations. Even when a report regarding a weakness or vulnerability is found, the analyst may have no way of assessing the accuracy of the report; therefore, the analyst may need to take time to verify the report. If a complete knowledge base of validated attacks were available at the analyst's fingertips, a significant amount of time spent in the assessment process would be eliminated.

Finally, when the analyst studies a system to identify the causal relationships with the system, the map of the system that develops remains within the analyst's brain. It is never documented explicitly

[2] Two years ago, the Food and Drug Administration, realizing that defective software in medical devices could threaten human lives, considered establishing quality requirements on medical software. The FDA backed off this position when the implications of human software inspection became clear. Similar time and money issues have lead the National Computer Security Center (the organization within the National Security Agency responsible for assessing high security computing products) to ease its accreditation criteria for lower assurance security systems.

and, therefore, cannot be examined readily by other analysts. Because of this, it is not clear to an outside observer whether or not the analyst has considered all of the causal relationships in the system. Similarly, when the vulnerability databases that an analyst uses in assessing the weaknesses of a computing platform remain in the analyst's head, one can never be sure whether the analyst does not discuss specific vulnerabilities because they were considered unimportant or because the analyst did not know them.

1.2 The Tool's Purpose

Because of the problems described above, the Source Code Assurance Tool is being developed. The goal of this development effort is to produce a tool that makes the analyst more effective in software assessments, both in terms of the time that it takes to deliver a product and in terms of the quality of the product. To this end, the tool will:

- Automate the mechanistic aspects of mapping the system's causal relationships.
- Put at the analyst's fingertips a knowledge base capable of documenting the vulnerabilities of a wide range of platforms.
- Enable the analyst to model the software and non-software elements of the system in a common way that supports *integrated* or *whole system* analysis using standard mechanisms such as fault trees and event trees.
- Provide automated coverage analysis to ensure that the analyst addresses all parts of the system in the assessment.

Using this tool, an analyst will be able to explore the system in various ways:

- The analyst selects a software variable and asks, "If this variable is changed at this point in the program, what is the effect on the rest of the software? On the system?"
- The analyst asks, "What things in the software or system could cause this variable to assume a specific value at a particular point in the program?"
- The analyst investigates how known vulnerabilities in the computing platform affect the rest of the system.
- The analyst then determines which behavioral aspects of a system or software are dependent on those parts of the computing platform known to be vulnerable.

1.3 How the Tool Will Operate

In order to understand how the tool will work, first consider Figure 1-1. The point of this picture is that every set of source code that must be assessed is part of a larger system. While assessing the source code itself is important, looking at only the source code may not uncover all of the surety issues associated with it. For example, even if a body of source code is rendered flawlessly, its dependence on an insecure operating system may leave the code vulnerable to manipulation by an adversary. Similarly, seemingly independent sections of the source code may be connected together in undesirable ways by system components external to the software (a feedback channel provided by a piece of hardware that is both monitored and controlled by the software is one example).

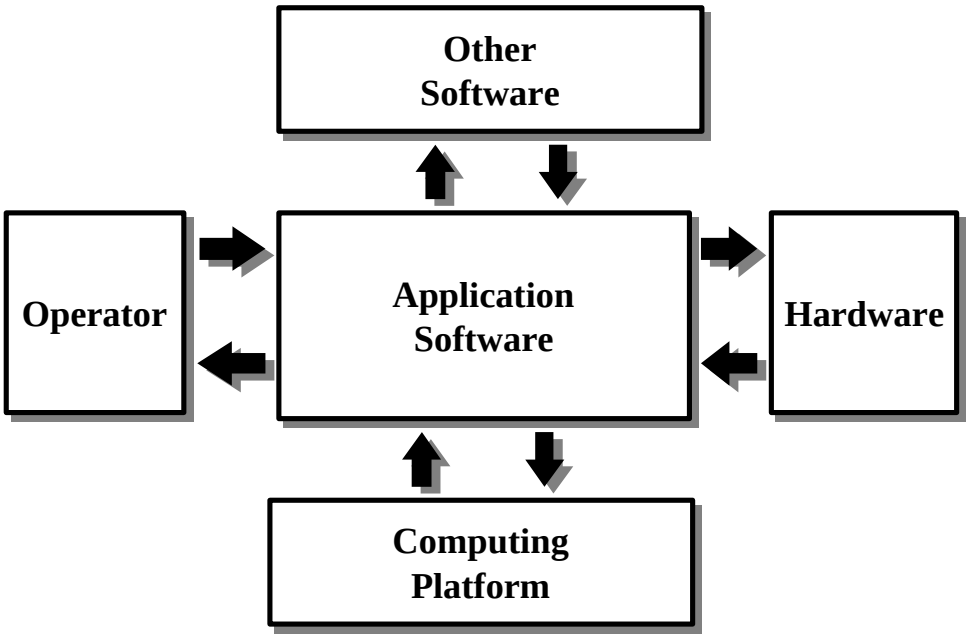

Figure 1-1. The Source Code's Context

To perform a full assessment of the source code using SAT, an analyst first builds a full model of the system to which the source code belongs. This model consists of two main parts: a model of the source codes itself and a model of the components with which the software interacts.

To produce the source code model, the analyst uses a suite of tools, as shown in Figure 1-2. The first tool identifies the organization of the software being assessed. For procedural code, this entails identifying lines of code, procedures and functions, files, programs, *etc.* and the relationships between these entities. For object-oriented implementations the tool extracts lines of code, methods, objects, processes, *etc.* and identifies how these relate to one another. The second tool creates a dependence graph representation of the source code. This portion of the model allows the analyst to *slice* the software portion of the system model. The third tool identifies those points at which the source code interacts with the world external to the code. This includes both the ways the code acts on external entities and the way it is acted upon. This makes it possible for the source code to be analyzed in the context of the entire system to which it belongs.

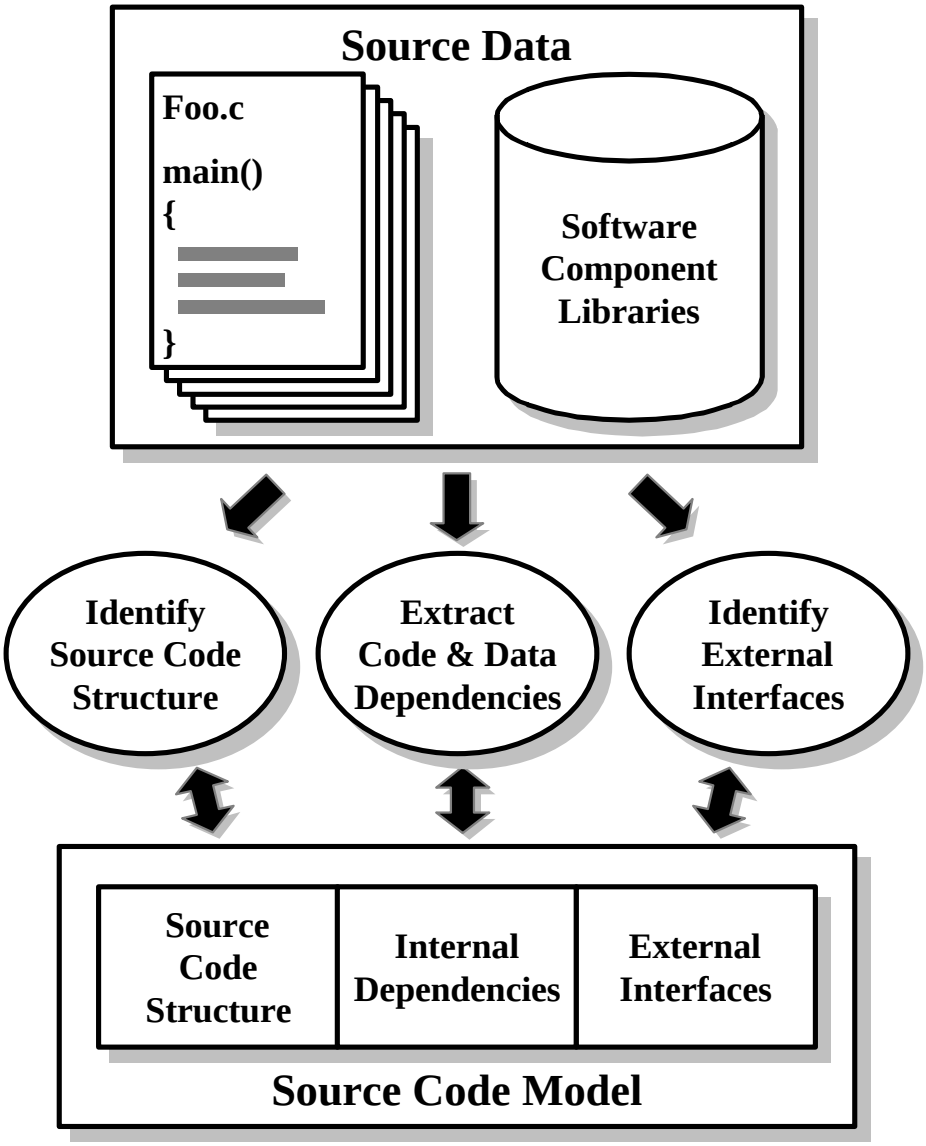

Figure 1-2. The Creation of the Source Code Model

In modeling the non-source code portion of the system being assessed, the analyst uses modeling techniques borrowed from the world of object-oriented analysis (OOA). Components in this portion of the model are treated as independent, self-contained entities that interact with one another through the exchange of data, material, or energy. In this context, elements that make up the structural portion of the source code model are also treated as components of the system model (Figure 1-3). Depending on its operating states, any given entity in the system model transforms a given set of input flows into a set of output flows. Similarly, if an entity is in a given operating state, the right set of input flows can drive the entity to a new operating state. Figure 1-4 depicts this approach to creating the non-software model.

