

Evaluator: Cloud-Based Software for Collaborative Evaluation

Gökçe Önen

Thesis submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Osman Balci, Chair
Reza Barkhi
Eli Tilevich

November 30, 2022

Blacksburg, Virginia

Keywords and phrases: Cloud software engineering, collaborative evaluation, evaluation methodology, qualitative assessment

Evaluator: Cloud-Based Software for Collaborative Evaluation

Gökçe Önen

ABSTRACT

Evaluation is a process of investigating a project's implementation and results methodically and objectively. The evaluation problems in industry are extremely complicated and call for the examination of numerous qualitative and quantitative variables. Evaluators from different disciplines examine these variables. In order for the evaluators to collaborate during a broad evaluation process, there is a requirement for a collaborative evaluation methodology. In this research, a methodology for evaluation of any general application was implemented which is initially developed as part of a research effort supported by the Office of Naval Research. This technique is based on identifying the indicators to be applied in the evaluation of complex projects. In order to decrease complexity, these indicators are organized hierarchically which is modeled after the Analytic Hierarchy Process method. While at the top layer of this hierarchy is the subject we want to evaluate, towards the lower layers it is divided into more and more simple indicators. Therefore, the evaluators are responsible for assessing the project in terms of only the most simplified indicators. As a result, a precise evaluation which is not subject to human bias is accomplished. We have created a web-based application called Evaluator which allows the users to execute this methodology step by step. It simplifies the process by providing a user-friendly and easily accessible cloud-based platform in accordance with today's user experience principles. Although we have used this method and the software to evaluate projects, it is a tool that provides general usage for the assessment of a broad range of systems from various contexts.

Evaluator: Cloud-Based Software for Collaborative Evaluation

Gökçe Önen

GENERAL AUDIENCE ABSTRACT

Evaluation of a project is a process of systematic and objective assessment using a certain methodology. The evaluation problems we encounter in industry are quite complicated. This is due to the fact that a project must be evaluated in light of the numerous indicators it contains. Therefore, the project being assessed must be divided into these indicators as the first stage in the evaluation process. This layered form of the problem creates a hierarchical structure. Because the people who take part in evaluation are only capable of handling problems up to a certain level of complexity due to human nature. Therefore, it is essential to simplify problems to the point that those evaluating them can come to informed determinations. The strategy employed in this study makes it feasible to assess the primary subject of the evaluation effectively by limiting the evaluators to assessing only the most simplified indicators at the bottom of this hierarchy. This is the only task that the people in the evaluation process are responsible for. As a result, an accurate assessment not subject to human bias of error is achieved. We have created a web-based application called Evaluator which allows users to carry out this method's procedures. The user-friendly user interface of Evaluator makes it straightforward for people to take part in this complex process. Though we have used this approach and the software to assess projects, it is a flexible software which can be applied broadly for the assessment of a wide variety of systems from different contexts.

ACKNOWLEDGMENTS

I would like to thank my thesis advisor Dr. Osman Balci for his invaluable support and suggestions from the very beginning. He has been a tremendous inspiration for me with his diligence and enthusiasm. His sympathy, patience, and thoughtfulness helped me to stay on track through challenging times. I left each one of our meetings motivated and full of optimism for the future. I will always be grateful for his efforts to provide me with insightful knowledge and encourage me improve myself.

I am grateful to my committee members Dr. Reza Barkhi and Dr. Eli Tilevich for their interest in and support on my thesis.

I owe a great gratitude to my dear parents and precious sister, Meral Önen, Hamza Önen, and Elvan Erol for having trust in me and standing by me through every phase of my life. I am grateful for their perseverance and outstanding efforts to lessen my troubles during the thesis period.

TABLE OF CONTENTS

ABSTRACT.....	ii
GENERAL AUDIENCE ABSTRACT	iii
ACKNOWLEDGMENTS.....	iv
LIST OF FIGURES.....	vii
LIST OF ACROYNMS	viii
Chapter 1: Problem Definition and Overview.....	1
1.1 Introduction.....	1
1.2 Statement of the Problem.....	2
1.3 Statement of the Objectives.....	2
1.4 Overview of the Thesis.....	2
1.5 Summary of Contributions.....	2
Chapter 2: Background	4
Chapter 3: Evaluator Architecture.....	5
3.1 Hardware and Software Development Environment.....	5
3.2 Architecture	5
3.2.1 Tier 1: Client	6
3.2.2 Tier 2: Web.....	6
3.2.2.1 Template.....	7
3.2.2.2 User Account.....	7
3.2.2.3 Super User.....	7
3.2.2.4 User File.....	7
3.2.2.5 Project.....	7
3.2.2.6 Evaluator	7
3.2.2.7 Score Set.....	7
3.2.3 Tier 3: Business	8
3.2.3.1 Entity Beans.....	8
3.2.3.2 Façade Beans.....	8
3.2.3.3 Controllers	8
3.2.3.4 Managers	9
3.2.3.5 Validators	9
3.2.3.6 Global.....	10
3.2.4 Tier 4: Data Mapping.....	10
3.2.5 Tier 5: Data Source.....	10
Chapter 4: Evaluator Design.....	11
4.1 Functionality Design.....	11
4.1.1 Database Design.....	11
4.1.2 User Account Design.....	12
4.1.3 Evaluation Project Design	13
4.2 Implementation Design.....	13
4.2.1 Package Design.....	13
4.2.2 Class Design	14
4.2.2.1 Entity Beans.....	14

4.2.2.2	Facade Beans.....	15
4.2.2.3	Controllers	16
4.2.2.4	Managers	18
Chapter 5: Evaluator Functionality.....		20
5.1	<i>Account Creation.....</i>	<i>20</i>
5.2	<i>User Sign-In.....</i>	<i>22</i>
5.3	<i>Password Change.....</i>	<i>23</i>
5.4	<i>Account Recovery.....</i>	<i>25</i>
5.5	<i>Project Creation.....</i>	<i>25</i>
5.5.1	Project Administrators	28
5.5.2	Project Evaluators	29
5.5.3	Creation of Indicators Hierarchy	30
5.6	<i>Evaluation.....</i>	<i>46</i>
Chapter 6: Evaluator Self-Assessment.....		49
6.1	<i>Evaluations of Leaf Indicators</i>	<i>55</i>
6.1.1	Complexity.....	55
6.1.2	Ease of Change	55
6.1.3	Readability.....	56
6.1.4	Traceability	56
6.1.5	Cohesion.....	56
6.1.6	Coupling.....	56
6.1.7	Well-Defined Interfaces.....	56
6.1.8	Early Error Detection	56
6.1.9	Visibility Behavior	56
6.1.10	Disk Storage Efficiency.....	56
6.1.11	Execution Efficiency.....	56
6.1.12	Memory Efficiency	57
6.2	<i>Evaluations of Branch Indicators</i>	<i>57</i>
6.3	<i>Evaluation of Software Quality.....</i>	<i>58</i>
Chapter 7: Conclusions and Future Research		59
7.1	<i>Conclusions.....</i>	<i>59</i>
7.2	<i>Future Research</i>	<i>59</i>
REFERENCES		60

