Source Code Analysis Tool

Concept of Operations

Version 1.0

Table of Contents

[1. Introduction 2](#_Toc93307244)

[2. Motivation 2](#_Toc93307245)

[3. Overview of the SCA Tool 3](#_Toc93307246)

[3.1 Definitions 3](#_Toc93307247)

[3.2 Analysis Criteria 5](#_Toc93307248)

[3.3 Benefits of Source Code Analysis Tool 6](#_Toc93307249)

[4. Source Code Analysis Tool – Desired Functionality 7](#_Toc93307250)

[5. System Structural Definition 7](#_Toc93307251)

[6. SCA System Architecture 7](#_Toc93307252)

[6.1 User Interface 8](#_Toc93307253)

[6.1.1 File Menu 8](#_Toc93307254)

[6.1.2 Information Menu 9](#_Toc93307255)

[6.1.3 Data Entry Menu 9](#_Toc93307256)

[6.1.4 Interface Menu 10](#_Toc93307257)

[6.2 Query Subsystem 10](#_Toc93307258)

[6.3 Report Subsystem 10](#_Toc93307259)

[6.4 Data Entry Subsystem 11](#_Toc93307260)

[6.4.1 Object Description 11](#_Toc93307261)

[6.5 Functional Analysis Subsystem 11](#_Toc93307262)

[6.5.1 Class Analysis 12](#_Toc93307263)

[6.5.2 Attribute Analysis 12](#_Toc93307264)

[6.5.3 Method Analysis 12](#_Toc93307265)

[6.5.4 Module Analysis 13](#_Toc93307266)

# 1. Introduction

(a) Source Code Analysis (SCA) is a method for analyzing the properties of source code as it is written and modified during development and testing, and, later during the enhancement stage after deployment. SCA requires that we capture the elements of source code – variables, classes, methods, functions, etc. – as well as structural information in order to be able to ask questions about the source code as it evolves. A second use of such a system is to understand where information is missing that will help developers in debugging and documenting a system.

(b) Few systems exist today that provide this capability to developers, documentation, and maintenance personnel. As our organization embarks upon developing more complex applications, we need a tool to help us keep track of the elements and structures that characterize our applications. A survey has revealed that no tools are adequate for the kind of tools that we need to support our development efforts.

(c) We did identify a tool that had been developed for the Interlisp-D system back in the late 1980s. Interlisp-D was an integrated software development environment for designing, developing, debugging, and deploying artificial intelligence applications. This Concept of Operations is drawn from our understanding of the functionality and features of that tool and a project called the Electronic Encyclopedia developed for the IRS in the mid-1990s.

# 2. Motivation

(a) Our primary motivations for developing this tool are to provide a capability that can significantly enhance our development teams’ ability to develop software applications to support our business operations. One aspect of this is to enable them to track the components and structures of our software applications as they are developed.

(b) Another aspect is to provide a repository to answer questions about the elements and structure of software applications not only during development (including documentation), but also during succeeding maintenance and enhancement stages of an applications life cycle.

(c) A third aspect is to ensure that the requirements actually define the system that the customer wants and is an acceptable description of the system to be implemented. If a system is built that does not meet our needs, then the staff will not use it. Similarly, if it does not perform efficiently, is missing some functionality, or is hard to use, our users will not use or – more often – not use it effectively. This can have an impact on our user’s morale, as well as affect our reputation and the reputation of our company as well.

(d) The objective of this tool is to enable the design, development, and description of high-quality, business-oriented software that ensures that developers and project managers have a coherent, consistent view of the status of the system under development and/or maintenance/enhancement on a continuing basis.

(e) Numerous studies in the technical literature have demonstrated that correcting an error or problem during requirements engineering is much less expensive than fixing it during system design, system development, or even as late as during system testing. Other studies have shown that while it is more costly to correct problems during system design and system development, the worst time to correct problems is during system testing and post deployment.

(f) Collecting data on a continuous basis during the life cycle stages can help to identify problems earlier and correct them before they impact overall success of a system development effort.

(g) Although it is desirable to have tools that can examine source code and collect data for inclusion in the repository, this tool will focus on manual collection of data as the source code is being developed.

(h) For illustrative purposes, this tool will assume an object-oriented system to be developed using an object-oriented programming language such as Java or R.