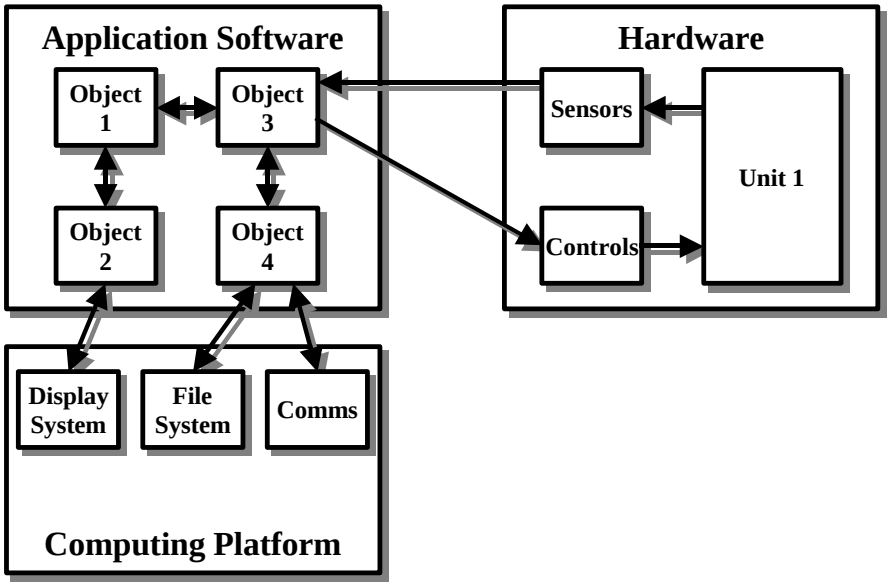

Figure 1-3. A Example System Model

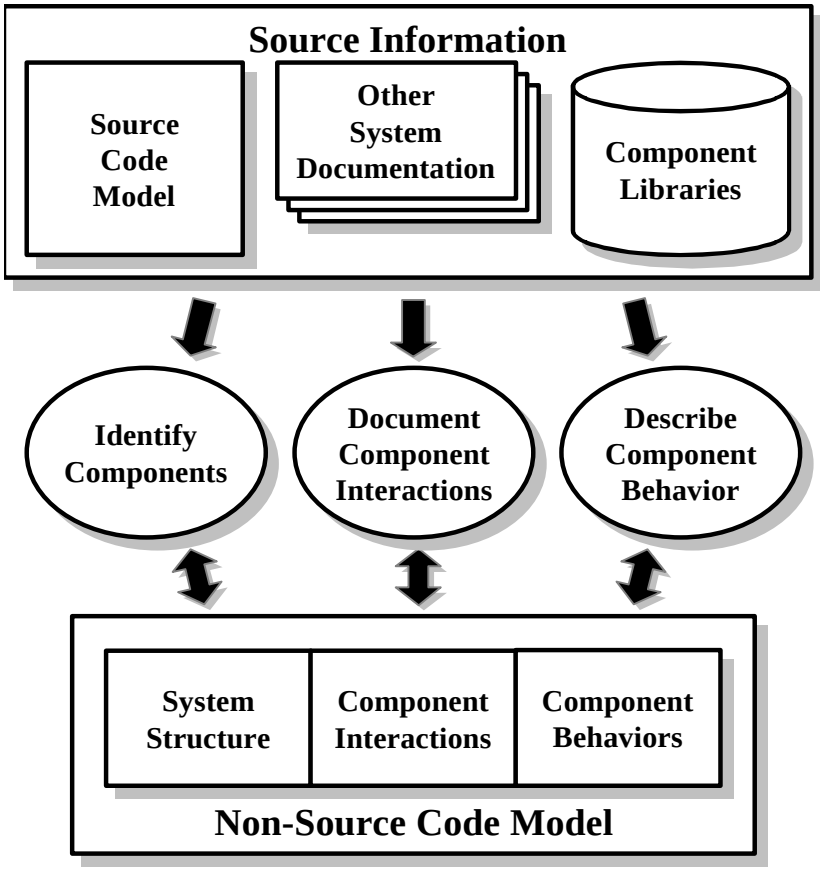

Figure 1-4. The Creation of the Non-Source Code Model

Once the source code and non-source code models have been created, the final step in the creation of the system model is to adorn the model with information regarding various component vulnerabilities. This is done using the process shown in Figure 1-5. Based on knowledge of the specific technologies from which the system is built (this knowledge is documented in the structural portion of the system model), the vulnerability tool augments the base system model. This augmentation consists of adding new components to the model or extending existing component models to reflect vulnerable behaviors.

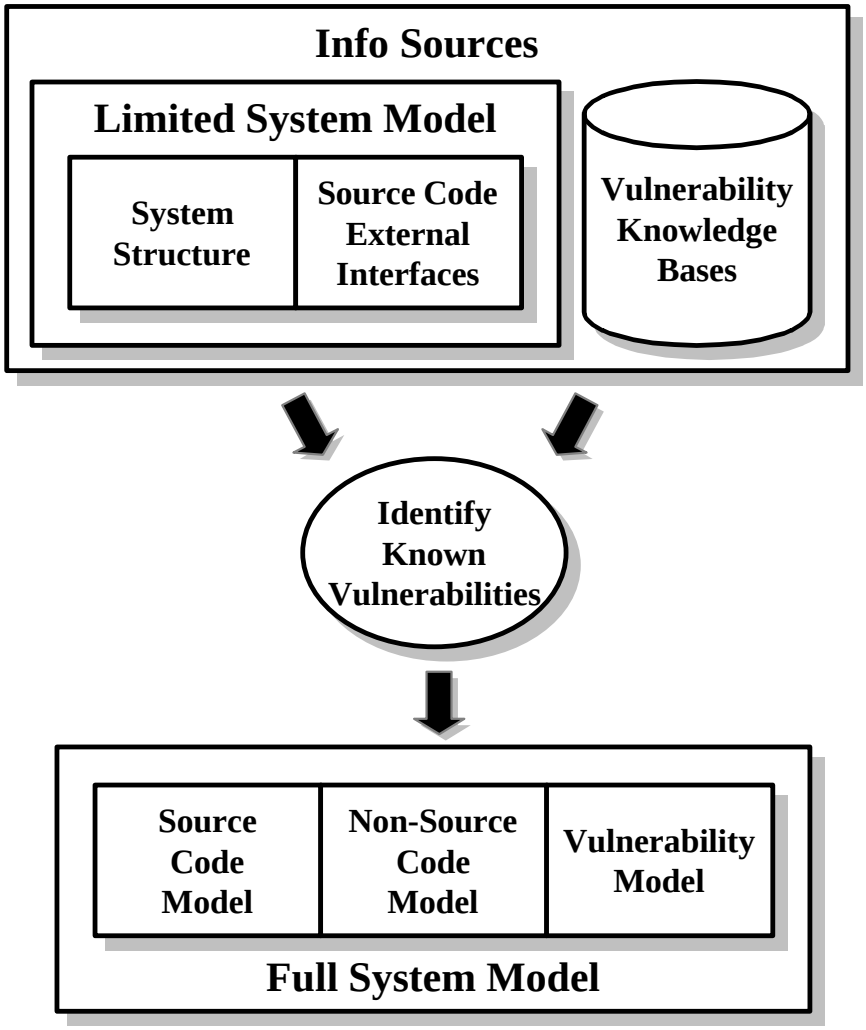

Figure 1-5. The Addition of Vulnerability Information to the System Model

Given the full system model, the analyst assesses the system using techniques borrowed from *program slicing* (a technique related to compilation that has been in existence for about 15 years). If the analyst wishes to determine how a given undesirable outcome could be achieved, then the analyst identifies the point in the system identified with the outcome and executes a *backward slice* of the system. If the analyst wishes to determine how the system responds to a given initiating event, then the analyst identifies the point in the system associated with the event and extracts a *forward slice*

from the system. If the analyst wishes to determine whether any dependencies exist between a given initiating event and a given undesirable outcome, the analyst selects the associated points in the system and then creates a *chop* that depicts the dependency relationships.

To assess the system using backward slicing or fault tree analysis, the analyst first uses the system model to guide the elicitation of *surety objectives*. In SAT, a surety objective is a statement that is attached to a specific point in the system model that identifies something that must not happen at that point in the system. In describing each such objective, the analyst identifies not only the undesirable occurrence, but also:

- The stakeholders who advocate this objective,
- Whether or not it represents a root objective (*i.e.*, one for which the stakeholders cannot provide a justification other than the fact that it is important), and
- How important this objective is relative to all of the other objectives.

Figure 1-6 depicts the process by which these objectives are set.

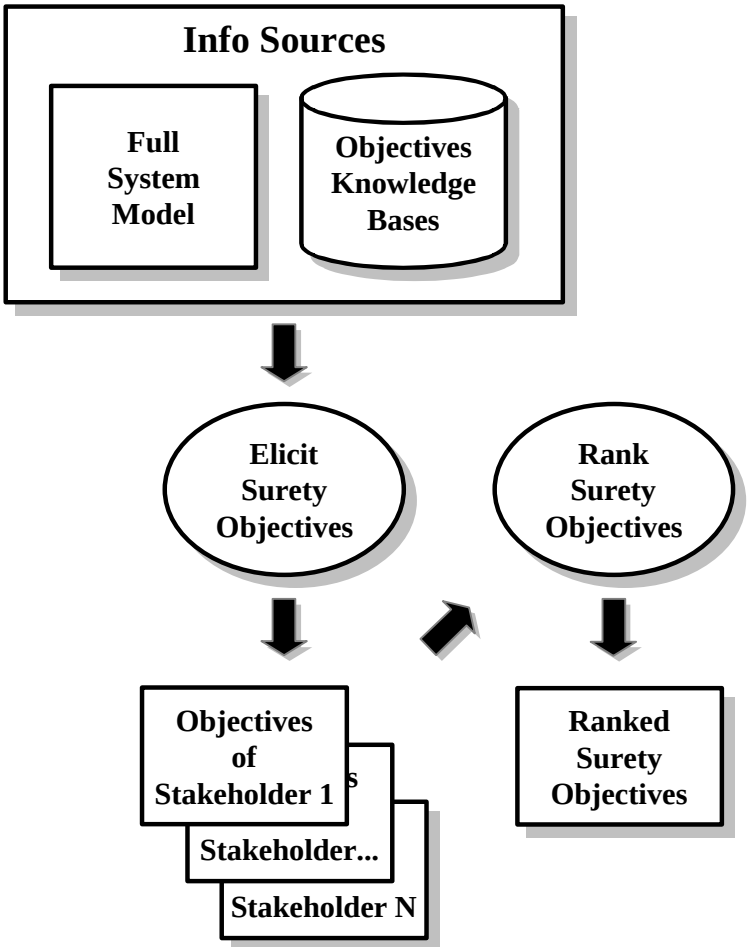

Figure 1-6. The Elicitation of Surety Objectives

Once the surety objectives are set, the analyst then executes a fault tree assessment against each of the objectives. This is an interactive process in which both the tool and the analyst play a role. Using the system model, SAT can indicate to the analyst which node in the model needs to be assessed next (based on the causality knowledge built into the model). At the same time, it is the analyst's job to nominate the things that can go wrong at the selected point in the model. As the analyst identifies these potential faults, the tool associates these faults both with fault tree being built and with the part of the system to which the faults refer. In this way, the construction of other trees brings the analyst once again to these previously assessed nodes; the analyst is able to make use of earlier work by noting the relevance of selected faults to the new trees.

Besides saving analyst time by permitting reuse of earlier assessments, this approach also makes it possible for the analyst to view assessment results in a number of different ways. For example, questions like "To which undesirable outcomes does a given fault contribute?" or "Which fault contributes to more undesirable outcomes than any other?" Since these faults are associated with components in the system model, it becomes possible to ask similar questions, such as "Which components contributes the greatest aggregate risk?" and "Which faults are exploitable by adversaries and which are rooted in reliability problems associated with specific components?" These elements that support this process of creating fault trees and ranking outcomes are shown in Figure 1-7.