LIST OF FIGURES

Figure 1. Evaluator Architecture.....	6
Figure 2. Database ER Diagram	12
Figure 3. UML diagram of Entity Beans	14
Figure 4. UML Diagram of Façade Beans.....	15
Figure 5. UML Diagrams of Controllers	16
Figure 6. UML Diagrams of Controllers	17
Figure 7. UML Diagram of Dependencies of Controllers.....	18
Figure 8. UML diagram of Managers.....	19
Figure 9. Account Creation Interface.....	21
Figure 10. User Sign-In Interface	22
Figure 11. TFA Interface	23
Figure 12. Password Change Interface	24
Figure 13. Password Change Interface	24
Figure 14. Password Change Interface	25
Figure 15. Account Recovery Interface.....	26
Figure 16. Account Recovery Interface.....	26
Figure 17. Project Creation Interface	27
Figure 18. Administrator’s Evaluation Projects	28
Figure 19. Evaluator’s Evaluation Projects	29
Figure 20. Indicators the Evaluator is Assigned to.....	30
Figure 21. Part of the Indicators Hierarchy of Software Quality Evaluation	31
Figure 22. Overview of the Adaptability indicator	32
Figure 23. Weights of the Hierarchical Decomposition Indicator	33
Figure 24. Editing Pairwise Comparison of Complexity and Cohesion	34
Figure 25. AHP Consistency Results	35
Figure 26. Assigning an Evaluator to a Leaf Indicator	36
Figure 27. Removing an Evaluator from a Leaf Indicator.....	37
Figure 28. Assigning One of the Score Sets to a Leaf Indicator.....	38
Figure 29. Some Nominal Score Set Examples.....	39
Figure 30. Scores of the Complexity indicator	40
Figure 31. Notes of the Complexity indicator	41
Figure 32. Evaluation Status Interface	42
Figure 33. Evaluation Status Interface	43
Figure 34. Evaluation Status Interface	44
Figure 35. A Page from a Project Report.....	45
Figure 36. Evaluator Selects a Score	47
Figure 37. Evaluator Score and Notes	48
Figure 38. Project Documentation of Software Quality Evaluation.....	49
Figure 39. Indicators Hierarchy of Software Quality Evaluation.....	50
Figure 40. Evaluators Assigned to the Complexity Indicator	51
Figure 41. Excellent to Unacceptable Score Set Selected for Readability.....	52
Figure 42. Weights of Main Indicators on Software Quality.....	53
Figure 43. Weights of Assigned Evaluators on Cohesion.....	54
Figure 44. Evaluation of Leaf Indicators	55
Figure 45. Computed Score of the Maintainability Indicator	57
Figure 46. Software Quality Score of Evaluator.....	58

LIST OF ACROYNMS

AHP	Analytic Hierarchy Process
API	Application Programming Interface
CDI	Context and Dependency Injection
CSS	Cascading Style Sheets
EJB	Enterprise Java Beans
EL	Expression Language
Jakarta EE	Jakarta Enterprise Edition
JDBC	Java Database Connectivity
JPA	Java Persistence API
JSF	JavaServer Faces
OOD	Object-Oriented Design
PDF	Portable Document Format
PF	PrimeFaces
RDBMS	Relational Database Management System
SHA-1	Secure Hash Algorithm 1
SSD	Solid State Drive
SQL	Structured Query Language
TFA	Two-Factor Authentication
UI/UX	User Interface/User Experience
UML	Unified Modeling Language
XHTML	EXtensible HyperText Markup Language

Chapter 1: Problem Definition and Overview

1.1 Introduction

Evaluation of a project is the process of systematically and objectively investigating its implementation and outcomes. Evaluation problems encountered in industry are very complex and require consideration of many qualitative and quantitative factors. The measurement of these factors is accomplished by combining several assessment techniques including examination by experts from different fields. There is a need for a collaborative evaluation methodology for experts to work together in a broad assessment process.

The evaluation methodology was originally developed for assessment of any generic application as part of a research program supported by the Office of Naval Research (ONR). This methodology is based on the identification of indicators to be used in the assessment of complex evaluation problems. These indicators form an acyclic graph and are in a hierarchical structure to reduce complexity. In the higher levels of this hierarchy, there are complex indicators with many factors to be considered in order to be evaluated. In the lower levels, more complex indicators at higher levels branch off and are represented by simpler child indicators. These indicators which have at least one child indicator and at least one parent indicator are called branch indicators. At the lowest level of the hierarchy are indicators that are the simplest and easy to evaluate by experts, called evaluators in the terminology of this methodology. These indicators which has no child indicators are called leaf indicators [Balci *et al.* 2002]. In the evaluation step, multiple evaluators contribute to the evaluation of the whole project by scoring only the leaf indicators to which they are assigned according to their field of expertise. Thus, the complexity of the evaluation is overcome, and it becomes convenient for collaborative evaluation.

Analytic Hierarchy Process (AHP) is a method to organize and simplify complex decision-making processes [Saaty 1980]. The hierarchical structure of the indicators in the evaluation methodology was created using this algorithm. Accordingly, the degree of influence of each indicator's sub-indicators on their parent indicator is determined by making pairwise comparisons. Thus, while the scores in the leaf indicators are propagated towards the root indicator, the weight of the indicators that will affect the upper levels is determined. The same weighting method is also used to determine the degree of effectiveness of different evaluators that score the same leaf indicator. Hence, it is ensured that the scores given to the leaf indicators are transferred accurately up to the root indicator.

In the implementation of the methodology, we have developed a web-based software to enable collaborative evaluation. The software is currently available for use at <https://shark.cs.vt.edu/Evaluator/>.

Cloud computing is the provision of services such as server, database, data storage, and networking through the Internet. We developed the application as a cloud-based software because of its flexibility, scalability, security, and cross platform functionality. Having software accessible via the cloud enables its use under a web browser on any device with Internet connection including mobile, desktop, or laptop computers.

In this thesis, Evaluator is designed, architected, and implemented as a cloud-based application. We modified the AHP algorithm to make it compatible with the evaluation methodology. We created a reliable system that facilitates user account creation, evaluation project creation, project evaluation and project report generation. Evaluator may be used on any platform that supports a web browser. With the aid of cloud software capabilities, user account data is synced between all various devices. Hence, all administrators and evaluators of a project can retrieve historical data within their privileges and continue to contribute to the evaluation process.

1.2 Statement of the Problem

A software tool that implements the Evaluation methodology was previously developed as a desktop and a web-based software under the leadership of Prof. Osman Balci at Orca Computer, Inc in 2002. Although the Evaluation methodology aims to simplify an evaluation process, it is still a complex method for both project administrators and evaluators to learn and implement. In this research, we aimed to simplify this process by developing a user-friendly and easily accessible cloud-based software application in accordance with today's user experience principles.

1.3 Statement of the Objectives

The objectives that are aimed to be achieved by this research are as follows:

1. Develop the Evaluator software as a cloud-based software application pursuing the modern user experience standards.
2. Allow all project administrators and evaluators to access the Evaluator software from any Internet-connected tablet, laptop, or desktop computer at any time and from any location.
3. Enable project administrators to create an evaluation project, create the hierarchy of indicators, assign the evaluators and score sets to the leaf indicators, evaluate the overall project by propagating leaf indicator scores, and generate evaluation project reports.
4. Enable project evaluators to evaluate the leaf indicators assigned to them by the project administrators.

