Assumptions Management in Software Development

Grace A. Lewis
Teeraphong Mahatham
Lutz Wrage

August 2004

Integration of Software-Intensive Systems Initiative

Technical Note
CMU/SEI-2004-TN-021

This work is sponsored by the U.S. Department of Defense.

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Contents

List of Figures

List of Tables

Preface

The idea of assumptions management came out of an Independent Research and Development project sponsored by the Software Engineering Institute (SEI) in 2002-2003 in the area of sustainment [Bachmann 03, Seacord 03]. This project was lead by Robert Seacord, a Senior Member of the Technical Staff who currently works in the SEI Networked Systems Survivability (NSS) Program.

A prototype was built that demonstrates the application of assumptions management, including the recording and extraction of assumptions from Java source code into a repository, and the Web-based management of these assumptions. In the course of developing this tool, assumptions management was utilized, illustrating many of its benefits. This tool is called the Assumptions Management System (AMS).

Abstract

Software developers constantly make assumptions about the interpretation of requirements, design decisions, operational domain, environment, characteristics of input data, and other factors during system implementation. These assumptions are seldom documented and less frequently validated by the people who have the knowledge to verify their appropriateness. Additionally, the business, legal, and operating environments are always changing, as well as the software itself, rendering previously valid assumptions invalid.

This technical note explores assumptions management as a method for improving software quality. This exploration covers assumptions management concepts, results of work on a prototype Assumptions Management System, conclusions, lessons learned, and potential work in this area.

1 Assumptions Management

1.1 Motivation

Assumptions have a long history in software. Parnas characterized *interfaces* as the assumptions that modules could make about each other, and most of his software engineering contributions reflect that observation [Parnas 71]. Garlan coined the term *architectura*l *mismatch* as the consequence of inappropriate assumptions that components make about the environment in which they are to operate and about other components [Garlan 95]. Lehman states that the gap between a system and its operational domain is bridged by assumptions, explicit and implicit [Lehman 00].

Some languages, although not explicitly, provide constructs for recording assumptions. An *assertion* is a Boolean statement that is used to check whether the functional behavior of a program corresponds with its intended behavior or whether a data value lies in a specified range. An assertion is therefore a statement of the *assumed* behavior of a piece of code or value of data. Languages such as Java, C, and C++ provide constructs to create assertions that can be checked at runtime. An *exception* is a piece of code that handles generic error conditions. An exception could therefore be seen as a way to state and manage behavior that does not fall within what is *assumed* as expected behavior. Exceptions are used in many modern programming languages, including C++, Java, and Eiffel. Pre-conditions and post-conditions are also formal constructs used by developers as part of the documentation of methods and procedures. A *precondition* states what must be true before the execution of a method or procedure is called. A *postcondition* states what must be true after the execution of a method or procedure. Preconditions and postconditions could then also be seen as *assumed* pre- or post-execution states.

The previous examples clearly illustrate that software developers during their day-to-day work are constantly making assumptions about the interpretation of requirements, design decisions, the operational domain, the environment, and the characteristics of input data, among others. Usually these assumptions are not documented and often they are not validated by the people with the knowledge to verify their appropriateness. Additionally, the real-world domain and the software itself are always changing. While the initial assumption set was valid, individual assumptions will, as time goes on, become invalid with unpredictable results or, at best, lead to operation that is not totally satisfactory [Parnas 94].

1.2 Assumptions

An *assumption*, as defined in one of the entries in the Webster's Revised Unabridged Dictionary, is "the act of taking for granted, or supposing a thing without proof; supposition; unwarrantable claim" [MICRA 98]. Most developers would agree that assumptions are an inherent part of software development. Every time a decision is made—about how to design an interface, how to implement an algorithm, if and how to encapsulate an external dependency—assumptions are made concerning how the software will be used, how it will evolve, and what environments it will operate in. What differentiates assumptions from assertions or pre-conditions is that they are not given to a developer as part of a specification. They correspond to a decision made by a developer in the light of insufficient or unavailable information, vague requirements, previous experience, lack of confidence or knowledge, or any situation in which a decision must be made so that a certain piece of code will work or progress can be made. As stated previously, the unfortunate aspect of software development is that these decisions are seldom if ever recorded, communicated, or reviewed. As a result, they may be incompatible with assumptions made elsewhere in the code, or with design- or system-level assumptions. These incompatibilities may lead to the insertion of defects or post-deployment failures. Furthermore, individual assumptions may become invalid as a result of changes in the environment in which the system is deployed, over the life of the system.

To address this problem, the following elements are necessary:

- an approach to record assumptions in "real-time" in a manner that is not disruptive for the developer
- a mechanism to validate the assumptions by the people that possess the knowledge to declare each assumption valid or invalid
- a tool to search for assumptions by other developers and people involved in the development effort

This set of elements is what we call *assumptions management*.

1.3 Assumptions Management

Assumptions management entails a shift in software development culture in that the assumptions that are part of the development process must also be recorded in source code and other software artifacts. The potential benefit resulting from this practice may be tremendous in the earlier identification and elimination of design- and requirement-level defects, and in improved change analysis for more predictable and cost-effective software evolution.

The failure of developers to keep design and other documentation consistent with evolving source code is an established and well-known phenomenon. From an infrastructure perspective, it is clear developers would be unlikely to manage assumptions independent of source code. Sun Microsystems, for example, has developed a tool called JavaDoc that allows a Java developer to embed documentation comments in the code, which are then used to generate the API documentation in the form of completely linked HTML pages [Sun 04]. This is a user-friendly and non-disruptive mechanism for developers to create and update documentation in the form of structured comments within their source code. The Fluid project at the Carnegie Mellon University® School of Computer Science is another example. The project team has created an Eclipse-based tool that allows a developer to capture *design intent* in concurrent programs as annotations that are validated against the code using static analysis. The intent of the tool is to provide "early gratification" to the developer because the inclusion of annotations in the code using familiar terminology is rewarded with increments of assurance or warnings of model-code inconsistencies [Greenhouse 03].

Assumptions management as proposed in this technical note allows developers to record a vast range of assumptions in structured English. These assumptions are easily recorded as part of the implementation, and then extracted from the source code using a pre-processor. Recording assumptions in source code alone might prove invaluable, but extracting them into a searchable repository should allow system architects and lead designers to review more easily the assumptions of individual developers to determine if they are consistent with design and system assumptions. This repository can also later be used by configuration control boards in change analysis to determine more accurately the impact of a proposed change.

1.4 Assumption Types

Developers make many and varied assumptions as they are coding, involving the interpretation of requirements, availability of resources, and types of data. We have characterized assumptions into several types.