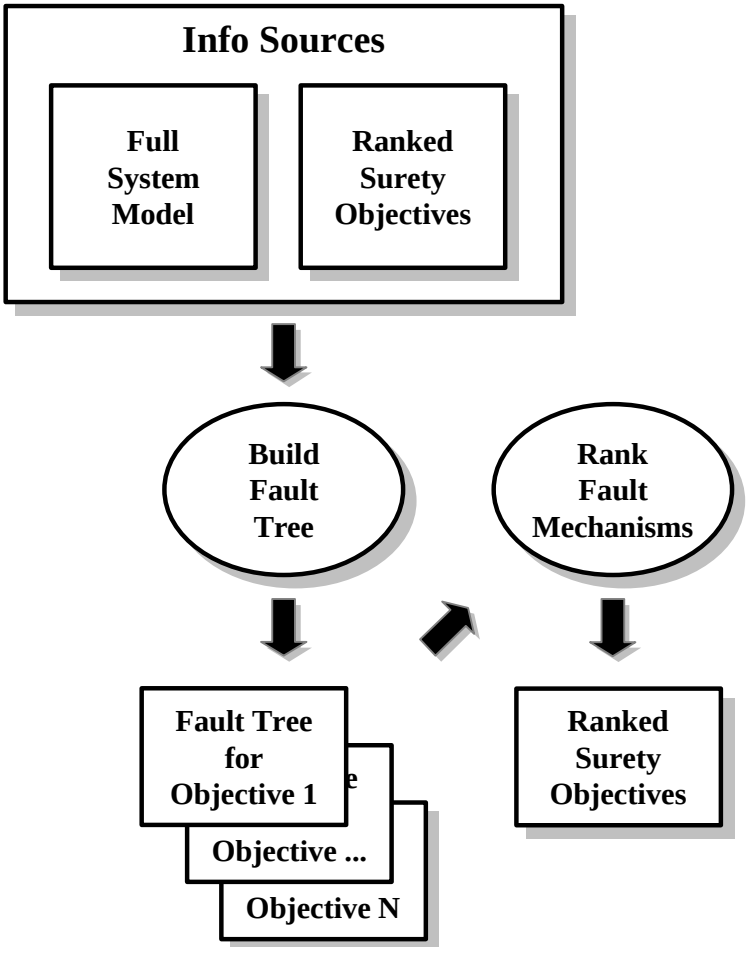

Figure 1-7. Assessing the System Using Fault Trees

While fault tree driven assessment is a common approach (because of its highly focused nature), SAT also supports assessments based on *forward slicing*. One typical assessment of this sort is event tree analysis (ETA). In this form of assessment, the analyst selects one or more initiating events and then determines how the system responds under this stimulus. This kind of analysis is useful for determining whether anything bad can happen to the system given that the system receives a particular input.

As with fault tree analysis, the tool's role in ETA is to guide the analyst through the chains of dependencies that fan out from the point in the system at which the initiating event can occur. The analyst's role is to consider whether various branches are possible (*i.e.*, at points in which the behavior of the system might follow several paths, the analyst determines which are possible and which are not). As the analyst identifies these branch states, SAT annotates the system model with these branch states so that they can be reused in subsequent analyses. Any given behavioral path that the system may follow terminates when the SAT reaches the boundary of the system model or when a part of the system model annotated with an undesirable outcome is reached. As with fault tree analysis, once the analyst has completed construction of the fault models, resulting models can be queried in various ways. For example,

- To which undesirable outcomes does a given initiating event contribute?
- Which initiating event contributes to the most undesirable outcomes?
- Which initiating events contribute to a given undesirable outcome?

1.4 The Tool's Implementation

Given this description of what the tool is to do, the next set of questions to address concern the nature of the tools implementation. In particular,

- On what platforms will the tool operate?
- What is the architecture of the tool?
- How will the tool be delivered (*e.g.*, monolithic application, web-based, *etc.*)?

As described in the introduction to this section, one goal in the development of the SAT tool is to permit team-based assessments to be conducted in an integrated fashion. SAT's approach to realizing this is to employ a unified model of the system from which each of the analysts on the team can derive their own assessment models. In this approach, each analyst, working at his own workstation, can edit and analyze the model that is stored on a computer somewhere on the network (see Figure 1-8).

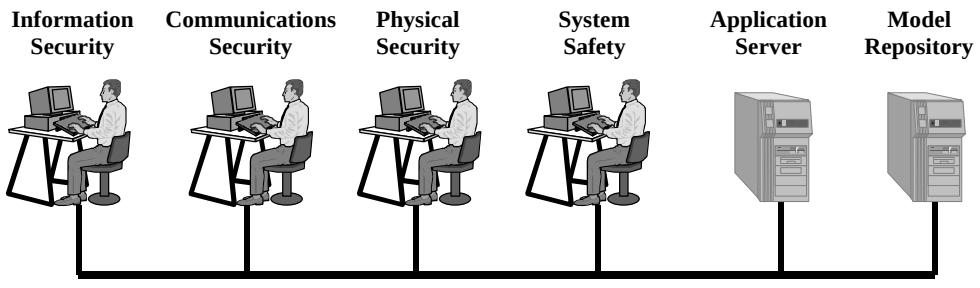

Figure 1-8. The Tool Suite With a Distributed Architecture

From an architecture viewpoint, the goal of this tool suite is openness. It must be possible for surety engineering researchers to add more tools to the tool suite and also to extend the content of the tool suite's information base. To realize this, the tool will employ a layered architecture as shown in Figure 1-9. In this design, the mechanisms used to store the data and the means by which the analyst interacts with the data are separated from one another and from the logic used to deliver the functionality of the tools that populate the suite. This approach is not too different from existing tool suites in the engineering world that permit, for example, an engineer to model a system, to run various analyses on the model, and to then view the results of these analyses in different ways.

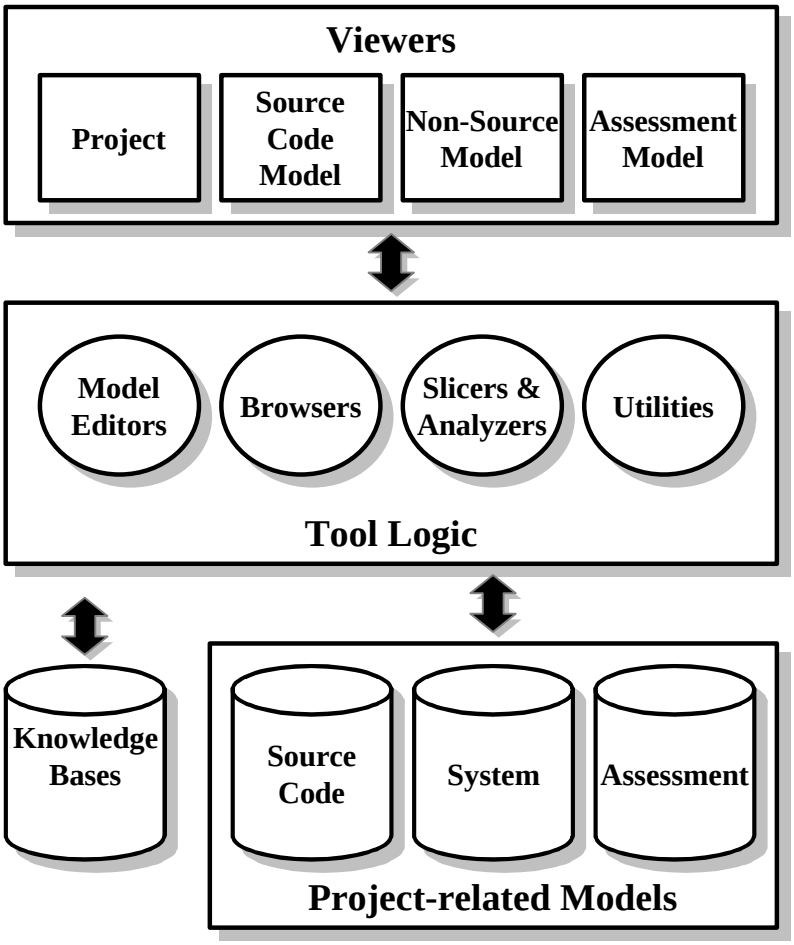

Figure 1-9. Tiered Design of the Tool Suite

Finally, the suite of tools that constitute SAT will initially be delivered on a single platform with each of the tools in the suite being rendered as a Windows application. While the long-term goal is to develop a set of capabilities that all understand the same model base, the reality of using Commercial-of-the-shelf (COTS) tools for part of the work will drive the use of filters that make it possible to import and export data between the SAT model base and various other applications (such as the Rose or Paradigm Plus CASE tools or the Visual Studio programming environment). Given the layered design discussed in the previous paragraph, it should also be possible, if desired, to replace Window-based viewers with web-based applets.

2. USING THE SOURCE CODE ASSURANCE TOOL

Whereas the last chapter talked to SAT in general terms, the purpose of this chapter describes in detail how the analyst uses the tool. The goal of these descriptions is to provide the use cases that drive the detailed design of the tool suite.

2.1 Managing a Project

The starting point for any analysis is the creation of a **project**, which is conceptually a container for all of the work products associated with a given analysis. Components of the project are:

- The **source code** that is being assessed,
- The **source code model** that is derived from the source code,
- The **object model** that documents the source code's context,
- The **vulnerability extensions** to these models,
- The **system surety objectives**,
- The **analysis models**,
- The **ranked findings**, and
- The various **diagrams** created by the analyst to provide views onto these models.

In addition to storing the data listed above, the project also stores the entire user preferences established during the course of the assessment. While viewed by the analyst as a single document, in practice, the project will be a number of files that are all accessed for the purpose of the assessment.

2.1.1 Creating a New Project

Requirements:

	Release				
	1	2	3	4	
R1	X				When a user starts the SAT tool, the tool shall present the user will a new (empty) project.
R2	X				If the analyst has already started the tool and then chooses to create a new project, the tool shall present the user with a new (empty) project and shall not close the currently open project.
R3	X				A user shall be able to view and edit the properties of project.

Questions/Issues:

- None

2.1.1.1 Sample Screen

When an analyst starts a new project, he is presented with a screen similar to the one shown in Figure 2-1.