(i) *For the purposes of this project, your team will develop requirements for and design the SCA Tool by describing it as you develop this repository.*

*(j) In order to help you focus on the concepts to be embodied in the tool, you will use the Java source code for a SNOBOL3 system to help you think through the issues for this tool.*

# 3. Overview of the SCA Tool

(a) The Source code Analysis (SCA) tool will support these activities (in the final version):

1. Provide a repository (database) for collecting data about functions, classes, objects, methods, and structures during analysis, design and development;

2. Provide descriptions of functions, classes, objects, methods, and structures for the system under development;

3. Map requirements to functions, classes, objects, methods, and structures;

4. Provide a GUI-based interface to enter and update data about functions, classes, objects, methods, and structures;

5. Provide functions for examining structural data to identify missing information, isolated elements, duplicate data, and other analyses;

*(b) Ultimately, upon completion of the tool, it will support all stages of the system life cycle.*

## 3.1 Definitions

(a) The following definitions are used in this document:

*Information Object*: An information object is an entity represented in the source code that contains data/information, may be constant or not, and is operated upon by Information Processes.

*Information Processes*: An information process is a collection of methods that transforms data/information or computes new data/information.

*Object*: An object is an entity representing some real-world entity –either tangible or intangible –within the application to be/being developed. Objects are implemented in the sense of an object-oriented programming language (OOPL).

*Class*: A Class is a description of a set of objects in the sense of an OOPL. The objects within a class are called *instances*. A class has attributes and methods.

*Attribute*: An attribute of a class holds the value of data/information. Attributes have a data type and structure.

*Method*: A method is a procedure within a class that operates upon attributes and objects. A method can have local variables which are instantiated for the duration that a method is being executed.

*Component*: A component is a collection of one or more classes and, recursively of its instances, which accomplishes a set of functions within the system. For purposes of this project, think ‘Package’ from Java as an example.

*Module*: A module is a collection of one or more components that together provide a capability within the system or subsystem.

*Subsystem*: A subsystem is a collection of one or more modules that implement a major piece of functionality for a software system. As an example, one might consider database management as a subsystem. Note that a database management system may rely upon external DBMS software such as MySQL to provide the bulk of the functionality, but the subsystem provides the interface from the rest of the software system to the DBMS.

*System*: A system is a collection of one or more subsystems that implement the functionality desired by the customer for the application.

*Process*: A process is a flow of control within a system starting with a class and proceeding via messages (invoking methods) through one or more classes.

*Data Type*: Attributes and method parameters have a data type. For purposes of this project, we will assume that the standard data types are Integer, Floating, String, Boolean, Array of <data type>. An attribute may also have a data type of <class> and, correspondingly, of Array of <class>.

(b) A conceptual Architecture for a software system is depicted in figure 1.

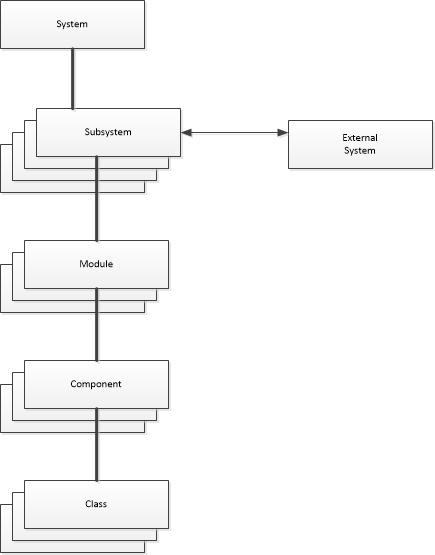


Figure 1: A Conceptual Architecture

## 3.2 Analysis Criteria

Analysis of the source code data captured in the repository is assessed according to four criteria:

(a) Completeness is the assurance that all parts, components, and steps are identified and included in the repository. Two questions need to be answered:

1. Does the repository include all entities called for or implied in the Statement of the Problem (e.g., the CONOPS)?
2. Does the repository identify all system interfaces and specify the system behavior with respect to each?

(b) Correctness involves two concepts:

1. Are the entity descriptions free from error?
2. Does the description of each entity (and any explanation) make common sense?

Three approaches to correctness checking are available:

1. Determine the correctness based on the background and experience of the reviewer;
2. Determine the correctness based on the background and experience of a Subject Matter Expert (SME); or
3. Identify what incorrectness might be suspected and what its impact would be on the system.