1.4 Overview of the Thesis

The overview of the thesis is as follows: Background and early work of Evaluator are described in Chapter 2. The five-tier design of Evaluator is introduced in Chapter 3. Chapter 4 outlines the functionality and implementation designs for the Evaluator. Chapter 5 introduces all of the Evaluator's new capabilities and functionalities. In Chapter 6, Evaluator is self-evaluated using 13 software quality criteria. Conclusions and ideas for future research are proposed in Chapter 7.

1.5 Summary of Contributions

The following list outlines the contributions of this thesis:

1. The Evaluator software was redeveloped as a cloud-based software application pursuing the modern user experience standards.
2. The software is currently available for the use at <https://shark.cs.vt.edu/Evaluator/>.
3. All project administrators and evaluators are enabled to access the Evaluator software from any Internet-connected tablet, laptop, or desktop computer at any time and from any location.
4. Project administrators are enabled to create an evaluation project, create a hierarchy of indicators, assign evaluators and score sets to the leaf indicators, evaluate the overall project by propagating leaf indicator scores, and generate evaluation project reports.
5. Project evaluators are enabled to evaluate the leaf indicators assigned to them by the project administrators.
6. The AHP algorithm is modified to make it compatible with the evaluation methodology.
7. Evaluator offers elastic scalability since it provides easy modification, testing and deployment without reinstalling or other user-side procedures.
8. Use of an industry encryption algorithm protects user confidential information.
9. Synchronization upon different devices is provided.
10. Operation complexity and the data source storage requirements is reduced
11. Users are provided a secure and customizable account.

Chapter 2: Background

The original research leading to the development of the evaluation methodology used in this thesis dates back to late 1980s. During the period of 1992-1995, an evaluation methodology was developed at Virginia Tech under funding from the Naval Surface Warfare Center Dahlgren Division (NSWCDD) Engineering of Complex Systems (ECS) research program funded by the Office of Naval Research (ONR) [Talbert 1995].

During Jan. 1997 – Sept. 1998, a software tool, called Evaluation Environment (EE), was developed at Orca Computer, Inc. under the NSWCDD ECS funding to enable the application of the methodology. During Oct. 1998 – Sept. 1999, the EE methodology and software were used for the National Missile Defense (NMD) work conducted by NSWCDD and further improved at Orca Computer, Inc. under the NMD / NSWCDD funding. During Oct. 1999 – Dec. 2002, the EE methodology and software were used for the Ground-based Midcourse Defense (GMD) work conducted by NSWCDD.

The EE software was a shrink-wrapped product usable on a personal computer and prevented collaborative evaluations. During June 2001 – Aug. 2002, under funding from the Navy's DD21 program, Orca Computer, Inc. developed the secure web-based version of the EE software to enable collaborative evaluations by people who are geographically dispersed. During Sept. 2002 – Sept. 2003, the EE methodology and web-based software were used for the Missile Defense Agency (MDA) Battlespace Environment and Signatures Toolkit (BEST) verification, validation, and accreditation evaluations.

The EE methodology and software have been used in many projects including the following: advanced gun system software requirements credibility assessment, NMD system performance evaluation, NMD deployment readiness review (DRR), LIDS modeling and simulation application acceptability assessment for accreditation, assessment of accreditation status of the NMD core baseline models, simulations and test beds for the DRR support, NMD system interoperability assessment, NMD system design evaluation based on government verification by analysis plan, software acceptability assessment for certification, document quality assessment, BEST configuration management plan evaluation, and BEST master test plan evaluation. The EE methodology was published for its application to certification of models and simulations [Balci 2001].

Based on our work in recent years, we have evolved the EE methodology. This thesis describes the cloud software application that was developed for the application of the EE methodology to enable collaborative evaluations by geographically dispersed people.

Chapter 3: Evaluator Architecture

Evaluator is built upon the Jakarta (Java Platform) Enterprise Edition (EE) client-server architecture. In accordance with this architecture, the application has five tiers, namely, “Client Tier, Web Tier, Business Tier, Data Mapping Tier, and Data Source Tier” [Balci 2022]. The way these tiers communicate with each other enables the Evaluator to function effectively and reliably.

3.1 Hardware and Software Development Environment

Evaluator is developed using the Jakarta EE platform. It provides numerous useful Application Programming Interfaces (API) for software development. Expression Language (EL), JavaServer Faces (JSF), Context and Dependency Injection (CDI), CDI-managed Beans, Enterprise Java Beans (EJB), Java Persistence API (JPA), JPA Façade Beans are some of the Jakarta EE APIs which we utilized for the development of Evaluator. The front-end development tools we use for the Client Tier of the Evaluator are EXtensible HyperText Markup Language (XHTML), Cascading Style Sheets (CSS), and PrimeFaces (PF).

To meet the hardware and software requirements by the environment mentioned above, Evaluator has been deployed to a server computer with the following features:

- Hardware
 - PowerEdge T330 Server Computer
 - 480 GB Solid State Drive (SSD)
 - 64 GB RAM
- Software
 - MySQL Relational Database Management System (RDBMS)
 - CentOS Linux Operating System
 - Jakarta EE
 - WildFly Jakarta EE Application Server

The software is currently available for the use at <https://shark.cs.vt.edu/Evaluator/>.

3.2 Architecture

The Evaluator architecture is built upon the Jakarta EE client-server architecture, that is composed of “the client tier, web tier, business tier, data mapping tier, and data source tier” [Balci 2022]. The structure supplied by the Jakarta EE platform enables software developers to implement application logic with ease. The separation ensures flexibility in modifying each tier independently of the entire application. Figure 1 presents the architecture of Evaluator. The purposes of all tiers are explained in the sections below.

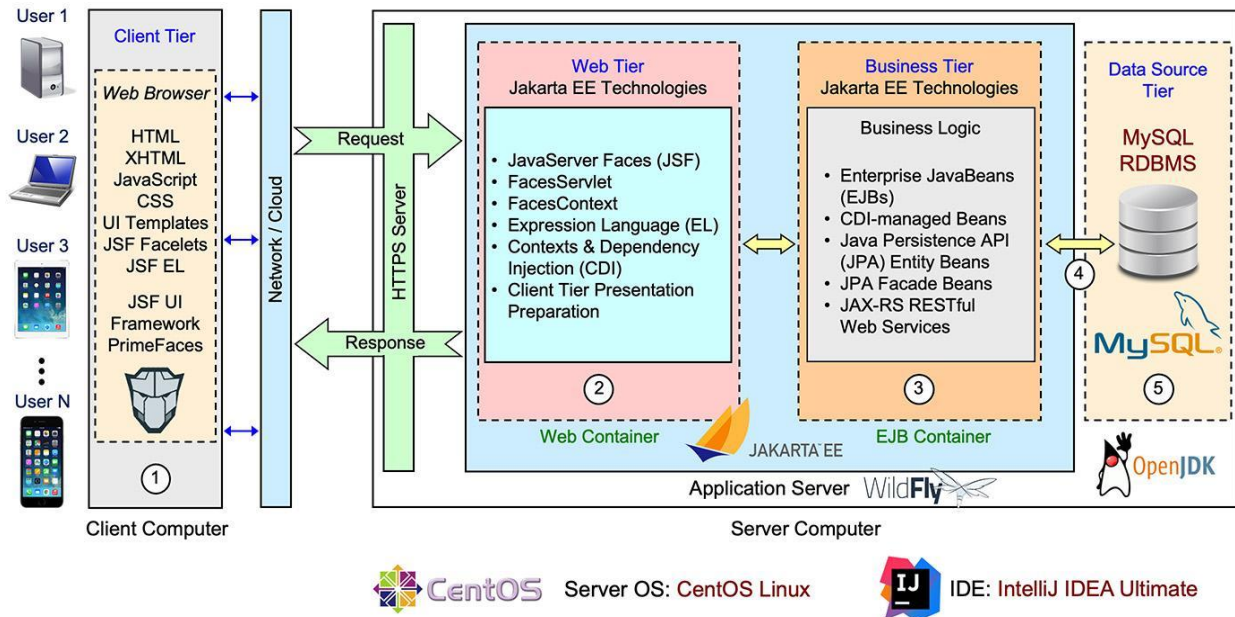


Figure 1. Evaluator Architecture [Balci 2022]