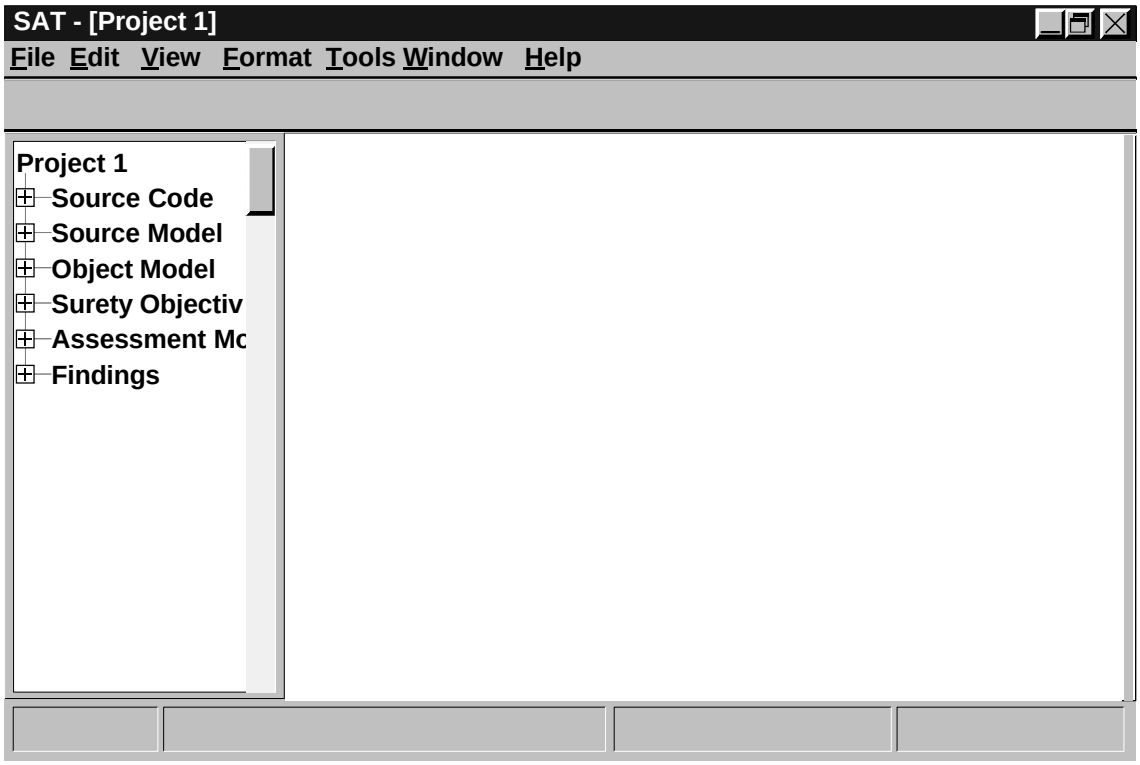

Figure 2-1. Creating a New Project

2.1.2 Opening an Existing Project

Requirements:

	Release				
	1	2	3	4	
R1	X				The user shall be able open an existing project that may be stored in the current directory, in another directory on the same drive, in a directory on a different drive on the same computer, or on a drive located on a different computer altogether.
R2	X				The tool shall permit multiple diagrams to be open simultaneously. Which diagrams are displayed, how they are configured, and how the overall collection of diagrams is arranged on the screen will be referred to as the project's workspace. When opening an existing project, the tool need not restore the workspace to its state when closed.

Questions/Issues:

- When an existing project is opened, what does the user see? Is there some context diagram that serves as the starting point to indicate to the user that the project has been loaded?

2.1.3 Saving a Project

Requirements:

	Release				
	1	2	3	4	
R1	X				At any point during work on a project, a user can save the changes made to the project since the last time that it was saved (or, if this is a new project, since it was opened).
R1.1	X				If the project is an existing project, the tool simply updates the project information in the same location as that from which it was originally read.
R1.2	X				If the tool is a new project, the user shall be asked to supply needed project name and to specify where the project is to be stored.
R2	X				At any point after the time that a project has been saved, a user shall be able to save the project under a new name. When this is done, the tool shall create a copy of the project, assign it the new name, and close the old project, so that the newly named project becomes the current project.

Questions/Issues:

- Do we want to allow for maintenance of backup copies? What about versioning or configuration management?
- Do we allow renaming?
- How about saving diagrams as html web sites?

2.1.4 Deleting a Project

Requirements:

	Release				
	1	2	3	4	
R1	X				The user shall be able to delete any project that has been created by the tool. The result of this action is the removal from the storage location of all information specific to the project being deleted.

Questions/Issues:

- None

2.1.5 Closing the Tool

Requirements:

	Release				
	1	2	3	4	
R1	X				The user shall be able to close the SAT tool. If any open projects contained unsaved changes, the user shall be given the opportunity to save these changes or to discard them. If they are to be saved, then the requirements described above for saving a project shall apply.

Questions/Issues:

- None

2.2 Identifying the Source Code to be Assessed

Assessment is only one part of a larger cycle in the assurance of a software-based system. Once problems are identified, the analyst's task is to identify alternative approaches to safeguarding the system. In doing so, the analyst creates different versions of the system, each typically addressed at some particular set of protection goals. Once created, these systems must be assessed and then compared with each and with the original system in order to determine what gains in assurance have been realized and at what cost. For this reason, the SAT tool will support the ability to manage multiple system configurations within the context of a single project.

For any given the configuration, SAT allows the analyst to identify one or more *program units* as being part of the configuration. In the context of this document, the term *program unit* is used to connote a collection of source code files that are compiled together to create a monolithic program. By allowing the analyst to identify multiple program units as belonging to a given configuration, the system permits the modeling of systems made up of multiple cooperating programs. These programs may all reside on a single computer or may distributed across a network of computers.

Finally, as one of the fixes that can be applied to software-based systems in modification of the software, it is necessary for the SAT tool to employ some sort of configuration management so that assessments results obtained by the tool can be repeated once the source code that drove the assessments has been modified.

2.2.1 Identifying System Configurations

Requirements:

	Release				
	1	2	3	4	
R1	X				A user shall be able to add to a project one or more configurations. In doing so, the user shall assign a name to each configuration.
R2	X				A user shall be able to rename a configuration.
R3	X				A user shall be able delete one or more configurations from a project
R4				X	A user shall be able to copy one or more configurations from a given project. The effect is to leave the configuration(s) in the project and to place the configuration contents in SAT's paste buffer.
R5				X	A user shall be able to cut one or more configurations from a given project. The effect is to remove the configuration(s) from the configuration and to place the configuration contents in SAT's paste buffer.
R6				X	A user shall be able to paste one or more configurations into a given project. The effect is to insert the configuration(s) in the project and to leave the configuration contents in SAT's paste buffer.

Questions/Issues:

- The point in being able to copy, cut, and paste configurations is to provide some mechanisms for a user to move one part of a given project into another project without having to import the entire project.

2.2.2 Identifying Program Units in a System Configuration

Requirements:

	Release				
	1	2	3	4	
R1	X				A user shall be able to add to a configuration one or more program units. In doing so, the user shall assign a name to each program unit.
R2	X				A user shall be able to rename a program unit.
R3	X				A user shall be able delete one or more program units from a configuration.
R4				X	A user shall be able to copy one or more program units from a given configuration. The effect is to leave the program unit(s) in the configuration and to place the program unit contents in SAT's paste buffer.
R5				X	A user shall be able to cut one or more program units from a given configuration. The effect is to remove the program unit(s) from the configuration and to place the program unit contents in SAT's paste buffer.
R6				X	A user shall be able to paste one or more program units into a given configuration. The effect is to insert the program unit(s) in the configuration and to leave the program unit contents in SAT's paste buffer.

Questions/Issues:

None

2.2.3 Identifying Source Code Files in a Program Unit

Requirements:

	Release				
	1	2	3	4	
R1	X				A user shall be able to add a program unit, one or more source code files that are required to create the program unit.
R2	X				A user shall be able to delete source code files from a program unit.
R3				X	A user shall be able to copy one or more source code files from a given program unit. The effect is to leave the source code files in the program unit and to place the source code file contents in SAT's paste buffer.
R4				X	A user shall be able to cut source code files from a given program unit. The effect is to remove the source code files from the program unit and to place the source code file contents in SAT's paste buffer.
R5				X	A user shall be able to paste source code files into a given program unit. The effect is to insert the source code files into the program unit and to leave the source code file contents in SAT's paste buffer.

Questions/Issues:

- As an assessment is based on a particular system configuration, it is important that the source code (and object models) that makes up the system being assessed not be altered. To do so, would invalidate the assessment results.
- In pasting source code files copied or pasted from one program unit into another, do we want to give the user the ability to specify whether the paste should be either create a copy or create a link to an existing copy? The issue is whether the new copy can be modified without regard to the source from which it was taken or whether when one changes all copies need to change.
- A nice feature of the tool would be the ability to open and select one or more files from a Visual Studio project or to import a project directly.

2.3 Building the Source Code Model

The source code model is a product derived from a given program unit. From a SAT user's perspective, the source code model documents the following about the program unit:

- Structure
- Internal dependencies
- External interfaces

SAT will be used for both procedural and object-oriented code. Accordingly, it will be able to model the structure of both kinds of software. For procedural code, this will include identifying a program unit's modules, functions, and lines of code. For object-oriented code this will entail the model of objects, methods, lines of code, *etc.*

In modeling internal dependencies, SAT will initially identify code and data dependencies that exist between source code lines. In time (hopefully), it will support modeling of variable-level interdependencies.

Finally, for each of the elements identified in the structural model of the source code, SAT will assist the analyst in identifying whether or not the element contains any interfaces that extend beyond the program unit that contains the element. For those external interfaces that do exist, SAT permits the user to document the nature of the interface in terms of various qualifiers (*e.g.*, operating system call or external messaging).

2.3.1 Extracting the Code's Structure

Requirements:

	Release				
	1	2	3	4	
R1		X			A user shall be able to extract the structural model of a program unit rendered in procedurally oriented C++ code. This structural model will identify the structural components of the code and the relationships that exist between these components (*i.e.*, parent-child or sibling)[3].
R2			X		A user shall be able to extract the structural model of a program unit rendered in object-oriented C++ code. This structural model will identify the structural components of the code and the relationships that exist between these components (*i.e.*, parent-child or sibling).
R3		X			In developing the structural model, SAT shall automatically assign a unique name to each component identified.
R4		X			A user shall be permitted to create a new name to replace the one assigned to a component by SAT.
R5		X			A user shall be able to edit a user-assigned component name.

[3] It seems that the refactoring literature might have something to say about this subject area. In particular, one of the ideas in refactoring, I believe, is that the elements that make up a piece of code that is rendered in one way can be restructured into an altogether different program architecture without changing the functionality delivered by the code.

R6		X			A user shall be able to remove a user-assigned component name. In this event, the name reverts to the unique identifier originally assigned by SAT.

Questions/Issues:

- When maintaining multiple configurations (or when the code from which one configuration was derived), how does the source code modeling utility know what has changed and what has not since the last time that the model of the subject source code was built?
- Should one of SAT's features be the ability to visually *diff* two models?
- How do we model code like C++ allows where part of the code may be rendered procedurally while the rest of it is rendered as object code?
- Can a user copy/cut and paste components of the source code model? What does this mean (clearly, this is visual programming of a sort) and how would it benefit a surety engineer?

2.3.2 Identifying Internal Dependencies

Requirements:

	Release				
	1	2	3	4	
R1		X			A user shall be able to extract the data and control dependencies of a program unit rendered in procedurally oriented C++ code.
R2			X		A user shall be able to extract data and control dependencies of a program unit rendered in object-oriented C++ code.
R3		X	X		In extracting these dependencies, the user shall be able to associate the data and control flows modeled by the dependencies with the structural components extracted in the structural modeling of the source code (see section 2.3.1)[4].

Questions/Issues:

- None

[4] The goal here is to support the creation of interaction diagrams based on the elements in the source code model.