- control
- environment
- data
- usage
- convention

This list is by no means exhaustive and could easily be complemented by assumption types that are particular to a domain, an organization, or to the use of a particular technology. The Java examples in the following sub-sections are taken from the Assumptions Management System (AMS) prototype itself. The assumptions are recorded using our proposed XML structure:

® Carnegie Mellon University is registered in the U.S. patent and trademark office.

```
/*-
    <assumption>
        <type>
            Assumption type.
        </type>
        <description>
            Assumption description.
        </description>
    </assumption>
*/
```

1.4.1 Control Assumptions

Control assumptions capture expected control flow. For example, developers often assume the order in which various methods, often within a single class or subprogram, are invoked.

```
public String executeJob() {
    /*-
        <assumption>
            <type>
                CONTROL
            </type>
            <description>
                The initialization() method must be called
                before invoking this method.
            </description>
        </assumption>
    */
    .........
}
```

Figure 1: Example of a Control Assumption

A benefit of recording control assumptions such as the one in Figure 1 is that a software designer or architect can evaluate control assumptions to make sure they are consistent with the application flow. System maintainers can also review them before making changes so they do not inadvertently violate these assumptions.

1.4.2 Environment Assumptions

Recording environment assumptions captures what is expected of the environment in which the application will operate. For example, applications are often developed assuming a particular database product and version for data storage, as in Figure 2. However, it is possible that these applications will eventually migrate to later versions of the database or be modified to support a different database product altogether.[1]

[1] The assumptions in the example in Figure 2 do not need to be captured separately. It was the developer's choice to do so.

```
public class DBConnection {
    /*-
        <assumption>
            <type>
                ENVIRONMENT
            </type>
            <description>
                The database server is the PCAAR machine.
            </description>
        </assumption>
    */

    /*-
        <assumption>
            <type>
                ENVIRONMENT
            </type>
            <description>
                The database management system is Oracle 9i.
            </description>
        </assumption>
    */

    /*-
        <assumption>
            <type>
                ENVIRONMENT
            </type>
            <description>
                The database name is "seidb".
            </description>
        </assumption>
    */

    /*-
        <assumption>
            <type>
                ENVIRONMENT
            </type>
            <description>
                The Oracle Thin JDBC driver is available.
            </description>
        </assumption>
    */

        private Connection conn;
        private Statement stmt;

        private static final String URL_PREFIX =
              "jdbc:oracle:thin:@";
        private static final String DEFAULT_DBSERVERNAME =
              "pcaar.sei.cmu.edu";
        private static final String DEFAULT_PORTNO = "1521";
        private static final String DEFAULT_SID = "seidb";
```

Figure 2: Examples of Environment Assumptions

Another example of the use of environment assumptions occurs when a developer must make assumptions about the interface between the system and the user. In Figure 3, the `toString()`

method was intentionally developed for output in a text screen using fixed-width font.

```
public String toString() {
    /*-
        <assumption>
            <type>
                ENVIRONMENT
            </type>
            <description>
                Output of this function will be printed with
                fixed-width font.
            </description>
        </assumption>
    */

    String out = "";
    out = out + "  [Method:name   = " + name + "]\n";
    out = out + "    Return Type  = " + returnType + "\n";
    out = out + "    Arguments    = " + arguments + "\n";
    out = out + "    Parsing Info = [Ln: " + parsingInfo.line
          + ", Col: " + parsingInfo.column +
          ", endLn: " + parsingInfo.endLine + ", endCol: " +
          parsingInfo.endColumn +
          ", Level: " + parsingInfo.level + "]\n";
    out = out + "    Number of comments = " +
          commentList.size() + "\n";
    out = out + "    Comments:" + "\n";
    for(int i=0; i<commentList.size(); i++) {
            out = out + ((CommentInfo)
                  commentList.get(i)).toString();
    }
    return out;
}
```

Figure 3: Example of an Environment Assumption

Some benefits of documenting environment assumptions are below:

- A developer can capture the assumptions about the operational environment and communicate them to end users.
- A change analyst can evaluate the extent to which an implementation is dependent on an existing technology, product, or component version, and more accurately estimate the cost of a modification request concerning that component.
- A system integrator and/or system tester can trace a problem related to a particular component or try to understand why a certain method is throwing an exception or producing an error.
- A maintainer can determine what kind of environment is needed and assumed before looking at a piece of code in an application.
- A product manager can evaluate the effect of running this application in a different environment.

1.4.3 Data Assumptions

Data assumptions capture what is expected of input or output data. These assumptions are different from pre-conditions or post-conditions in that they do not correspond to specifications to which a developer must code, but rather conditions created by the developer under which a program will function correctly. Figure 4 illustrates a situation where even though a specific input into this method can consist of many paragraphs, the output will always be formatted to one paragraph so that it can be displayed properly on the screen.

```
private String trimDescription(String description){
    /*-
        <assumption>
            <type>
               DATA
            </type>
            <description>
                Even though the assumption description can
                have many paragraphs, it is OK to convert it
                to a single string of text with no line
                breaks.
            </description>
        </assumption>
    */

   String newDescription = "";

   //Tokenize the description using the line feed in the
   //description.

   StringTokenizer st = new StringTokenizer(description,"\n");

   while (st.hasMoreTokens()){
       newDescription =
          newDescription.concat(st.nextToken().trim() + " ");
   }

   … …
}
```

Figure 4: Example of a Data Assumption

Another use of a data assumption is to capture the way data is used in internal processing. In Figure 5, the developer creates a hash code from the assumption type and description strings. The developer assumes that the combination of the two strings will generate a unique hash code. In this particular case, we rejected this assumption during assumption review with the developers. Since it is common for developers to copy and paste pieces of code, two assumptions in different parts of the code might be identical and therefore would generate the same hash code.

```
// Generate a hash code to be used as a unique key.
/*-
    <assumption>
        <type>
            DATA
        </type>
        <description>
            The combination of the assumption type and the
            assumption description will create a unique string
            that is used as the assumption key.
        </description>
    </assumption>
*/

String hashString = assumptionInfo.type +
                    assumptionInfo.description;
assumptionInfo.hashValue = StringHasher.hashString(hashString,
                           hashString.length());
```

Figure 5: Example of a Data Assumption

Another way of using data assumptions involves assumptions about the nature of the data. The following example illustrates the assumption that data will always fall within a particular range. If developers assume data will always have certain characteristics, they can optimize their algorithms and/or data handling code, resulting in improved performance. It is necessary to record these assumptions and verify whether they are valid and consistent with other modules.

```
public static double compareLocation(int line1, int col1,
                                     int line2, int col2) {
    double res = line1 - line2;
    if(res != 0) return res;
      double res2 = col1 - col2;
    if(res2 == 0) return 0;
    /*-
        <assumption>
            <type>
                DATA
            </type>
            <description>
                A line contains fewer than 1,000,000
                characters.
            </description>
        </assumption>
    */
    return (res2/1000000);
}
```

Figure 6: Example of a Data Assumption

Some benefits of documenting data assumptions are below:

- A data assumption can point out vagueness in a requirement.