3.2.1 Tier 1: Client

The application clients that access the Jakarta EE server are part of the client tier and are often resides on a separate computer from the server [Oracle 2018]. A client may be a web browser, another server machine, or a desktop program. Evaluator is an application which serves clients through a web browser.

Clients interact with the application through the client tier. User interface in the client tier of Evaluator is primarily created with XHTML, CSS, and PF which is an open-source UI component library for JSF [PrimeFaces 2022].

3.2.2 Tier 2: Web

CDI and JSF enable the web tier to handle the interaction between client and business tier. CDI's services enable the use of Enterprise JavaBeans in Jakarta EE web applications. Thus, there is no need to create factory classes to create entity objects to be used in the web applications. In this way, for example, a session bean can be instantiated and stored in the EJB container during runtime. This object can be used inside an EJB without any initialization.

The connection between EJB and the web tier is handled by the EL. It provides access to data obtained from beans in business tier within XHTML tags. It ensures that actions such as reaching a value from backing beans or calling Java methods can be performed as a result of an interaction in the user interface. The request-response mechanism in this client-server communication is established by the JSF technology. The JSF pages in Evaluator are introduced below focusing on their features.

3.2.2.1 Template

Template contains consistent elements of the application such as header, footer, session timeout, and growl messages. All modules in the application are placed within this fixed structure. In terms of reusability, using a single template which includes all these structures that repeatedly take place in the application is essential.

3.2.2.2 User Account

Functions related to user accounts including account creation, user sign-in, and password changing are offered by the user account. Functionalities such as profile editing, Two-Factor Authentication (TFA), account recovery, and account deletion are all included in this component.

3.2.2.3 Super User

Evaluator has a single super user. This user can list all other users and access information about them. It can also access information of all projects created by other users in the application. These functionalities are provided by the super user component.

3.2.2.4 User File

The user file provides a space for users to upload and store their files. Users can display the image or other viewable files and play video files. This component allows the users to download or delete the files as well.

3.2.2.5 Project

The project component allows users to create a new evaluation project, and view, edit or delete existing projects. In addition, it allows the administrators to open existing projects. Thus, they can create the indicators hierarchy of the projects and edit all the details about indicators such as assigning evaluators, selecting score sets, making pairwise comparisons, and propagating scores. The evaluation of the project by the administrators is completed through this component.

3.2.2.6 Evaluator

The evaluator component allows evaluators to see the projects they are assigned to. Thus, they can select one of these projects and view their indicators hierarchy. Evaluators can score the project in terms of an indicator they select among the indicators assigned to them from the indicators hierarchy and write their notes on the score they give.

3.2.2.7 Score Set

Evaluators can use various score sets in the evaluation of a project indicating their criteria for scoring a particular indicator such as degree of agreement, probability of occurrence, or degree

of reality. The score set component allows all users to view which nominal scores the various score sets contain and which numeric score ranges correspond to these nominal scores.

3.2.3 Tier 3: Business

The main functionality of an application is created on the business tier. Several technologies in this tier provided by the Jakarta EE platform are utilized in Evaluator. One of them is EJB which passes data to the web tier by the help of Expression Language as discussed in 3.2.2. The primary business logic of Evaluator is developed using JPA in conjunction with EJB. The responsibility of the JPA is storing and retrieving data in the business tier. The business tier can access the database's stored data by inserting Structured Query Language (SQL) queries into Java code. Performing database operations such as accessing, storing, deleting, and modifying data in business tier is the responsibility of the entity manager. Therefore, EJB and JPA are used to provide all business logic.

Business tier determines the base logic of Evaluator through the following tasks. It receives the data from the user interface, performs the necessary operations on this data and saves it to the database. It also accesses the data in the database, reformats it and renders it to the frontend. Therefore, it is essential for the Evaluator to work properly for the operations in this tier to be carried out effectively.

3.2.3.1 Entity Beans

One of the most fundamental parts of the business tier are entity beans. Their purpose is to represent objects corresponding to tables in the database. These objects are manipulated by upper layer classes according to Object-Oriented Design (OOD).

3.2.3.2 Façade Beans

The façade beans are used to construct the business tier's database operations. Façade beans provide simpler-to-use APIs such as the create, read, update, and delete functions in order to encapsulate the complicated database operations [Schmidt *et al.* 2013]. In the business tier, façade beans participate in a sub-bottom layer.

3.2.3.3 Controllers

The top layer component in business tier is controllers. They leverage database operations provided by facade beans. They transmit data from entity beans to web tier using EJB technology and provide methods to be called from web tier. They can also implement complex business logic using CDI injections. The user controller, for instance, offers all features related to the user accounts including account creation, user sign-in, and profile updating. There are ten more different controllers for manipulating the logic of different functionalities of Evaluator. Lower coupling and higher cohesion can be attained by dividing the application into several controllers [Eder *et al.* 1994].

3.2.3.4 *Managers*

Managers form an advanced structure by leveraging multiple components. Like controllers, they can correspond to various entity beans to implement complex business logic related to them by utilizing the operations in Façade Beans as well. However, they are not as strict as controllers as described in 3.2.3.3. Below some managers developed for Evaluator are introduced.

All the login logic is handled by the login manager. The login manager first determines if the username is present in the database after getting the login information from the frontend. If it is, it uses SHA-1 (Secure Hash Algorithm 1) [Eastlake and Jones 2001] to hash the password in order to validate it. The login manager will log the user in if all the information can be matched.

The password reset manager consists of three steps. First, it receives the username from the user and determines whether it exists in the database. If the username is recognized, the user enters the response to the security question for the associated account. The password manager requests the new password from the user twice to make sure that the user has typed the intended password if the entered response matches the answer in the database. If these two passwords match, the new password is hashed with the SHA-1 algorithm and stored in the database.

The account recovery manager works similarly to the password reset manager to bypass TFA for the cases where a user no longer has access to the email address, or the phone number provided for TFA. First, it requests the username and password from the user. It verifies the username exists in the database and the password matches the one in the database. After this step, the security question of that user is displayed. If the user's answer to this question is correct, they are permitted to log in.