2.3.3 Identifying External Dependencies

In modeling source code, SAT assumes that a program unit can interface to:

- The computing platforms on which the source code runs.
- Human operators.
- Other program units that exist either on the same computing platform or on other platforms.
- Other hardware (typically electromechanical systems that are monitored or controlled by the program unit).

2.3.3.1 Dependencies on the Computing Platform

Requirements:

	Release				
	1	2	3	4	
R1		X			For a program unit rendered in C++ code and targeted for the Windows environment, a user shall be able to identify all of the places in which the code interacts with the underlying operating system and shall be able to characterize the interaction to the extent that the source code allows.
R2			X		For a program unit rendered in C++ code and targeted for an embedded environment, a user shall be able to identify all of the places in which the code interacts with the underlying operating system or with the hardware directly and shall be able to characterize the interaction to the extent that the source code allows.
R3		X			The tool shall permit the modeling of the Windows operating system as a collection of cooperating components.
R4		X			The tool shall support the modeling of a hardware platform and any supporting operating system as a collection of cooperating components.
R5		X	X		In extracting these interactions, the user shall be able to associate these flows with the structural components extracted in the structural modeling of the source code (see section 2.3.1)[5] and with the non-source code model structural components (see section **2.4.1**) created to model the computing platform.

Questions/Issues:

- We need to address the issues associated with the logical versus physical rendering of software so that we can handle issues associated with things like overrunning array boundaries.

[5] The goal here is to support the creation of interaction diagrams based on the elements in the source code model.

2.3.3.2 Dependencies on Operators

Requirements:

	Release				
	1	2	3	4	
R1			X		For a program unit rendered in C++ code and that supports operator interactions, a user shall be able to identify all of the places in which the code interacts with an operator.
R2		X			In documenting the interaction, SAT shall allow the user to distinguish between the information flowing between the source code and the user and the mechanism used to realize this flow[6].
R3		X			The tool shall support the modeling of an operator as a collection of co-operating components.
R4		X	X		In extracting these interactions, the user shall be able to associate these flows with the structural components extracted in the structural modeling of the source code (see section 2.3.1)[7] and with the non-source code model structural components (see section **2.4.1**) created to model the operator(s).

Questions/Issues:

- None

2.3.3.3 Dependencies on Other Program Units

Requirements:

	Release				
	1	2	3	4	
R1	X				For a program unit rendered in C++ code and that interacts with other program units located on the same computing platform, a user shall be able to identify all of the places in which the code interacts with the other units and shall be able to characterize these interactions.
R2	X				For a program unit rendered in C++ code and that interacts with other program units located on other computing platforms, a user shall be able to identify all of the places in which the code interacts with the other units and shall be able to characterize these interactions.
R3		X			In documenting these interactions, SAT shall allow the user to distin-

[6] The tool should permit an analyst to document both the fact that the user tells the software to start a given operation and that a given mechanism (*e.g.*, a button or a command line) is used to receive the *start* command and to be able to show how they relate to one another in a hierarchical modeling scheme. The issue is that *start* and the means by which *start* reaches the code that processes the concept *start* will typically be thought of at two different levels of abstraction with the mechanism being hidden in the most abstract system models that only show an operator and the program unit with *start* flowing between them.

[7] The goal here is to support the creation of interaction diagrams based on the elements in the source code model.

					guish between the information flowing between the program units and the mechanism used to realize these flows (*i.e.*, it shall be possible to model both abstract and actual flows and to show how they map to one another).
R4	X				The tool shall support the modeling of other program units as collections of cooperating components (even if the source codes for the program units are not available).
R5	X				In extracting these interactions, the user shall be able to associate these flows with the structural components extracted in the structural modeling of the source code (see section 2.3.1)[8] and with the source code and/or non-source code model structural components (see section 2.3.1 and section **2.4.1**) created to model the other program units.

Questions/Issues:

- None

2.3.3.4 Dependencies on Other Devices

Requirements:

	Release				
	1	2	3	4	
R1		X			For a program unit rendered in C++ code and that interacts with other non-computing devices, a user shall be able to identify all of the places in which the code interacts with the devices and shall be able to characterize these interactions.
R2		X			In documenting these interactions, SAT shall allow the user to distinguish between the information flowing between the computer and the devices and the mechanisms used to realize these flows (*i.e.*, it shall be possible to model both abstract and actual flows and to show how they map to one another).
R3		X			The tool shall support the modeling of other devices as collections of cooperating components.
R4		X			In documenting these interactions, the user shall be able to associate these flows with the structural components extracted in the structural modeling of the source code (see section 2.3.1)[9] and with the non-source code model structural components (see section 2.4.1) created to model the other devices.

Questions/Issues:

- None

[8] The goal here is to support the creation of interaction diagrams based on the elements in the source code model.
[9] The goal here is to support the creation of interaction diagrams based on the elements in the source code model.

2.4 Building the Non-software Model

In the assessment of software-based system, SAT permits a user to assess the software in its context. For this reason, in building a system model, the user models both the software and non-software portions of the system. To model generic non-software components, a SAT user employs techniques borrowed from object oriented modeling. In particular, the user can:

- Develop a model of a component's structure/composition,
- Identify the component's interactions with other system components, and
- Describe the component's macro- and micro- behaviors.

2.4.1 Defining the System Structure

Requirements:

	Release				
	1	2	3	4	
R1	X				A user shall be able to add to a configuration one or more components. In doing so, the user shall assign a name to each component.
R2	X				A user shall be able to define a given component as consisting of one or more components (the goal is to allow hierarchical decomposition). In doing so, the user shall assign a name to each component.
R3	X				A user shall be able to rename a component.
R4	X				A user shall be able to set the type of a component as a simple source or sink for the purposes of allowing flows (see section 2.4.2) to and from other (*i.e.*, non-source and non-sink) components to be identified in the absence of any real knowledge regarding the nature of the sources and sinks with which the component is associated.
R5	X				A user shall be able delete one or more components from a configuration.
R6				X	A user shall be able to copy one or more components from a given configuration. The effect is to leave the component(s) in the configuration and to place the component contents in SAT's paste buffer.
R7				X	A user shall be able to cut one or more components from a given configuration. The effect is to remove the component(s) from the configuration and to place the component contents in SAT's paste buffer.
R8				X	A user shall be able to paste one or more components into a given configuration. The effect is to insert the component(s) in the configuration and to leave the component contents in SAT's paste buffer.

Questions/Issues:

- None

2.4.2 Defining Intercomponent Flows

Requirements:

	Release				
	1	2	3	4	
R1	X				A user shall be able to document the flows of energy, material, or information that occur between two or more components with a given configuration.
R2	X				A user shall be able to define a flow a consisting of one or more flows or flow attributes. A flow attribute is a variable that qualitatively or quantitatively describes some aspect of a flow.
R3				X	A user shall be able to copy one or more flows from a given configuration. The effect is to leave the flow(s) in the configuration and to place the flow contents in SAT's paste buffer.
R4				X	A user shall be able to cut one or more flows from a given configuration. The effect is to remove the flow(s) from the configuration and to place the flow contents in SAT's paste buffer.
R5				X	A user shall be able to paste one or more flows into a given configuration, either as a top-level flow between two components or as a constituent flow of some higher-level flow. The effect is to insert the flow(s) in the configuration and to leave the flow contents in SAT's paste buffer.

Questions/Interactions:

- We might consider using both UML object collaboration views.

2.4.3 Describing Component Behavior

Given the set of input flows that a given component will receive, the user must be able to describe how the component responds to these flows. This will typically be expressed as the generation of new flows and alteration of the component's state.

In SAT, the user develops two types of behavioral models. The macro model captures the states in which the component can exist. This allows the user to document the fact that a given component's response to a set of input flows is not always constant. It can change as a result of previous inputs, the passing of time, and a number of other factors. In modeling the component's macro-behavior, the user identifies the states in which the component can exist and events which cause the component to change from one state to another. The micro model documents the component's response to input flows while in any given state. The focus in the micro-model is on how a component's current state and input flows combine to alter the state and create new output flows.

2.4.3.1 Macro behavior

Requirements:

	Release				
	1	2	3	4	
R1		X			A user shall be able to create one or more state transition diagrams for a given component. In doing so, the user shall assign a name to each diagram.
R2		X			A user shall be able to rename a state transition diagram.
R3				X	A user shall be able to copy one or more state transition diagrams from a given component. The effect is to leave the state transition diagram(s) in the component and to place the state transition diagram contents in SAT's paste buffer.
R4				X	A user shall be able to cut one or more state transition diagrams from a given component. The effect is to remove the state transition diagram(s) from the component and to place the state transition diagram contents in SAT's paste buffer.
R5				X	A user shall be able to paste one or more state transition diagrams into a given component. The effect is to insert the state transition diagram(s) in the component and to leave the state transition diagram contents in SAT's paste buffer.
R6		X			A user shall be able to delete one or more state transition diagrams from a given component. The effect is to permanently remove the selected diagrams(s) and all supporting model elements from the system model.
R7		X			A user shall be able to add one or more states to a given state transition diagram. In doing so, the user shall assign a name to each state.
R8		X			A user shall be able to rename a state in a state transition diagram.
R9				X	A user shall be able to copy one or more states from a given state transition diagram. The effect is to leave the state(s) in the state transition diagram and to place the state contents in SAT's paste buffer.
R10				X	A user shall be able to cut one or more states from a given state transition diagram. The effect is to remove the state(s) from the state transition diagram and to place the state contents in SAT's paste buffer.
R11				X	A user shall be able to paste one or more states into a given state transition diagram. The effect is to insert the state(s) in the state transition diagram and to leave the state contents in SAT's paste buffer.
R12		X			A user shall be able to delete one or more states from a given state transition diagram. The effect is to permanently remove the selected state(s) and all supporting model elements from the system model.
R13		X			A user shall be able to add a transition between any two states in a given state transition diagram.

R14		X			A user shall be able to delete one or more transitions from a given state transition diagram. The effect is to permanently remove the selected transition(s) and all of the supporting model elements from the system model.
R15		X			A user shall be able to add one or more events[10] to a given transition. In doing so, the user shall assign a name to each event.
R16		X			A user shall be able to rename an event associated with a given transition.
R17				X	A user shall be able to copy one or more events from a given transition. The effect is to leave the event(s) in the transition and to place the event contents in SAT's paste buffer.
R18				X	A user shall be able to cut one or more events from a given transition. The effect is to remove the event(s) from the transition and to place the event contents in SAT's paste buffer.
R19				X	A user shall be able to paste one or more events into a given transition. The effect is to insert the event(s) in the transition and to leave the event contents in SAT's paste buffer.
R20		X			A user shall be able to delete one or more events from a given transition. The effect is to permanently remove the selected event(s) and all supporting model elements from the system model.