- Test cases can be created to determine what happens if the condition stated in the assumption is violated—the result might suggest a code modification to deal with the situation.
- The developer can record assumptions that allow him/her to optimize the code and reduce the complexity.

1.4.4 Usage Assumptions

Usage assumptions capture how an application is expected to be used. In the example in Figure 7, the developer is assuming that the application will execute from the command line with two required parameters.

```
public class Extractor {
    /*-
        <assumption>
            <type>
                USAGE
            </type>
            <description>
                Application will be run from the command line
                with two parameters: first parameter is a
                fully qualified folder name and the second
                parameter is a userid.
            </description>
        </assumption>
    */
    /*-
        <assumption>
            <type>
                DATA
            </type>
            <description>
                The existence of the folder name and userid
                entered from the command line are validated
                by the routine that uses them.
            </description>
        </assumption>
    */
    public static void main (String args[]) {
        FolderJob aFolderJob = new FolderJob();
        aFolderJob.initialize(args[1]);
        aFolderJob.executeAFolder(args[0]);
    }
}
```

Figure 7: Example of a Usage and a Data Assumption

Some benefits of documenting usage assumptions are below:

- A stakeholder or requirements analyst can verify whether the assumed usage is appropriate or corresponds to user expectations.
- Document writers can leverage assumption information as they generate end-user documentation.

1.4.5 Convention Assumptions

Convention assumptions capture the standards or conventions that the developer is following. In the example in Figure 8, the developer is assuming that assumptions are always recorded using XML W3C Recommendation, 2 May 2001 as the standard.

```
private int nextState (int curState, int curLevel,
                       int curType, Node p) {
    … … …
    /*-
        <assumption>
            <type>
                CONVENTION
            </type>
            <description>
                Assumptions are described using XML Schema
                W3C Recommendation, 2 May 2001.
            </description>
        </assumption>
    */
    case 20: … … …
```

Figure 8: Example of a Convention Assumption

Some benefits of documenting convention assumptions are below:

- Change analysts can look at the impact of changes in a standard or any type of convention.
- A reviewer can verify if the correct version of a standard or convention is being used.

Extensions for Other Types of Assumptions

As previously mentioned, the above list of assumption types is by no means exhaustive. This list could easily be complemented by assumption types that are particular to a domain, an organization, or to the use of a particular technology.

Quality attributes are a source of assumption types. For example, if security or performance is important for a specific application, all assumptions regarding these attributes could be labeled as SECURITY and PERFORMANCE respectively and validated by the experts in these fields.

2 Assumptions Management System

Our prototype AMS allows for the recording and extraction of assumptions from Java source code into a repository, and for the Web-based management of these assumptions.

Developers record assumptions as structured comments in Java source code using the XML structure presented in the previous section. The developer then uses the Assumption Extractor to find the assumptions in the code and record them in the Assumption Repository.

After the assumptions are stored in the Assumption Repository they are ready for validation using the Assumptions Management System. Validation in this case consists of marking an assumption as valid or invalid by a validator—a person who has been assigned to verify the appropriateness of assumptions of a certain type.

Project staff can use the Management System to browse or search for assumptions that fulfill given criteria. For example, a system architect can check consistency of all assumptions in a certain project, a certain package, or of a certain type.

Finally, the Management System maintains system and project information such as users, roles, projects, and assumption types. This information must be entered prior to using AMS and at the start of every project where AMS will be used. A system administrator creates users, roles, assumption types, and projects. A project manager assigns users and validators of each assumption type to the projects for which he or she has been assigned as manager.

The following sections will present portions of the analysis and design of AMS.

2.1 Use Cases

The assumptions management process implemented by the AMS prototype assumes the existence of four types of users, or roles:

- System Administrator: A system administrator can create, update, and delete projects, users, roles, and assumption types.
- Project Manager: A project manager assigns users to a project, assigns users to roles, and assigns roles for assumption creation[2] and validation.

[2] The current version of the AMS prototype focuses on assumptions embedded in the source code and therefore assumes the developer is always the creator. We envision these concepts can be applied to other phases of software development.

- Developer: A developer includes assumptions in the code at the time of writing and then uses AMS to extract assumptions from the code. At any point, the developer can use AMS to browse and search the assumptions extracted from any developer's code.
- Validator: A validator is a system designer, system architect, tester, domain expert, or any person that has the knowledge to determine whether an assumption made by a developer (or creator) is valid or invalid. A validator can also search through the assumption set, as part of the validation process, to look for inconsistencies.

A use case diagram for AMS is presented in Figure 9.