The report manager helps to generate a report of the evaluation projects. Briefly, it takes all the data of the selected project and summarizes it by formatting it under title, description, indicators hierarchy, and indicator details sections. As a result, the project report is generated in Portable Document Format (PDF) format.

The file download manager gets the name, path and file type of the file to be downloaded. After receiving this information, it creates a file stream ready to be downloaded.

The file upload manager creates a new file name for the file to be uploaded by attaching the username to the original file name. It checks if there is already a file with that name. If it exists, the file upload manager deletes this file to replace it with the new file. Then, it writes the stream of the uploaded file to the database with this new file name.

3.2.3.5 *Validators*

Validators offer tools for determining if user input is accurate. There are four validators in Evaluator: a zip code validator, an email validator, a password validator, and a username validator. Each of them is used to validate user input and safeguard the entire application from input validation attacks.

3.2.3.6 *Global*

All the frequently used constant values, passwords, and methods, including growl message display components and password encryption, are contained in global. Every class in the project can access these values and call the global component's functions.

3.2.4 *Tier 4: Data Mapping*

Between the business tier and the data source tier, the data mapping tier serves as a linking mechanism. Java Database Connectivity (JDBC) and JPA are two APIs that Jakarta EE offers for data mapping. The link between the SQL databases Java programming language is supported via JDBC. JDBC must be active in order to keep the application server running. To construct database tables, developers can use JDBC to run SQL queries. JPA offers a method for producing entity classes by inspecting RDBMS tables.

Evaluator stores the data of registered users and evaluation projects. The entity classes corresponding to them are created by the Persistence API. Retrieving data from the database and executing SQL queries in entity manager are provided by Persistence API which constructs session beans. Jakarta EE offers all tools needed to manage data mapping in Evaluator.

3.2.5 *Tier 5: Data Source*

Numerous relational and non-relational databases are supported by Jakarta EE. There are several reasons why we use a RDBMS for Evaluator. First, data which is stored by Evaluator is always consistent and organized. As a result, this system does not benefit from data flexibility of NoSQL databases. Second, since all the necessary computing power is provided by a single server computer, Evaluator does not require a non-relational database to provide horizontal data scalability. In Evaluator, SQL queries provide secure and effective data processing. Therefore, among many other possibilities like Oracle, Java DB, and PostgreSQL, we have chosen MySQL, an RDBMS.

Evaluator's data source tier consists of the MySQL tables. EvaluatorDB database is created with Project, ScoreSet, User, UserFile, and UserPhoto tables. Along with the RDBMS, Evaluator offers an external file storage and retrieval mechanism for personal files and profile pictures of the registered users. Before the files are stored in the storage server, unique file names are produced in the business tier in RDBMS, these file names are saved linked to the projects. We use Undertow which is an internal web server provided by WildFly for the purpose of external file storage and retrieval.

Chapter 4: Evaluator Design

OOD on which Evaluator is based is a system of interacting objects. In OOD, all structures are objects created based on the class and interface concepts. This paradigm uses the encapsulation, inheritance, and polymorphism methods to enable the applications to be constructed in a scalable, reusable, and maintainable manner.

Unified Modeling Language (UML) is a standard language for defining, illustrating, and reporting the design of software development processes [Booch *et al.* 1996]. It offers various modeling notations to represent the software systems through diagrams.

Entity Relationship Diagram (ERD) is a structural diagram model which visualizes the attributes of the entities as well as their relationships with each other. It is usually used for displaying the logic of the database design.

In this section, Evaluator's design will be presented with the use of ERD, UML, and OOD tools that explains and visualizes functionality and implementation designs.

4.1 Functionality Design

Functionality design is essential for defining tasks before starting the development of a project. In this context, we explain how the database, user account, and evaluation project systems are designed.

4.1.1 Database Design

We designed database tables using Entity Relationship Diagrams (ERD) (Figure 2). We created three tables that contain information about general user data for project administrators and evaluators, user files, and user profile pictures. While the main user table contains information about login credentials and profile information, the other two tables associate files and photos with the users who own them.

Data of the projects is stored in another table. It contains all information of a project including its indicators graph. The indicators graph attribute holds the information about an evaluation project's indicators as serialized binary data, including the hierarchy, evaluators, weights, and scores of indicators.

The scores given to the indicators by the evaluators are selected from the score types of the score set assigned to those indicators by the administrators. The score sets that can be selected and the score types they contain are kept in a separate score set table in the database.

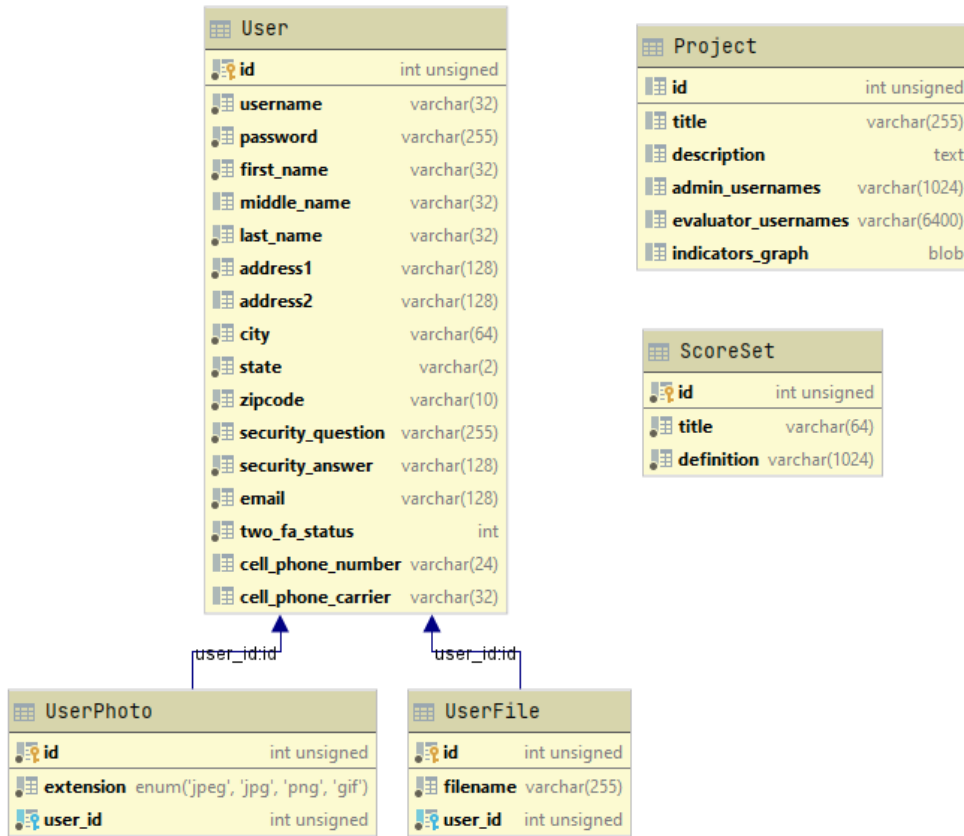


Figure 2. Database ER Diagram

4.1.2 User Account Design

Evaluator has two types of users which are super user and regular user. There can be only one super user in the application. All registered users and evaluation projects can be displayed by the super user.

Regular users are divided into two categories on a project basis as project administrators and evaluators. A user who is the administrator of one project can be the evaluator of another project. These two user types have different permissions and privileges in the application.