Questions/Issues:

- None

2.4.3.2 Micro behavior

Requirements:

	Release				
	1	2	3	4	
R1		X			A user shall be able to create a data flow diagram for a given state.
R2				X	A user shall be able to copy a data flow diagram from a given state. The effect is to leave the data flow diagram in the state and to place

[10] Supported events shall include:

- The arrival of a flow,
- The attainment of some state internal to the component (*e.g.*, some component attribute takes on a given internal value),
- The attainment of some global state (*e.g.*, a certain date and time have been reached or a certain amount of time has elapsed since some previous event), and
- Internally generated events (typically used to synchronize different STDs or to cause automatic transition into one state upon completion of activities in another state).
- The only event exported by a component is a flow. In the event that a given component's macro-behavior is modeled with multiple state transition diagrams, the user shall be allowed to model events generated by one state as being consumed by the other state(s).

					the data flow diagram contents in SAT's paste buffer.
R3				X	A user shall be able to cut a data flow diagram from a given state. The effect is to remove the data flow diagram from the state and to place the data flow diagram contents in SAT's paste buffer.
R4				X	A user shall be able to paste a data flow diagram into a given state. The effect is to insert the data flow diagram in the state and to leave the data flow diagram contents in SAT's paste buffer.
R5		X			A user shall be able to delete a data flow diagram from a given state. The effect is to permanently remove the selected diagrams and all supporting model elements from the system model.
R6		X			A user shall be able to add one or more processes to a given data flow diagram. In doing so, the user shall assign a name to the process(es).
R7		X			A user shall be able to rename a process.
R8		X			A user shall be able to define a process in terms of the dependency relationships that exist between its input and output flow attributes.
R9		X			For each such dependency, a user shall be able to assign minimum, nominal, and maximum delays associated with the transformations represented by the dependency.
R10				X	A user shall be able to copy one or more processes from a given data flow diagram. The effect is to leave the process(es) in the data flow diagram and to place the process contents in SAT's paste buffer.
R11				X	A user shall be able to cut one or more processes from a given data flow diagram. The effect is to remove the process(es) from the data flow diagram and to place the process contents in SAT's paste buffer.
R12				X	A user shall be able to paste one or more processes into a given data flow diagram. The effect is to insert the process(es) in the data flow diagram and to leave the process contents in SAT's paste buffer.
R13		X			A user shall be able to delete one or more processes from a given data flow diagram. The effect is to permanently remove the selected process(es) and all supporting model elements from the system model.
R14		X			A user shall be able to add one or more stores to a given data flow diagram. A store is any mechanism within a component that caches energy, material, or information. In adding a store so, the user shall assign a name to the store.
R15		X			A user shall be able to rename a store.
R16		X			A user shall be able to define a store as consisting of one or more store attributes. A store attribute is a variable that qualitatively or quantitatively describes some aspect of the stuff[11] cached in the store.
R17				X	A user shall be able to copy one or more stores from a given data flow diagram. The effect is to leave the store(s) in the data flow diagram

[11] This is the technical term for "information, material, or energy".

					and to place the store contents in SAT's paste buffer.
R18				X	A user shall be able to cut one or more stores from a given data flow diagram. The effect is to remove the store(s) from the data flow diagram and to place the store contents in SAT's paste buffer.
R19				X	A user shall be able to paste one or more stores into a given data flow diagram. The effect is to insert the store(s) in the data flow diagram and to leave the store contents in SAT's paste buffer.
R20		X			A user shall be able to delete one or more stores from a given data flow diagram. The effect is to permanently remove the selected store(s) and all supporting model elements from the system model.
R21		X			A user shall be able to document the flows of energy, material, or information that occur between two or more processes or between processes and stores.
R22		X			A user shall be able to define a flow as consisting of zero or more flows or flow attributes. A flow attribute is a variable that qualitatively or quantitatively describes some aspect of a flow. In cases where zero attributes are assigned to a flow, the flow shall be understood to represent a control flow.
R23				X	A user shall be able to copy one or more flows from a given data flow diagram. The effect is to leave the flow(s) in the data flow diagram and to place the flow contents in SAT's paste buffer.
R24				X	A user shall be able to cut one or more flows from a given data flow diagram. The effect is to remove the flow(s) from the data flow diagram and to place the flow contents in SAT's paste buffer.
R25				X	A user shall be able to paste one or more flows into a given data flow diagram, either as a "top-level flow" between two or more processes or processes and stores or as a constituent flow of some higher level flow. The effect is to insert the flow(s) in the configuration and to leave the flow contents in SAT's paste buffer.

Questions/Issues:

- None

2.4.4 Unifying Submodels

While this paper's description of the process for building models has implied a specific ordering of process steps, in a real system assessment some of these steps will occur in parallel. It is entirely reasonable to expect that different sub-models in a given configuration will be developed independently and then knit together. When this occurs, each sub-model will have its own set of external interfaces (*i.e.*, input and output flows) that its components present to the world outside of the sub-model. The knitting together of these sub-models entails identifying which sub-models interact with one another and identifying which flows in one sub-model are really the same flows shown in another sub-model.

Requirements:

	Release				
	1	2	3	4	
R1			X		A user shall be able to identify that two independently developed components interact with one another
R2			X		A user shall be able to identify which flows associated with interacting components map to one another.
R3			X		When to flows are mapped to one another, the user shall be able to specify how the flows are to be labeled for the purpose of the combined system model. The selected name may be that supplied by either sub-model or a wholly new label supplied by the user.
R4			X		A user shall be able to remove a mapping between flows.

Questions/Issues:

- None

2.4.5 Using Component Libraries

In addition to permitting the assessment of systems by teams, the ability to knit sub-models together also makes it possible to draw on lessons learned in previous assessments. By permitting previously developed sub-models to be stored in libraries and then used wholesale into new assessments, SAT makes it possible to more quickly assemble and evaluate new systems.

Requirements:

	Release				
	1	2	3	4	
R1				X	The user shall have the ability to add to a configuration components selected from component libraries. In doing so, the user incorporates into the configuration elements of the system model and elements of the assessment models.
R2				X	The user shall have the ability to edit the name assigned to an included component.

R3				X	The user shall have the ability to browse and edit the contents of an included component. Since, in doing so, the user may end up destroying some links within the component's model, the tool shall warn the user when this about to occur.
R4				X	If the user edits a component's model, the tool's coverage database shall be updated to reflect the new uncovered branches.

Questions/Issues:

- None

2.5 Modeling Component Vulnerabilities

Once the normal model of the system has been defined, the analyst needs to identify what can go wrong in each of the components that make up a given system configuration. To do this, the analyst uses one set of methods to address source code vulnerabilities and another to address vulnerabilities in the rest of the system.

2.5.1 Modeling Vulnerabilities in the Source Code Model

In modeling source code vulnerabilities, the primary focus is on what could go wrong if variables within the program took on unexpected values or if events occur out of order.

Requirements:

	Release				
	1	2	3	4	
R1			X		For each variable in the source code model, the user shall be able to specify whether the variable represents an enumeration or a range of values.
R2			X		For an enumerated variable, the user shall be able to identify which values are normal and which are abnormal.
R3			X		For range-valued variables, the user shall be able to identify landmark values that bound ranges and to identify whether or not given ranges delineated by these landmark values are normal or abnormal.

Questions/Issues:

- This needs a lot of work before it's ready for prime time. My guess is that there are many software-testing techniques that address this subject.
- I also see this area as being a point at which we can later bridge (as in another project) to problems arising out of software and hardware interactions.

2.5.2 Modeling Vulnerabilities in the Non-software Model

Component vulnerabilities are modeled using the same mechanisms used to model normal component behavior. As a result of modeling vulnerabilities for a given component, the component's submodel can grow with respect to the flows that it accepts or generates and with respect to its behavior.

Requirements:

	Release				
	1	2	3	4	
R1			X		For each attribute in the system model, the user shall be able to specify whether the attribute represents an enumeration or a range of values.

R2			X		For an enumerated attribute, the user shall be able to identify which values are normal and which are abnormal.
R3			X		For range-valued attributes, the user shall be able to identify landmark values that bound ranges and to identify whether or not given ranges delineated by these landmark values are normal or abnormal.
R4			X		A user shall be able to add new flows into and/or out of a component submodel that reflect abnormal flows to which the component responds or that it can generate.
R5			X		A user shall be able to add or remove behavioral model elements to describe abnormal component behavior. Included in the elements that can be added or removed are entire state transition diagrams, portions of state transition diagrams, and portions of data flow diagrams and their associated process descriptions.

Questions/Issues:

- This needs a lot of work before it's ready for prime time.
- Need words here about how the library of component vulnerabilities plays into system modeling.

2.6 Analyzing the System Model

Once the system model is built, the analyst can begin assessment of the system with respect to its ability to support various "surety objectives" and with respect to its vulnerability to various "initiating events". Each such analysis run on a given system configuration can be merged with other analyses run on the configuration to create an "integrated fault model". In addition, as other system configurations are created that are variations on the original configuration, the same analyses can be rerun and results compared across configurations.

2.6.1 Identifying Surety Objectives

Requirements:

	Release				
	1	2	3	4	
R1			X		A user shall be able to add one or more surety objective stakeholder lists to a given configuration. In doing so, the user shall assign a stakeholder's name to each list added.
R2			X		A user shall be able to edit the stakeholder's name assigned to a list.
R3			X		A user shall be able to delete a stakeholder list from a configuration.
R4			X		A user shall be able to add to a surety objective to a stakeholder list. Each such objective shall be associated with a specific attributed (flow or store) within the system.
R5			X		A user shall be able to delete a surety objective from a stakeholder's list.
R6				X	A user shall be able to copy one or more objectives from a given stakeholder's list. The effect is to leave the objective(s) in the stakeholder's list and to place the objective contents in SAT's paste buffer.
R7				X	A user shall be able to cut one or more objectives from a given stakeholder's list. The effect is to remove the objective(s) from the stakeholder's list and to place the objective contents in SAT's paste buffer.
R8				X	A user shall be able to paste one or more objectives into a given stakeholder's list. The effect is to insert the objective(s) in the stakeholder's list and to leave the objective contents in SAT's paste buffer.
R9				X	The tool shall encourage completeness on the user's part by questioning whether unaddressed attributes within the system have no associated objectives or whether these objectives are yet to be identified.
R10			X		A user shall be able to rank order surety objectives both within a stakeholder's list and across all stakeholder lists.

Questions/Issues:

- None

2.6.2 Identifying Initiating Events

Requirements:

	Release				
	1	2	3	4	
R1			X		A user shall be able to identify those events capable of initiating system action and/or changing system state and that are driven by components on the system's boundary.
R2			X		A user shall be able to identify those events capable of initiating system action and/or changing system state and that are driven by component failures.
R3			X		A user shall be able to mark selected initiating events as being of particular interest from an assessment point of view.
R4			X		A user shall be able to rank order those events that have been marked as being of interest.