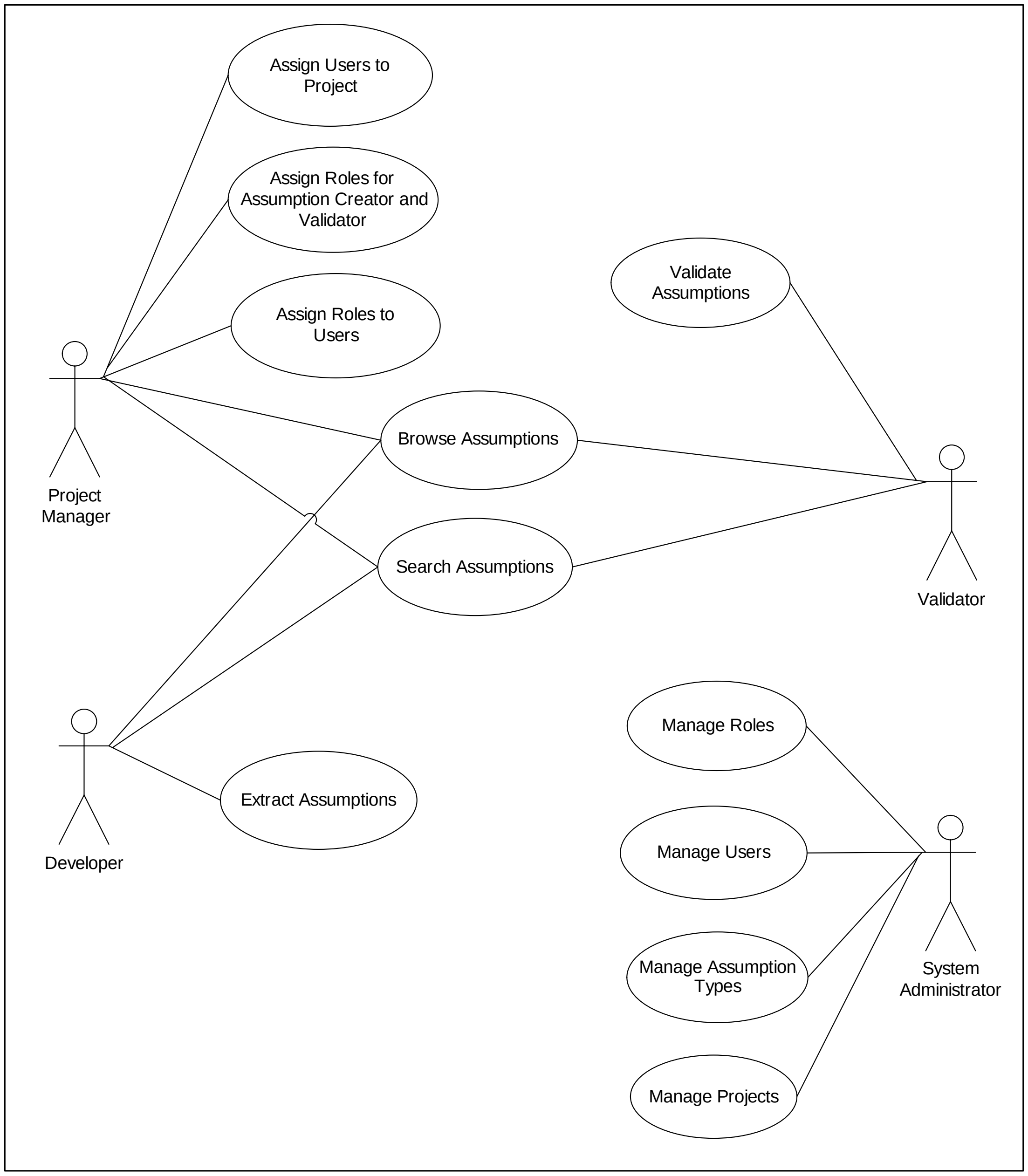


Figure 9: AMS Use Case Diagram

2.2 Architecture Description

2.2.1 Module View

AMS has two major components, implemented as separate applications: the Assumption Extractor and the Management System. A module view of the architecture is presented in Figure 10.

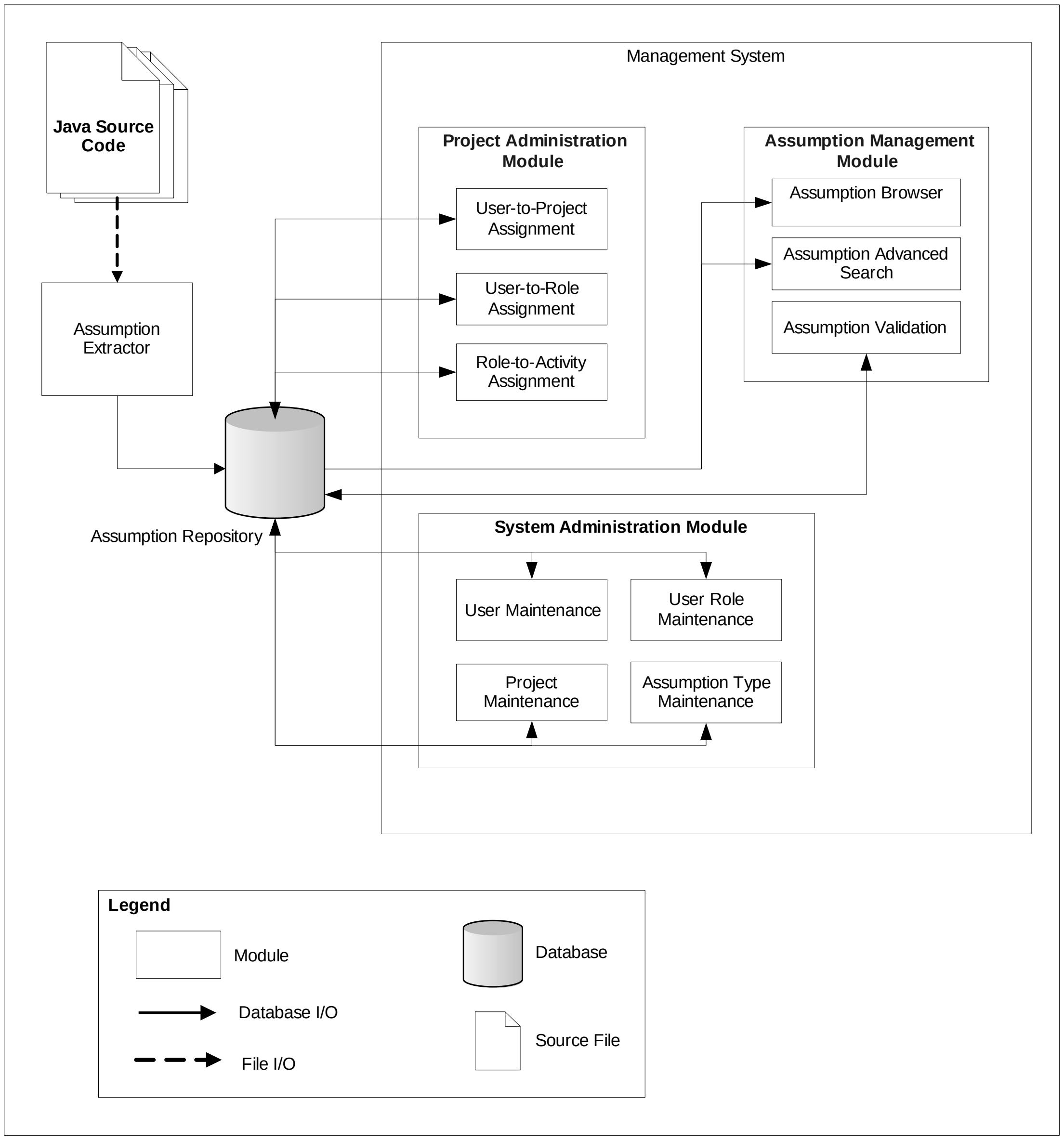


Figure 10: AMS Architecture—Module View

2.2.1.1 Assumption Extractor

The Assumption Extractor is a Java command-line application that runs on the developer's machine. The extractor takes a folder as one of its parameters and searches for Java source code (*.java) in that folder as well as in its sub-folders. It then parses each source file found to obtain the location of each recorded assumption, determined by package, class, and method; extract these assumptions; and store them into the Assumption Repository.