Administrator users can list the projects which they manage. They can change the names, descriptions, administrators, and evaluators of these projects. They can create and modify the project's indicators hierarchy. The score sets to be used in the evaluation of leaf indicators and the evaluators that will make the evaluation are appointed by the administrators. They also make pairwise comparisons of the sub-indicators in terms of their effects on their parent indicator. Similarly, they compare the assigned evaluators according to their expertise levels for the leaf indicators. Finally, they check the evaluation status of the project, and if the project is ready, they complete the evaluation of the project by propagating scores towards the root indicator.

Evaluator users can list projects with indicators that they are assigned as evaluators. They can view the hierarchy of indicators of these projects and the location of their assigned indicators in this hierarchy. The responsibility of evaluators is to give the evaluation project a score in terms of each of these indicators and write notes explaining the evaluations they made while giving that score.

4.1.3 Evaluation Project Design

During the creation of an evaluation project, its administrators and evaluators are determined. While administrators can edit a project's title, description, administrator, and evaluators, they also determine that project's indicator hierarchy and the properties of its indicators. We used a tree structure to represent the indicator hierarchy of a project. All information such as name, description, evaluators, scores, weights of the indicators that make up this tree are available in this structure. In order to store this complex structure in the database, we did not prefer to create separate tables for indicators and indicator hierarchy. Instead, we serialize the tree structures representing the indicator hierarchy of each project and store it as binary data in the database. While it is retrieved, we deserialize these records and obtain the tree structure containing all the information.

4.2 Implementation Design

This section presents package and class designs using UML diagrams. The OOD paradigm which offers structures that embody the three key design concepts of encapsulation, inheritance, and polymorphism serves as the foundation for the construction of classes and packages.

4.2.1 Package Design

In Evaluator, there are six key packages: EntityBeans, FaçadeBeans, Controllers, Managers, Globals, and Validators. Hierarchically, EntityBeans are at base level of the calling chain. The FaçadeBeans calls the EntityBeans and access their fields. At the top-level Controllers and Managers calls FaçadeBeans and they also have access to the EntityBeans. On the other hand, the Globals and Validators packages are independent packages.

All of the entities that represent database tables are contained in the EntityBeans package. All the façades that give controllers access to database operations through APIs are held by FaçadeBeans. EntityBeans and FaçadeBeans are examples of lower-level packages that are used by the Controllers and Managers packages to construct the business logic of the backend. The global tool and validation classes are included in the Globals and Validators packages.

The design of the packages is important in terms of categorizing and organizing the classes in the project. For the project to reach high standards such as high cohesion and low coupling, it is necessary to have a good package design.

4.2.2 Class Design

Class design is a vital phase of development that must be completed before the project implementation. In order for the modules of the project to be reusable and maintainable, it must have a good class design. Class designs are displayed in this section using UML diagrams.

4.2.2.1 Entity Beans

The entity beans diagram displays five entity bean classes and the dependencies between them (Figure 3). The methods and properties of each class are shown under the class name. The discontinuous arrow lines in the chart shows the dependencies between classes. The continuous arrow lines display the database relationships since the entity bean classes represent the database tables. UserFile and UserPhoto classes have dependencies to the User class since they both contain a User object. There are not any dependencies between any two of the User, Project, and Score Set entity classes.

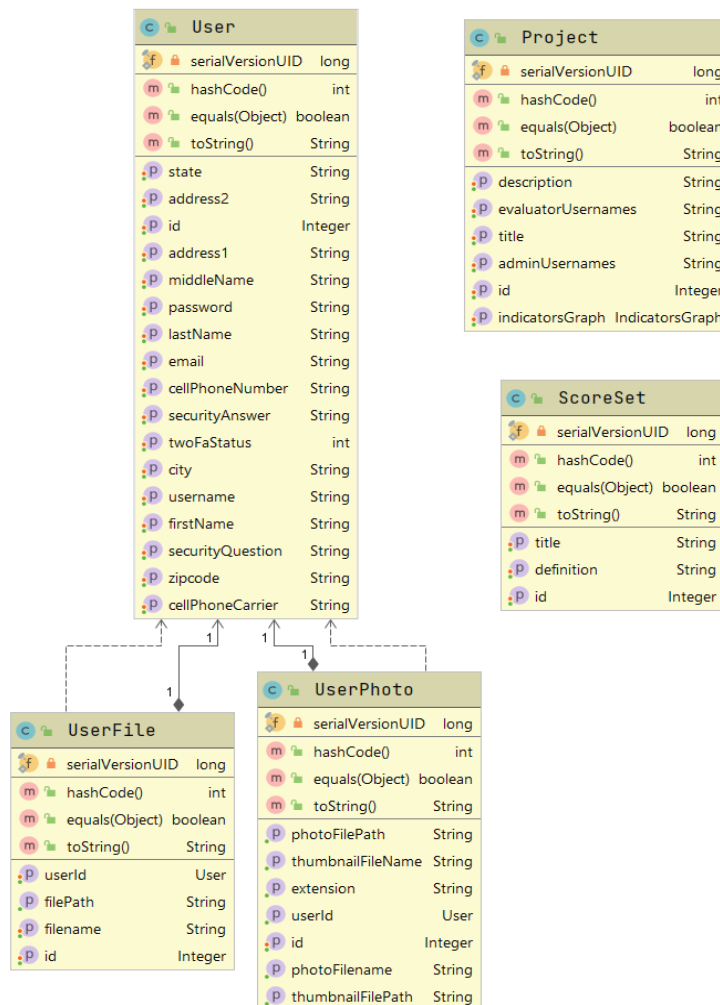


Figure 3. UML diagram of Entity Beans

4.2.2.2 Façade Beans

Figure 4 displays the UML diagrams of the façade. As the continuous blue lines show the façade beans which are inherited from *Abstract Façade*, which contains methods for fundamental database operations. *UserFacade*, *UserPhotoFacade*, *UserFileFacade*, *ProjectFacade*, and *ScoreSetFacade* inherit from *Abstract Façade*, and each of them individually implements some specialized operations for their corresponding entity beans.