Questions/Issues:

- None

2.6.3 Slicing the System Model

Requirements:

	Release				
	1	2	3	4	
R1			X		A user shall be able to slice the system model backwards from a given surety objective.
R2			X		A user shall be able to slice the system model forwards from an initiating event that the user has identified as being of interest.
R3			X		A user shall be able to chop between a selected initiating event and a given surety objective
R4			X		As the user traces through the slice, the tool shall track which branches have been covered and which have not and shall give the user the ability to select the top of a branch not taken as the starting point for more tracing through the slice.
R5			X		A user shall be able to superimpose two or more slices/chops on each other and see where they intersect.
R6	X				A user shall be able to create chops and forward and backward slices of the system model that are rooted in arbitrary points in the system (*i.e.*, not in just those points that have been marked with surety objectives and initiating events of interest).
R7	X				A user shall be able to select a point within the system model and to si-

					multaneously slice forwards and backwards from that point.

Questions/Issues:

- One of the goals in having the user set surety objectives and to designate initiating events of interest is to lay the groundwork for later automated analyses that run a large number of slices and from these slices distill abstract views that are of interest to the analyst.

2.6.4 Performing Fault Tree Analyses

Requirements:

	Release				
	1	2	3	4	
R1				X	A user shall be able to specify that a given surety objective be used as the top event in a fault tree analysis.
R2				X	The tool shall show the user the dependency graph that propagates backwards in the system model (*i.e.*, against flows) from the point with which the top event is associated. In the balance of this document this will be referred to as a backward slice.
R3				X	As the user traces backwards through the dependency graph, the tool shall allow the user to identify intermediate events that contribute to the event currently being analyzed.
R4				X	In identifying an event, the tool shall require that the user associate the event with an attribute or transformation (*i.e.*, a dependency specification contained within a process in a data flow diagram) contained in the system model.
R5				X	If events from other analyses have already been associated with a given attribute or transformation, the tool shall give the user the opportunity to identify one or more of these as contributing to the event currently being analyzed.
R6				X	If the user adds an event to the fault tree and the attribute to which the event is attached cannot be traced backward any farther, then the tool shall mark the event as being a basic event (*i.e.*, one for which this is no other reason for its existence than *just because*).
R7				X	As the user traces through the dependency graph, the tool shall track which branches have been covered and which have not and shall give the user the ability to select the top of an unworked branch and to begin tracing this branch in development of the fault tree.
R8				X	The tool shall give the user the ability to collapse a contiguous set of events into a single event that represents the group and to expand the single event back into the full set of represented events.

Questions/Issues:

- One thing to note here is that a user might approach modeling and assessment in a spiral fashion. After creating a small initial model – sort of a skeleton – he could begin a fault tree analysis. When the analysis takes him to the basic events, he could revert to modeling around these basic events. This elaboration process – putting some meat on the bones – would then turn the basic events into intermediate events and which could then be analyzed further until more basic events are encountered. This iterative process could continue until the real system boundaries are reached and the real basic events associated with these boundaries are identified.
- We need to think about the parameters that control the backward slicing of the system model. Phil has some good ideas on ways to minimize the bushiness of the slice that is generated.
- We need to look at how timing analyses can be used to prune a fault tree.
- We probably need some way of identifying those events produced during forward propagation versus those produced during back propagation. As I write this, I realize that I am really talking about another graph structure here that links events (and in which the events themselves are rooted in the system model) in such a way that any given event can be examined with respect to its predecessors and successors.

2.6.5 Performing Event Tree Analyses

Requirements:

	Release				
	1	2	3	4	
R1				X	A user shall be able to specify that a given flow be used as the initiating event in an event tree analysis.
R2				X	The tool shall show the user the dependency graph that propagates forwards in the system model (*i.e.*, in the direction of the flows) from the point with which the initiating event is associated. In the balance of this document this will be referred to as a forward slice.
R3				X	As the user traces forwards through the dependency graph, the tool shall allow the user to identify the intermediate event(s) that can result from the event currently being analyzed.
R4				X	In identifying an event, the tool shall require that the user associate the event with an attribute or transformation (*i.e.*, a dependency specification contained within a process in a data flow diagram) contained in the system model.
R5				X	If events from other analyses have already been associated with a given attribute or transformation, the tool shall give the user the opportunity to identify one or more of these as resulting from the event currently being analyzed.
R6				X	If the user adds an event to the event tree and the attribute to which the

					event is attached cannot be traced forward any farther, then the tool shall mark the event as being an outcome event[12] (*i.e.*, one for which this is no other reason for its existence than just because).
R7				X	The tool shall make it possible for the analyst to consider which branch(es) of the event tree pass through surety objectives defined by the user.
R8				X	As the user traces through the dependency graph, the tool shall track which branches have been covered and which have not and shall give the user the ability to select the root of an unworked branch and to begin tracing this branch in development of the event tree.
R9				X	The tool shall give the user the ability to collapse a contiguous set of events into a single event that represents the group and to expand the single event back into the full set of represented events.

Questions/Issues:

- Greg and I have discussed various ways of pruning an event tree (*e.g.*, binning the results as they leave a component in order to limit the component's fan-out).
- We need to look at mechanisms for limiting the tree based on timing analyses.

2.6.6 Performing Other Analyses

This section is an oddball section. Its purpose is to be a place to put ideas about other analyses that can be done in the context of the SAT tool.

Analysis Method:

	Release				
	1	2	3	4	
R1					The analyst should be able to select any attribute or transformation within the system model and create models that propagate in both the forward and backward directions as described in the previous two sections. The intent is to support methods like HAZOP and FMEA/FMECA. This permits forward propagation from specified events in a fault tree in order to answer the question, "What else might be happening if this fault is realized in this particular way?" This permits backward propagation from specified events during event tree analysis to answer the question, "How could these events (*i.e.*, those listed in the event tree) occur?"
R2					Timing analyses to check for race conditions. The goal here is to answer the question, "Given an initial set of conditions and assuming that every-

[12] Need to get the correct terminology from Greg.

					thing else behaves as it should, what effect does timing variations have on the outcomes given a certain set of initiating events?"
R3					For the (non-software) portion of the system, we may want to add the ability to model component reliability and to use component failures as initiating events in our analyses.
R4					Set value propagation

Questions/Issues:

- None

2.6.7 Ranking Findings

Requirements:

	Release				
	1	2	3	4	
R1				X	The tool shall be able to show the user which components participate in the largest number of fault trees.
R2				X	The tool shall be able to show the user the relative impact of each component found in the fault trees based on the rankings assigned to surety objectives.
R3				X	The tool shall be able to show the user how often each event appears in the system's fault or event trees.
R4				X	For a given event tree, the user shall be able to weight the severity of the various outcome event relative to one another.
R5				X	For a collection of event trees, the user shall be able to weight the severity of all outcome events relative to one another.

Questions/Issues:

- Need better qualitative measures.
- Need to begin thinking about quantitative measures, particularly if reliability analyses are supported.

2.6.8 Analyzing Assessment Coverage

Requirements:

	Release				
	1	2	3	4	
R1				X	For a given stakeholder, the tool shall track which attributes in the system model have been selected for objectives, which have been excluded, and which have not yet been considered.

R2				X	The tool shall track which input flow attributes have been considered for initiating events, which have been rejected, and which have not yet been addressed.
R3				X	The tool shall track which reliability-based failures have been considered for initiating events, which have been rejected, and which have not yet been considered.
R4				X	The tool shall track which dependency graph branches have been traced in a fault tree analysis and which have not.
R5				X	The tool shall track which dependency graph branches have been traced in an event tree analysis and which have not.
R6				X	The tool shall track which attributes that populate the branch between two events in a fault or event tree have no events assigned to them.

Questions/Issues:

- We may want to give the user the ability to annotate a given attribute or reliability-based failure mechanism it order to explain why it has been rejected from consideration.

2.7 Managing Diagrams

A key goal of the SAT tool is to permit the analyst to both build and analyze the models in as visual a manner as possible. To this end the tool provides the user with both a range of diagram types and with mechanisms for easily managing large collections of diagrams. In order to give the analyst greater insight into the system being assessed, the tool also permits the user simultaneously display two or more diagrams and to synchronize the diagrams so that changes in one diagram are automatically reflected in the other diagrams.

This section identifies the requirements associated with:

- Creating a diagram in SAT.
- Formatting a diagram and its contents.
- Printing a diagram.

2.7.1 Creating a Diagram

Requirements:

	Release				
	1	2	3	4	
R1	X				The user shall be able to view the overall structure of a given system configuration at a glance.
R2	X				The user shall be able to view the immediate structure of a given component (*i.e.*, the major subsystems into which the component decomposes).
R3		X			The user shall be able to view the full structure of a given component (*i.e.*, all of the atomic components that comprise the component).
R4	X				The user shall be able to view, for a source-code derived model element, the source code that that model element represents.
R5	X				The user shall be able to view the interaction diagram associated with a given flow entering or exiting an object.
R6	X				The user shall be able to view all of the flows associated with a given component.
R7	X				The user shall be able to view all of the components associated with a given flow.
R8		X			The user shall be able to view which model elements correspond to a given line or section of source code.
R9		X			The user shall be able to view the state transition diagrams that describe an atomic component.
R10		X			The user shall be able to view the data flow diagram that describes a given state.
R11	X				The user shall be able to view the structure of a given flow.

R12		X			The user shall be able to view the structure of a given store.
R13		X			The user shall be able to view the dependency relationships between the attributes flowing in and out of a given process.
R14			X		The user shall be able to trace the flow of execution in a body of source code (this should include the ability to document where branching decisions were made and to backtrack and continue execution from these decision points).
R15			X		The user shall be able to trace the dependency relationships through the non-source code parts of the model.
R16				X	The user shall be able to view which portion(s) of a fault tree or event tree map to a given component in the system model.
R17				X	The user shall be able to view to which system component(s) a given portion of a fault tree or event tree maps.
R18				X	The user shall be able to view which assessment model events are associated with a given component or flow.
R19	X				The user shall be able to view which components are associated with a given slice of the system.
R20				X	The user shall be able to select specific endpoints in a fault tree or event tree and view which components participate in the selected thread.
R21			X		The user shall be able to view which attributes in the system have been considered by a given stakeholder and which have not during the establishment of surety objectives (I imagine this being a little like the displays used in physical security in which you can view at a high level if any problems exist and then drill down as needed in order to view the source of the problem).
R22				X	The user shall be able to view which input flow attributes (*i.e.*, those coming from the system's boundary) have been considered as initiating events in event tree analyses and which have not.
R23				X	The user shall be able to view which reliability-based failure mechanisms have been considered as initiating events in event tree analyses and which have not.
R24				X	The user shall be able to view which dependency graph branches have been traced in a given fault tree analysis (or collection of analyses) and which have not.
R25				X	The user shall be able to view which dependency graph branches have been traced in a given event tree analysis (or collection of analyses) and which have not.
R26			X		In slicing backwards from a surety objective, the user shall have the ability to highlight, within the context of the entire slice, those threads that intercept initiating events that the analyst has marked as being of interest.

R27			X		In slicing forwards from an initiating event, the user shall have the ability to highlight, within the context of the entire slice, those threads that intercept surety objectives that the analyst has established.
R28	X				A user shall be able to open multiple diagrams at one time.
R29	X				A user shall be able to synchronize all visible diagrams.
R30	X				A user shall be able to annotate a diagram.
R31		X			When a diagram is created from the system model without user intervention, a user shall be able to save the diagram if desired. In doing so, the user assigns a name to the diagram.
R32		X			A user shall be able to display a diagram's identifying information (project to which it belongs, configuration to which it belongs, diagram revision number, date/time of last revision, *etc.*).