2.2.1.2 Management System

The Web-based Management System manages the assumptions stored in the Assumption Repository. It makes the assumptions available for browsing, validating, and searching. This application also includes administrative functions.

System Administration Module

Only the system administrator has access to this module that maintains system-level information: user maintenance, assumption type maintenance, user role maintenance, and project maintenance.

Project Administration Module

The project manager for each project has access to this module that maintains project-level information: user-to-project assignment, user-to-role assignment, and role-to-activity assignment. These assignments will affect the visibility, or permissions, that each user has over the assumptions stored in the database.

Assumptions Management Module

This module presents the assumptions stored in the system for a user to browse and/or validate, depending on his or her permissions. It also includes an advanced assumption search capability to search by assumption type, project, class, free-text, and so on.

2.2.2 Deployment View

AMS was built using a series of Web technologies, primarily Java-based, as shown in Figure 11.

- The Assumption Extractor is a Java program that runs on a client computer with a Java runtime environment.
- The Management System is implemented as a set of Java Server Pages (JSPs) executed by a Tomcat servlet engine and JavaBeans that are called by the JSPs.
- The Assumption Repository is implemented in an Oracle 9i database.

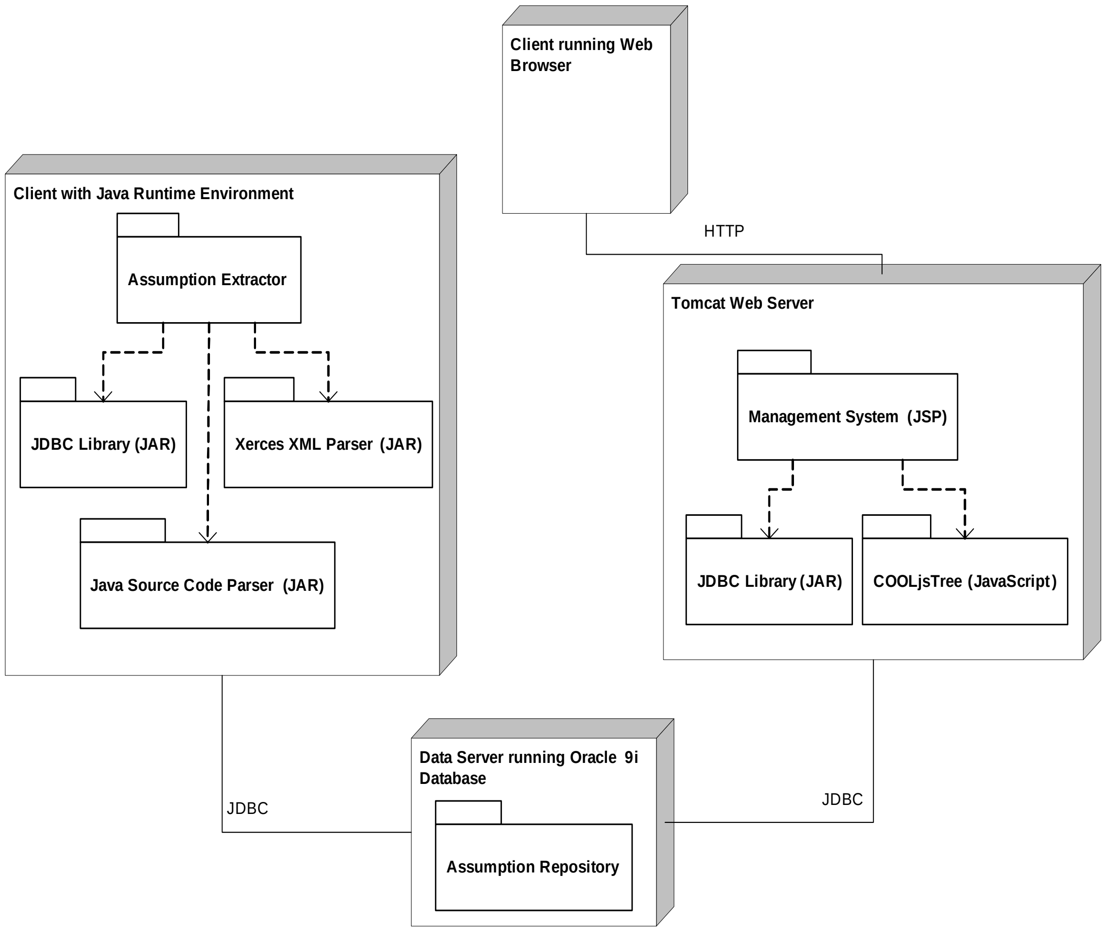


Figure 11: AMS Architecture—Deployment View

In addition to the major technology components, the system also makes use of a set of Java libraries:

- Oracle's Java Database Connectivity (JDBC) library (classes12.jar): the Assumption Extractor and Management System communicate with the repository using the JDBC library.

- Xerces XML library (xerces.jar): the Xerces library is used to parse the assumption content, which is recorded using XML.

- Java source code parser (query.jar): this Java package parses Java source programs into an internal tree structure that is then used to determine the location of each assumption. Location is determined by package, class, and method. The library is developed by Glen McCluskey & Associates LLC and can be downloaded at http://www.glenmccl.com/.

- Tree Display Utility (COOLjsTree): this Java Script library provides tree-like navigation menus. It is available free for evaluation, personal use, and non-profit at http://javascript.cooldev.com/scripts/cooltree/.

2.3 Assumption Repository

The database stores assumption information, project information, and system information. The database schema is shown in the entity-relationship diagram in Figure 12. Details of the tables can be found in Appendix A.