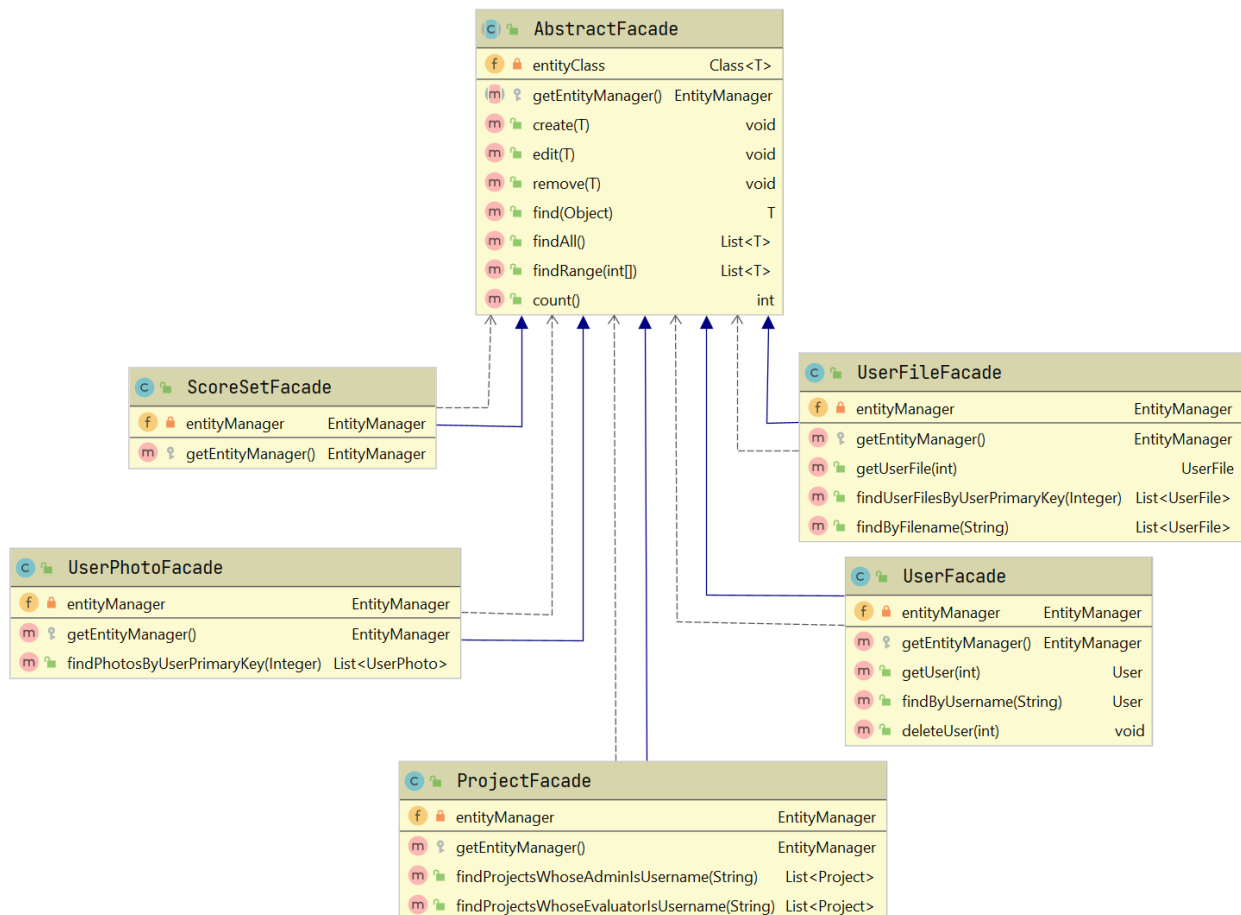


Figure 4. UML Diagram of Façade Beans

4.2.2.3 Controllers

The UML diagrams of the controllers are shown in Figure 5 and Figure 6. Since controllers handle most of the business logic, the classes in this package have more complex relationships.

UserController, UserPhotoController, UserFileController, ProjectController, and ScoreSetController are responsible for the logic related to the five corresponding entity beans. The controllers that include properties holding data obtained from the editable fields in the user interface have dependencies to EditorController. EmailController and TextMessageController include TFA operations and have no relationships with any other controller. TreeTableController and TabViewController are for formatting very complex data for two user interface components displaying hierarchy and details of the indicators. EvaluatorController consists of the operations for scoring the indicators and evaluating the project. The relationships of all these controller classes are displayed in Figure 7.