(c) Consistency is extremely important to ensure that the data about the entities and all concepts are clear and free from error insofar as the users and the SME can determine. It also examines all models, algorithms, and numerical techniques to ensure they are mathematically compatible and other explanations about a system’s proposed operation to ensure that they are not contradictory.

External consistency is divided into direct and indirect consistency. Direct consistency involves ensuring that concepts carried forward from earlier documents are correct and do not contradict one another. For example, the System Requirements Document (SRD) should ensure that each requirement is testable, that acceptance criteria are specified, and acceptance criteria are consistent with accepted practices and principles in software design and construction. Indirect consistency examines concepts, processes, etc. that may be described in different documents – both across different stages as well as across documents describing different systems or subsystems of complex software architectures at each stage.

(d) Coherency is defined as the clear, unambiguous, systematic connection of component parts of the system. This can be applied to one or more systems in a complex software architecture, one or more subsystems for a particular system and across systems, and one or more modules for a subsystem, across subsystems or systems. Ambiguity may arise when statements are examined in relation to applicable guidelines, standards, and other requirements.

## 3.3 Benefits of Source Code Analysis Tool

Our organization expects the benefits of using the Source Code Analysis (SCA) tool, once it is completed and deployed, to be the following:

(a) Provides a manpower multiplier that allows project personnel to find information about the software system structure as the software system evolves through the development process;

(b) Enables project personnel to identify relevant issues easily and quickly by querying the repository;

(c) Support many projects through different repository instantiations;

(d) Avoids repetitive analysis as data about the software system is captured and analyzed as the system is designed, developed, and tested.

(e) Captures design and development knowledge about the software system which is useful in developing system documentation and informing new team members as they join a project;

(f) Provides an automated foundation for future tools which can support automatic code generation.

# 4. Source Code Analysis Tool – Desired Functionality

(a) The Source Code Analysis (SCA) tool should provide designers and developers with the capability to capture and track the elements of a software system design. It is assumed that an object-oriented system approach, similar to Java, R, Smalltalk, etc., will be used to implement new applications for our organization.

(b) Initially, this process will require manual entry of data by designers and developers. Our organization is examining – in other projects, a semiautomated approach to capturing some of the information to be entered into the SCA Tool repository.

# 5. System Structural Definition

Examples provided in this section will be drawn from the files in the SNOBOL3src directory.

(a) The system will capture a description of the intended software system in terms of the entities specified in section 3.1(a).

(b) For each type of entity, it will create an object that has relevant attributes for that type of entity.

(c) For each type of entity, it will record the relations to other entities in the system For example, an *attribute* belongs to a class. A *class* is contained within a *component*, and so forth.

(d) For structural entities, such as a module, the system will keep a list of the subordinate entities that are contained within it. For example, a *module* will have a list of *components* belonging to it.

(e) It is a good idea to draw diagrams, using UML, to show how entities relate to each other.

Note: We will describe the most useful UML diagrams in Lecture 4.

# 6. SCA System Architecture

The SCA Tool will have a software architecture. As you proceed to develop this tool, you should draw a software architecture which shows the relationships between the parts of the tool.

(a) The SCA will be organized into several subsystems:

1. User Interface
2. Query Subsystem
3. Report Subsystem
4. Data Entry Subsystem
5. Functional Analysis Subsystem

(b) For each subsystem, the repository will capture:

1. The subsystem architecture
2. Its interfaces to other subsystems and to external systems

(c) For each module, the repository will capture

1. A functional description of the module
2. Its interfaces with other modules in its subsystem and in other subsystems

(d) For each interface of a subsystem or module, the system will capture:

1. The data input, output, or both through the interface
2. The entities that the interface connects

(e) For each process, the sequence of calls made by the process to provide the relevant information.

Note: A process is usually initiated by a query or report action. Consider defining a class called Process which has instances of the different types of processes in the system.

## 6.1 User Interface

(a) Based on an analysis of the requirements, a preliminary user interface (at a high level) can be designed. A mockup sketch of the screen layout can be prepared based on your understanding of the functions.

(b) Preliminary menus and menu items can be identified and briefly described along with possible functionality.

(c) A command line interface will allow the user to enter a command string with arguments which will be processed by the user interface and directed to the appropriate subsystem.

(d) Menus will allow a user to select functions which the invoke functionality in the appropriate subsystem.