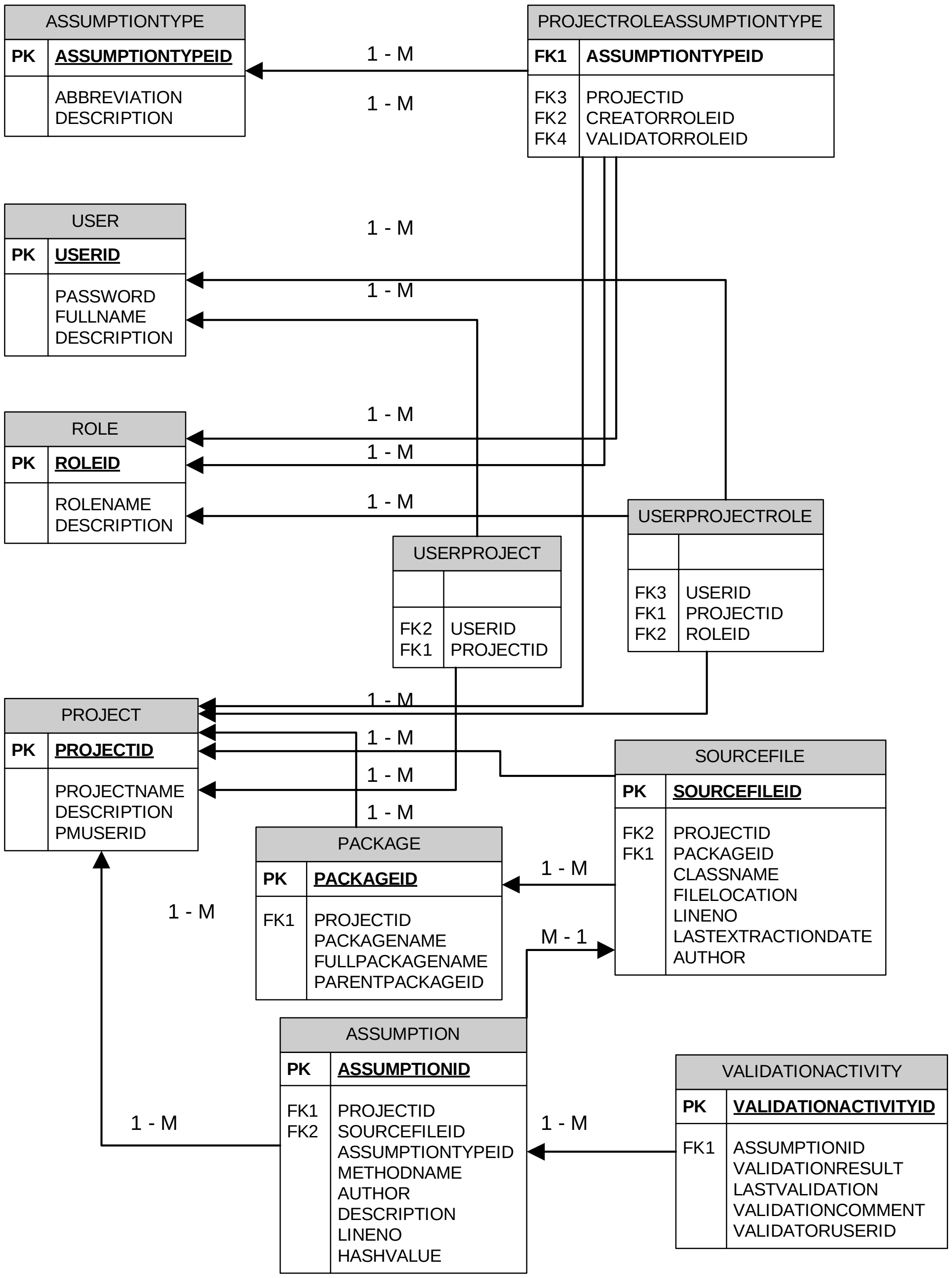


Figure 12: Entity-Relationship Diagram for the Assumption Repository

2.4 System Details

2.4.1 Assumption Extractor

The Assumption Extractor is a Java command-line application that traverses a specified folder and extracts assumptions from every Java source file. Package, class, and method information are also extracted to determine the location of each assumption. During the extraction process, progress and error messages are reported on screen and also sent to files named extractor.log and extractor_error.log.

The Assumption Extractor performs the following checks:

- Assumption types are valid (previously created in the system).
- User running the application has been registered in the system.
- For each assumption type, user running the application is allowed to create assumptions of that type.
- Every anomaly is recorded in the error log file.

The application takes three parameters as follows:

- Username: User name of the developer who is considered the assumption creator of the extracted assumptions.
- Project name: Project name to which the source code files belong.
- Root folder: Folder containing source files—the extractor looks through all its subfolders as well.

2.4.2 Management System

The Management System is a Web-based application that manages the assumptions that are stored in the repository. Its functionality is separated into three main modules.

2.4.2.1 System Administration Module

This module maintains system-level information, that is, information used by all projects in the organization. Only the system administrator (username: admin) has access to this module that contains the following functionality:

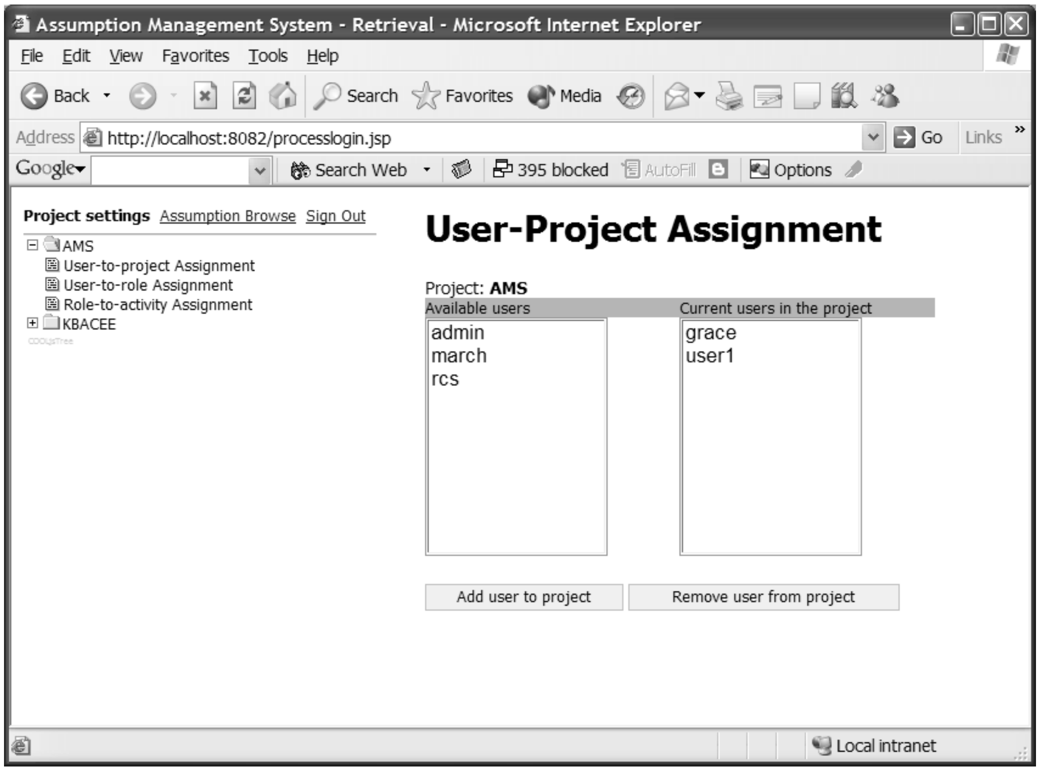
- User maintenance: adds, updates, and deletes system users and their associated information.
- User role maintenance: adds, updates, and deletes user roles. Roles group types of users, such as system architect, designer, developer, and tester. These roles later determine which users are allowed to create and validate assumptions for a specific project.