Questions/Issues:

- Is it possible to create a view that allows a fault tree or event tree to be hierarchically collapsed and expanded as the associated components in a system model view are collapsed or expanded? It seems that this would require identifying/labeling nodes that aggregate collections of events or collections of the aggregate nodes.
- It is entirely possible that the user will be able to generate from the system model views that were never seen in the course of building the model. As some of these may have usefulness beyond simply browsing, the tool should allow the user to capture that view so that it can be retained for later use. One example of this would be a slice of the model that is used to explain the flow of execution from one point in the system to another point.

2.7.2 Formatting a Diagram

SAT provides a number of diagram types for an analyst to use. Each of these diagram types carries with it its own default set of formatting properties. Each instance of a given type of diagram derives its own formatting properties from this default set. As desired, the analyst can override one or more properties for a given diagram instance or for elements within this instance.

Requirements:

	Release				
	1	2	3	4	
R1	X				A user shall be able to inspect the formatting properties for any diagram type in SAT.
R2	X				A user shall be able to alter the formatting properties for any diagram type in SAT.
R3	X				A user shall be able to return a diagram type's formatting properties to their defaults if they have been altered in any way.

R4	X				A user shall be able to inspect the formatting properties for any diagram instance in SAT.
R5	X				A user shall be able to alter the formatting properties for any diagram instance in SAT.
R6	X				A user shall be able to return a diagram instance's formatting properties to their defaults if they have been altered in any way.
R7	X				A user shall be able to inspect the formatting properties for any element contained in a diagram instance in SAT.
R8	X				A user shall be able to alter the formatting properties for any element contained in a diagram instance in SAT.
R9	X				A user shall be able to return a diagram element's formatting properties to their defaults if they have been altered in any way.
R10				X	A user shall be able to copy the formatting properties of a diagram type, diagram instance, or diagram element and then apply this formatting to one or more diagram types, diagram instances, and/or diagram elements.
R10.1				X	In working with elements in a diagram, the user shall be able to select one or more for the purpose of copying or applying attributes.
R10.2				X	When the user copies properties from multiple diagram elements at one time, only those properties that are set the same for all selected elements shall be copied.
R10.3				X	When the user applies properties to multiple elements in a diagram, all properties set are applied to all selected elements.
R11	X				When SAT closes a project and then reopens it later, all of the formatting applied to each of the diagrams shall be preserved.
R12	X				A user shall be able to set the page size and associated parameters (*e.g.*, margins) of any diagram instance in a project.
R13	X				A user shall be able to zoom in and out on any given diagram instance.
R14	X				For diagrams that are zoomed in so as to not display the entire diagram at once, the user shall be able to pan the drawing up, down, left, and right as needed in order to be able to inspect the entire diagram.
R15	X				A user shall be able to reposition elements within a diagram.
R16	X				At a minimum, a user shall be able to set the formatting on: • Fonts (style, size, color, location relative to the items with which they are associated) • Drawing elements (line style and thickness, color, fill style)

Questions/Issues:

- None

2.7.3 Printing a Diagram

Requirements:

	Release				
	1	2	3	4	
R1				X	A user shall be able to preview the results of diagrams to be printed.
R2				X	A user shall be able to print one or more diagrams.
R3				X	A user shall be able to select the printer that is to be used in printing diagrams.
R4				X	A user shall be able to configure a printer selected for printing diagrams.

Questions/Issues:

- None

3. ANCILLARY FUNCTIONS

This section describes other support capabilities needed by SAT. These include mechanisms to manage the vulnerability knowledge base, mechanisms to manage the surety objectives knowledge base, a means of managing the library of components used in building system models, and miscellaneous functions (*e.g.*, undo) that apply to all aspects of SAT tool use.

3.1 Managing the SAT Libraries

SAT uses three libraries in support of its operation:

- Component library
- Vulnerability library
- Surety objectives library

The component library stores component models that have been created in earlier assessments and then saved to the library. The information stored in the library includes both its "system model" information and its analytical model information.

The vulnerability library consists of component fragments that capture the vulnerability information associated with well-known commercial products and with user-developed components stored in the component library.

The surety objectives library enumerates objectives associated with a number of different domains. For example, the *things of value* domain might contain objectives like:

- Can't be stolen
- Can't be forged
- Can't be diminished

3.1.1 Managing Component Libraries

Requirements:

	Release				
	1	2	3	4	
R1				X	The user shall have the ability to alter the iconic representation of a component stored in or being added to the library.
R2				X	The user shall have the ability to create named collections that organize components into groupings useful to the user. In doing this, a given component may be assigned to more than one collection.
R3				X	A user shall be able to browse collections and to inspect the internals of any component in the library.
R4				X	A user shall be able to conduct a computer-aided search of a component that returns a list of components matching a given description.
R5				X	A user shall be to add to the component library a component defined within a given project.

R6				X	A user shall be able to delete a component from a component library.
R7				X	A user shall be able to import a collection of components into the tool's component library.
R8				X	A user shall be able to export a collection of components from the tool's component library.
R9				X	A user shall be able to maintain in the library multiple versions of a given component.

Questions/Issues:

- None

3.1.2 Managing the Vulnerabilities Library

Requirements:

	Release				
	1	2	3	4	
R1			X		The user shall be able to inspect the database's contents (*i.e.*, which components are referenced in the database).
R2			X		The user shall be able to add to the database a new component to which vulnerabilities are to be attached
R3			X		The user shall be able to maintain multiple versions of a given component in the database.
R4			X		The user shall be able to delete a given component from the database.
R5				X	The user shall be able to copy a component from the database. The effect is to place the component contents in the SAT paste buffer and to leave the component in the database.
R6				X	The user shall be able to cut a component from the database. The effect is to place the component contents in the SAT paste buffer and to remove the component from the database.
R7				X	The user shall be able to paste a component into the database. The effect is to create a new database entry and to leave the component contents in the SAT paste buffer.
R8			X		The user shall be able to create one or more sub-models that explain a given component's vulnerabilities in terms of flows, processes, *etc.*
R9			X		The user shall be able to delete one or more vulnerability sub-models from a given component in the library.
R10				X	The user shall be able to copy one or more vulnerability sub-models from a component in the database. The effect is to place the sub-model contents in the SAT paste buffer and to leave the sub-model in the component in the database.

R11				X	The user shall be able to cut one or more vulnerability sub-models from a component in the database. The effect is to place the sub-model contents in the SAT paste buffer and to remove the sub-model from the component in the database.
R12				X	The user shall be able to paste one or more vulnerability sub-models into a component in the database. The effect is to place the sub-model(s) in the component in the database and to place the sub-model contents in the SAT paste buffer.

Questions/Issues:

- Should vulnerabilities just be another aspect of components in the component library?

3.1.3 Managing the Surety Objectives Library

The surety objectives library contains a collection of lessons learned regarding things to be assured and things to be prevented in various domains. While these domains typically apply to a given technology or kind of entity (*e.g.*, microwave transmissions or humans), the library also contains more abstract domains, such as business processes and roles in an organization. Within each domain in the library are zero or more objectives. As appropriate, each such objective is cross-linked with related objectives that may be of interest to the analyst.

Requirements:

	Release				
	1	2	3	4	
R1		X			The user shall have the ability to browse the domains that populate the surety objectives database.
R2		X			The user shall have the ability to browse the objectives that populate any given domain.
R3		X			The user shall have the ability to browse the objectives that are cross-linked with a selected objective.
R4		X			A user shall be able to add a new domain to the library. In doing so, the user shall enter the name of the domain.
R5		X			A user shall be able to alter the name of a domain.
R6		X			A user shall be to delete a domain from the library. This has the effect of removing both the domain and the objectives that it contains.
R7		X			A user shall be able to add a surety objective to a selected domain
R8		X			A user shall be able to edit a selected surety objective.
R9		X			A user shall be able to delete a surety objective from a domain.
R10		X			To a selected objective, a user shall be able to attach crosslinks (bi-directional hyperlinks) to one or more other objectives in the same or in

					different domains.
R11		X			For a selected objective, a user shall be able to remove crosslinks with another objective.

Questions/Issues:

- I would like to be able to use domain names as attributes that a user modeling the system could use to characterize the system's components and flows. For example, "Is this an electromechanical system?" or "Does this component have monetary value?" Associated with these attributes would be some number of objectives/issues, such as the fact that things that have monetary value should retain the value they have or, at the least, not depreciate more quickly than is reasonable to expect. The goal here is to make it possible to identify a set of potential surety objectives that an analyst can prune as needed to identify real objectives for a component or flow.
- How does the system handle domain name changes if the domain name is treated as an attribute of components and flows for the purpose of establishing objectives?
- One kind of hyperlinking between objectives that I want to be able to provide is between abstract objectives in abstract domains and the associated concrete objectives in concrete domains.

3.2 General Capabilities

This section lists other miscellaneous capabilities of the SAT tool.

3.2.1 Help

Requirements:

	Release				
	1	2	3	4	
R1	X	X	X	X	SAT shall provide the user with complete on-line help facilities.
R2	X				SAT shall provide an *About* screen that gives developers credit.

Questions/Issues:

- None

3.2.2 Source Document Management

Requirements:

	Release				
	1	2	3	4	
R1				X	A user shall be able to add to a project the source documents used by the analyst to guide the development of the system model.
R2				X	A user shall be able to hyperlink from a diagram of a system component to a given location in the source documentation.
R3				X	A user shall be able to hyperlink from a given location in the source documentation to a diagram in a given configuration.
R4				X	A user shall be able to remove a project one or more source documents. When this occurs, any hyperlinks between the source document and the system model will be removed.

Questions/Issues:

- None

3.2.3 Import/Export

Requirements:

	Release				
	1	2	3	4	
R1				X	A user shall be able to export diagrams and model segments to other applications (this includes embedding them in documents in other applica-

					tions).

Questions/Issues:

- None

3.2.4 Usability Features

Requirements:

	Release				
	1	2	3	4	
R1				X	A user shall be able to *undo* and *redo* n number of operations performed in SAT.
R2				X	A user shall be able to find and replace text in one or more diagrams within a project.
R3				X	A user shall be able to view changes made to a given project since the last time that changes were committed.
R4				X	A user shall have the ability to control to the maximum reasonable extent the environment variables that affect SAT's operation (*e.g.*, where the starting directory for opening and saving data files is to be located).

Questions/Issues:

- None

DISTRIBUTION

1	MS	0188	LDRD Office, 1030
2		0785	J. Espinoza, 6514
1		0785	R. E. Trellue, 6514
2		0785	P. L. Campbell, 6516
1		0785	R. L. Hutchinson, 6516
2		0839	R. L. Craft, 16000
1		0899	Central Technical Files, 8945-1
2		0899	Technical Library, 9616
1		0612	Review & Approval Desk, 9612 For DOE/OSTI