(e) The user interface (UI) will provide the following menus:

- File

- Information

- Data Entry

- Interface Menu

*Note: The detailed design of the user interface and its implementation is the responsibility of the programming team. This section will need to be extended during the system development process, which you will not cover in this effort.*

*Note: As you think about a User Interface, it is Subsystem of the SCA Tool. As a Subsystem, it will have modules, which have components and classes. You need to specify these as part of your design.*

*For example, a User Interface will have one or more windows, where Window is a class. Similarly, Menu, Report, and Interface will be classes.*

### 6.1.1 File Menu

*Note: A Repository would be created prior to using the SCA Tool using the file creation tools associated with the particular operating system.*

*You do not need to do this, but should consider how it would be done and describe this in your System Test Plan. Why? Because in order to test the implementation and operation of the SCA Tool, you will need to have a populated repository.*

*For purposes of illustration, I will assume that a repository named SNOBOL3.sca exists.*

The File Menu will have at least the following menu items, which will pop up a window to:

- Open a Repository

- Close a Repository

- Log into the SCA Tool

- Logout of the SCA Tool

*You will need to decide what data is required to handle each menu item.*

*So, to open the SNOBOL3.sca repository, you would need to know its locations in your system environment, whether on a local machine or a server.*

### 6.1.2 Information Menu

The Information Menu will have at least the following menu items, which will pop up a window to:

- Query

- Report

- Analyze

*You will need to decide what data is required for to handle each menu item. Discussion of some of these functions is contained in Section 6.*

### 6.1.3 Data Entry Menu

The Data Entry Menu will have at least the following menu items, which will pop up a window

- Create an Object (Class, Attribute, Method, Module, Subsystem, System, Process)

- Update an Object (by Name, which pops up appropriate window)

- Print an Object (by Name)

*You will need to decide what data is required to handle each menu item.*

*So, to get you started, assume the System is named SNOBOL3. You would create a System object with the name SNOBOL3 using the SCA Tool.*

*For the purposes of this project, let us consider Java subdirectories to be subsystems. Inspecting SNOBOL3, we see two subdirectories – SNOBOL3 and tools. Each of these contains multiple files.*

*For the purposes of this project, consider each file in a subsystem to be a module. A module may have one or more classes defined within it.*

*So, you would need to create two entries for subsystems for SNOBOL3src.*

*Within Tools, there are several files – each of which is a module. Picking Pos.java, we see it has one class in it. So, we would define a module for Pos and a Class for Pos.*

**NOTE: You are not really defining these entities. What you should do is to inspect these files to see what information should be used to describe them in the SCA Tool. This will help you define requirements and, later, specify the design for the tool.**

### 6.1.4 Interface Menu

(a) An *interface* will be a special type of class that is used to access external system and also external files on mass storage or across a network. Thus, an interface is a subclass of Class.

(b) The reason or decoupling the interface to external elements (other systems, files) is to allow such entities to be update or replaced without requiring significant updates across the system under description.

(c) In the future, an update to a system under description may/will be required if an external entity is enhanced or replaced such that it provides either new functional capabilities or new types of data that would be beneficial to the system herein described.

**Note: An interface as defined here is NOT a Java interface.**

## 6.2 Query Subsystem

Some types of queries will be suggested in Section 5, but you should consider as you develop the specification for the repository what other types of analysis you could perform on the available data.

(a) The Query Subsystem will accept queries entered through the User Interface and process them to return data/information to the user.

(b) Queries generally yield small amounts of information, often very precise, such “What attributes does class <x> have?”

(c) Queries may have formats or be ad hoc.

(d) A query should be checked that the entities that it refers to are contained within the repository. If not, it should return a warning/error message.

## 6.3 Report Subsystem

(a) The Report Subsystem will produce formatted reports of, possibly, substantial amounts of content from the repository.

(b) Some types of reports are (but you should determine what additional reports might be needed):

- List all [classes, modules, subsystems] by name

- List all interfaces by name with the classes they connect to.

- List the structure of a class in terms of attributes and/or methods

- List the structure of a module

- List the structure of a subsystem

- List the structure of the system.

*For example, you would specify a requirement for a particular type of Report called System Structure Report which would have the option of describing the structure of any one of the entities listed above.*

*As an example, a user could specify Module and then specify Compiler. The report would show two classes (if you inspect ompiler.java).*

## 6.4 Data Entry Subsystem

(a) The Data Entry Subsystem (DES) will have both command-based data entry and menu-selected forms for entering data into the repository.