- Assumption type maintenance: adds, updates, and deletes assumption types that will be used by the organization using the system.
- Project maintenance: adds, updates, and deletes projects. The user who acts as project manager is also recorded. This allows the system to display only those projects assigned to the project manager.

2.4.2.2 Project Administration Module

This module maintains project-specific information. Only users who have been assigned as project manager of a project will be able to use this module. The left pane of Figure 13 shows the list of projects that are assigned to the user. The project manager can then perform the following tasks for each project:

- User-to-Project Assignment: This sub-module, shown in Figure 13, adds or removes users from the selected project.
- User-to-Role Assignment: This sub-module assigns one or more roles to each user in the selected project.
- Role-to-Activity Assignment: This sub-module assigns the roles that will be creators and validators of each assumption type in the project.


Assumption Management System - Retrieval - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8082/processlogin.jsp Go Links
Google Search Web 395 blocked AutoFill Options
Project settings Assumption Browse Sign Out
AMS
User-to-project Assignment
User-to-role Assignment
Role-to-activity Assignment
KBACEE
Add user to project
Remove user from project
Local intranet

User-Project Assignment

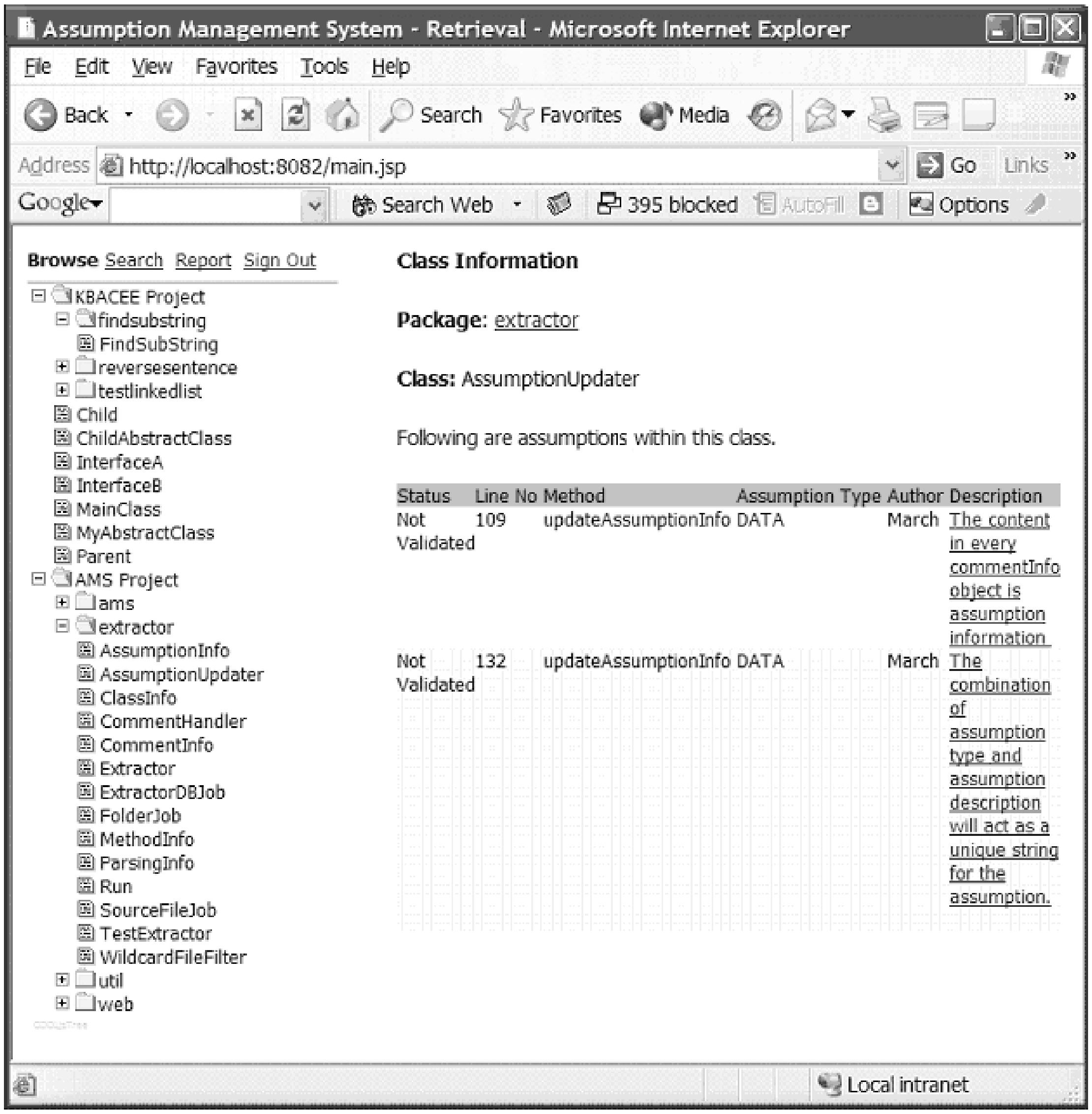
Project: **AMS**

Available users	Current users in the project
admin	grace
march	user1
rcs	

Figure 13: User-to-Project Assignment Screenshot

2.4.2.3 Assumption Management Module

This module retrieves the assumptions stored in the system for browsing, validating, and searching. The module will only present to the user the assumptions to which he or she has authorized access. Sets of assumptions can also be printed as a report.



Class Information

Package: extractor

Class: AssumptionUpdater

Following are assumptions within this class.

Status	Line No	Method	Assumption Type	Author	Description
Not Validated	109	updateAssumptionInfo	DATA	March	The content in every commentInfo object is assumption information
Not Validated	132	updateAssumptionInfo	DATA	March	The combination of assumption type and assumption description will act as a unique string for the assumption.

Figure 14: Assumption Management Screenshot

The screen is divided into two areas, as shown in Figure 14.

- Class list pane: The left side pane shows a list of classes grouped by project and package.
- Information pane: The right side pane displays the information for the element selected from the left pane.

- Package information: Lists the classes and sub-packages that are contained in the selected package. From this screen a user can select a class or sub-package and view more detailed information.
- Class information: Lists the assumptions in the selected class. From this screen a user can select an assumption to view more detailed information.
- Assumption Information: Shows the detailed assumption information along with any validation activity. When extracted, assumptions are labeled as *Not Validated*. Users designated as validators can validate the assumption by marking the assumption as *Valid* or *Invalid* and optionally include comments. The history of the validation process is maintained in the Assumption Repository.

3 Conclusions

Assumptions management is at the core of quality software engineering. Managing assumptions proved beneficial in the creation of AMS itself. Weekly meetings with the developers included the discussion of the assumptions they included in their code. We discovered that most assumptions that we marked as invalid corresponded to the interpretation of requirements. Because we were not available all the time to answer their questions, they would go ahead and make assumptions about what a specific requirement meant so they could continue their work. This is a situation not uncommon in software projects. In this case, documenting assumptions prevented the delivery of a system that would not satisfy its stakeholders.

Other assumptions were marked as invalid because of their potential for producing errors in the system—they were making invalid assumptions about the environment, control flow, data, and user interaction. In this case, documenting assumptions provided a way to detect defects before deployment.

The developers did not feel they were doing a lot of extra work by documenting these assumptions—"It's just like writing comments." The benefit that they found is that it is an easy way to get early feedback:—"It's good to know if you're on the right track and that you don't have to waste time fixing errors that have propagated all over the code because of wrong assumptions."

The prototype is fully functional and was very useful in strengthening our understanding of assumptions and their potential impact. Our experience indicates there are several enhancements that would increase its robustness and usability:

- The capability for extracting assumptions from programming languages other than Java
- A pop-up window or a template associated with a "hot-key" that would make it easier for the developer to record assumptions without worrying about formatting and XML
- An e-mail notification when assumptions are loaded into the repository, informing the validator that a set of assumptions is awaiting validation
- A graphical user interface for the Assumption Extractor instead of the current command line interface
- Implementation as a plug-in for an integrated development environment (IDE)

Even though assumptions management proved very useful in the coding phase of a project, we feel that from an interoperability perspective this is too late a stage for uncovering inconsistencies. To address interoperability requirements, the use of assumptions management would have to be moved to other activities and artifacts of software development, such as requirements analysis, architecture, and design. One of the many challenges we envision with this move is the recording of assumptions in view of the variability with which organizations gather requirements and design their systems. For this reason, the Integration of Software-Intensive Systems Initiative will unfortunately not pursue this topic further. However, we do strongly encourage the agile development and the maintenance and sustainment communities to pursue this topic further. In an agile development environment where you have short development iterations, assumptions management as described in this report has the greatest benefit because the timely feedback from the validation step of the process will influence subsequent iterations. For the maintenance and sustainment community, the Assumption Repository can provide valuable data for change impact analysis.

Appendix A: Table Description for the Assumption Repository

The Assumption Repository stores system information, assumption-related information, and project-related information. What follows is a list and short description of the tables that form the Assumption Repository.

System Tables

These tables store system information used by both the Assumption Extractor and the Management System.

Table 1: System Tables

Table Name	Description
`AssumptionType`	Assumption types available for all projects
`Project`	Projects
`Role`	User roles—a user can have several roles in a project
`User`	AMS users

Assumption-Related Tables

These tables store the information related to assumptions. Assumptions are grouped by class (or source file) and source files are grouped by packages. Validation results are also stored in a table.

Table 2: Assumption-Related Tables

Table Name	Description
`Assumption`	Assumption information extracted from Java source code files
`Package`	Java packages that group classes from which assumptions have been extracted
`SourceFile`	Java source code files (or classes) from which assumptions have been extracted
`ValidationActivity`	Validation results for each assumption

Project-Specific Tables

These tables store information related to each project. This information is used to determine which users in which roles can perform which activity in each project.

Table 3: Project-Specific Tables

Table Name	Description
`UserProject`	Users working on a project
`UserProjectRole`	Roles that each user is performing on a project
`ProjectRoleAssumptionType`	Roles that can create assumptions and roles that can validate assumptions on a project, per assumption type

References

[Bachmann 03]	Bachmann, F. et. al. *SEI Independent Research and Development Projects*(CMU/SEI-2003-TR-019 ADA418398). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. <http://www.sei.cmu.edu/publications/documents/03.reports/03tr019.html>.
[Garlan 95]	Garlan, D.; Allen, R.; & Ockerbloom, J. "Architectural Mismatch: or Why It's Hard to Build Systems Out of Existing Parts." *Proceedings of the International Conference on Software Engineering*. Seattle, April 23-24, 1995. New York, NY: ACM Press, 1995.
[Greenhouse 03]	Greenhouse, A.; Halloran, T.J.; & Scherlis, W. L. "Using Eclipse to Demonstrate Positive Static Assurance of Java Program Concurrency Design Intent." 99-103. *Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology Exchange*. Anaheim, California, Oct. 26-30. New York, NY: ACM Press, 2003.
[Lehman 00]	Lehman, M. & Ramil, J. "Software Evolution in the Age of Component Based Software Engineering." *IEEE Software 147,* 6 (December 2000) 249-255.
[MICRA 98]	Webster's Revised Unabridged Dictionary. MICRA, Inc. 1998.
[Parnas 71]	Parnas, D. "Information Distribution Aspects of Design Methodology." *Proceedings 1971* IFIP Congress. New York, NY: North Holland Publishing Company, 1971.
[Seacord 03]	Seacord, R. "Assumption Management." *news@sei interactive 6*, 1, First Quarter 2003. <http://interactive.sei.cmu.edu/news@sei./columns/the_cots_spot/2003/1q03/cots-spot-1q03.htm>.
[Sun 04]	Sun Microsystems. JavaDoc Tool. <http://java.sun.com/j2se/javadoc> (1994).

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.			
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE August 2004		3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Assumptions Management in Software Development			5. FUNDING NUMBERS F19628-00-C-0003
6. AUTHOR(S) Grace Lewis, Teeraphong Mahatham, Lutz Wrage			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213			8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2004-TN-021
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116			10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES			
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS			12B DISTRIBUTION CODE
13. ABSTRACT (MAXIMUM 200 WORDS) Software developers constantly make assumptions about the interpretation of requirements, design decisions, operational domain, environment, characteristics of input data, and other factors during system implementation. These assumptions are seldom documented and less frequently validated by the people who have the knowledge to verify their appropriateness. Additionally, the business, legal, and operating environments are always changing, as well as the software itself, rendering previously valid assumptions invalid. This technical note explores assumptions management as a method for improving software quality. This exploration covers assumptions management concepts, results of work on a prototype Assumptions Management System, conclusions, lessons learned, and potential work in this area.			
14. SUBJECT TERMS Assumptions Management System, AMS, Assumption Extractor, Assumption Repository			15. NUMBER OF PAGES 36
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18 298-102