(b) When attempting to enter new objects into the repository, the DES will check to see if an object with the same already exists. If so, it alerts the user.

(c) The DES will allow the user to update objects in the repository.

(d) The DES will allow the user to assign attribute to an object.

(e) The DES will allow the user to assign methods to an object. The description of a method will contain it input arguments and the data type of its result. An example of a specification might be:

[<result class name>, void]<class name>.<Method name>(<list of arguments by name and data type>)

Note: What is to be captured in the repository is a template for a class of objects consisting of attributes and methods. The repository – at this time – will not contain any information about code of a method. This is possibly a later extension to the system.

(f) The types of objects that the user will be able to enter into the repository include System, Subsystem, Module, Class, Method, Attribute, Interface.

(g) The Type of a <System> can be either Internal or External. Internal systems are owned by our organization, while External systems exist outside of our organization, but our systems must communicate with them.

(h) The variables that will be used by a method.

(i) For each variable, its data type and description.

(j) For methods, a pseudocode description of the method including key variables.

Note that many of the classes use other Modules, e.g., other Java packages. So a provision would need to be made for a class to have a dependency list both for modules and classes within those modules.

### 6.4.1 Object Description

This information is needed for every object in the repository.

(a) Each object will be described by a textual description.

(b) Each object will have a date and time stamp associated with its creation in the repository.

(c) Each object will have a date and time stamp associated with the last time the object was updated.

(d) Each object will have an identifier of the user who created the object in the repository.

(e) Each object will have an identifier of the last person to update the object in the repository

(f) Each object will have an associated list of identifiers and date/time stamps for the last N updates to the object. N will be a system parameter.

## 6.5 Functional Analysis Subsystem

The Functional Analysis Subsystem contains methods and procedures for analyzing different aspects of the contents of the repository and returning those results to either the Query Subsystem or the Reporting Subsystem.

### 6.5.1 Class Analysis

Some of the functions that could be performed in analyzing classes as they are defined. You will need to think about additional functions, the data structures in the repository needed to support them, and the queries that would initiate them.

(a) What classes are used by this class? Another class B is used by this class A if class A sends a message to class B.

(b) What are the attributes of this class?

(c) What are the methods of this class?

### 6.5.2 Attribute Analysis

Some of the functions that could be performed in analyzing methods as they are defined. You will need to think about additional functions, the data structures in the repository needed to support them, and the queries that would initiate them.

(a) List the attributes by data type.

(b) List the descriptions for each attribute.

### 6.5.3 Method Analysis

Some of the functions that could be performed in analyzing methods as they are defined. You will need to think about additional functions, the data structures in the repository needed to support them, and the queries that would initiate them.

(a) What methods in class A are invoked from other classes?

(b) What input parameters are required by this method?

(c) Which methods generate a void result?

(d) Which methods generate a result of <class> type?

(e) List the variables a method uses.

As you inspect some of the methods in the Java classes in the SNOBOL3 files, you should think about what you want to specify in the SCA Tool.

Consider the Frame.java file which contains several pieces of information:

1. It uses a Java package java.util.HashMap

2. It has multiple attributes which would be defined if you were using the tool. Keeping the requirements abstract, you might say:

*The SCA Tool shall create an entry for each attribute of a class.*

*The SCA Tool shall record the data type of each attribute in a class.*

*The SCA Tool shall record an initialization value, if specified, for each attribute in a class.*

*The SCA Tool shall record a description for each attribute in a class, if specified.*

These are examples to get you started, but you will need to do further analysis.

Similarly, there are two methods defined, so you might say:

*The SCA Tool shall create an entry for each method of a class.*

*The SCA Tool shall record the result type for each method or void.*

*The SCA Tool shall record the names and types of each parameter required by a method.*

Again, these are examples to get you started.

### 6.5.4 Module Analysis

Some of the functions that could be performed in analyzing modules as they are defined. You will need to think about additional functions, the data structures in the repository needed to support them, and the queries that would initiate them.

(a) What classes are defined within this module?

(b) What module attributes are defined, e.g., global to all classes within the module?

(c) Where is global attribute used, e.g., in which classes?

(d) What attribute values are set by methods as return values?

And so forth.