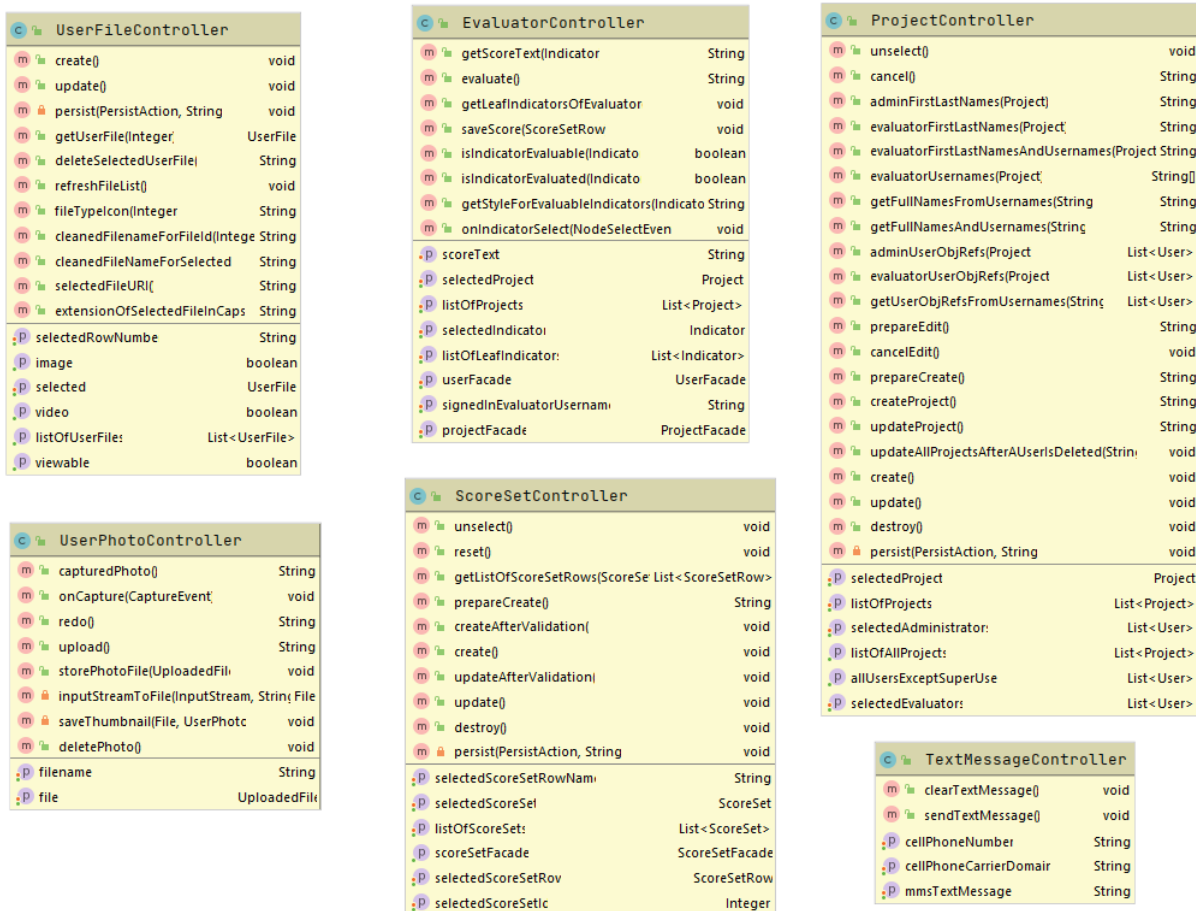


Figure 5. UML Diagrams of Controllers

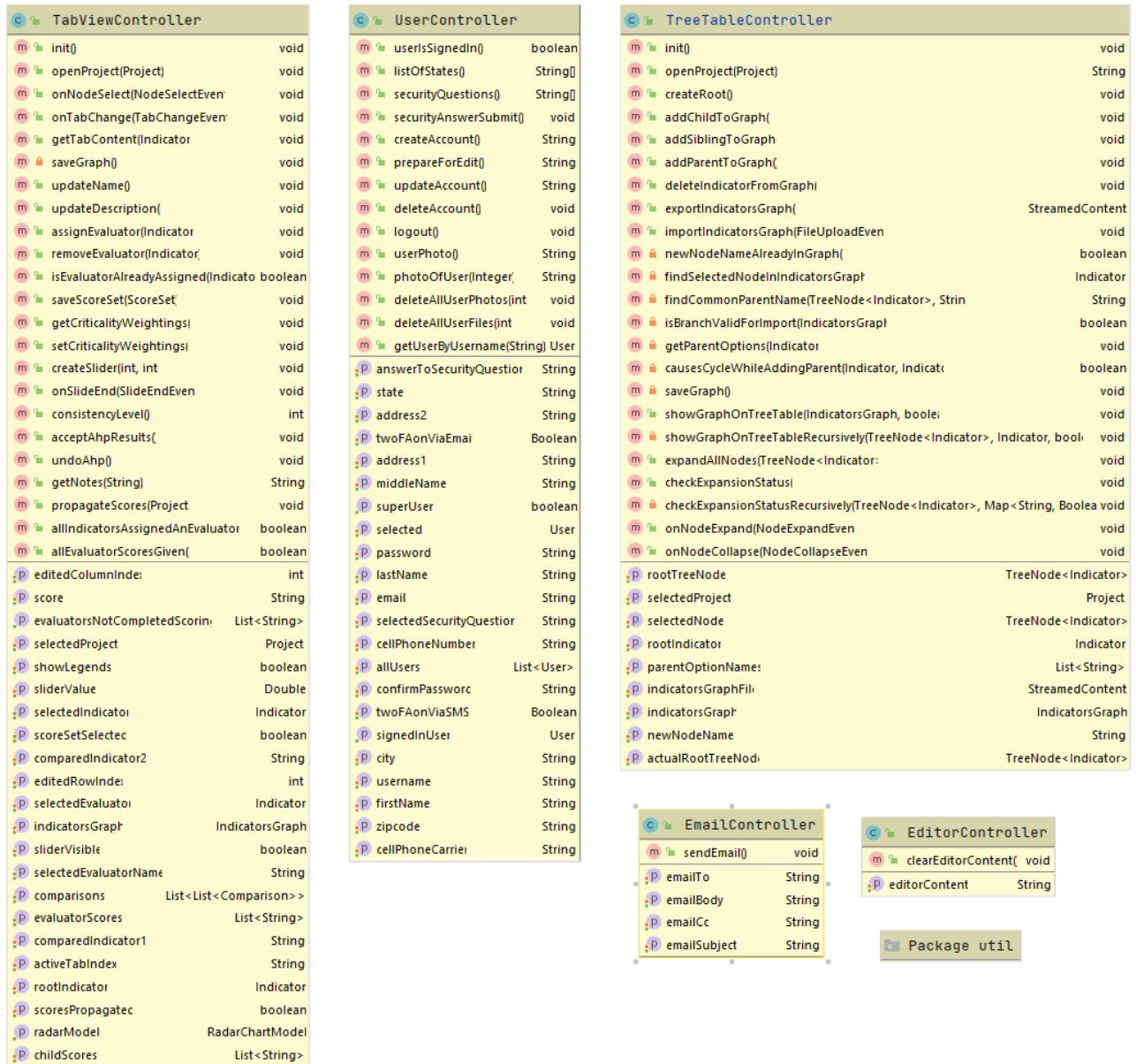


Figure 6. UML Diagrams of Controllers

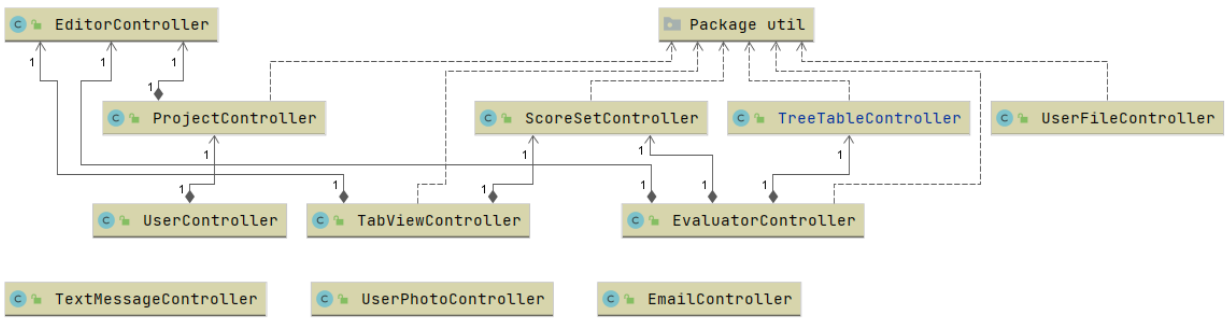


Figure 7. UML Diagram of Dependencies of Controllers

4.2.2.4 Managers

The UML diagram of the managers are shown in Figure 8. LoginManager, PasswordResetManager, and AccountRecoveryManager handle account related operations. ReportManager is responsible for formatting data to create a PDF report of the evaluation projects. There are also FileUploadManager and FileDownloadManager for the file operations. These managers are all independent classes with no relationships between each other.

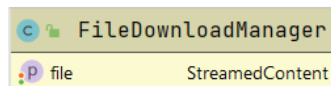
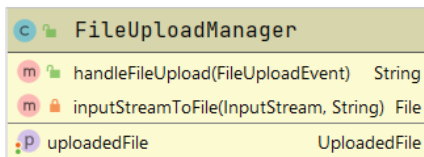
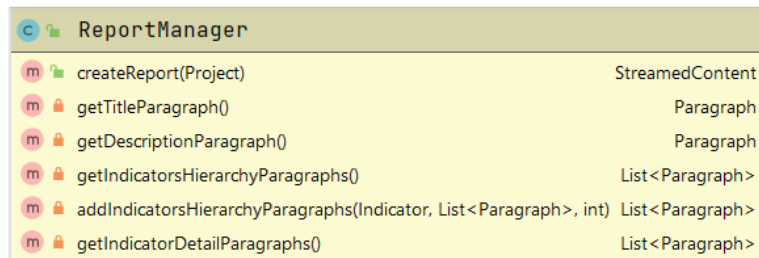
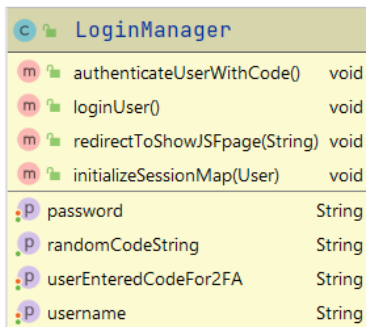
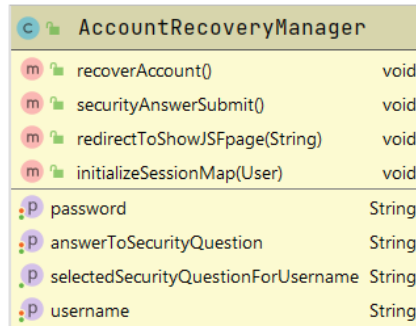
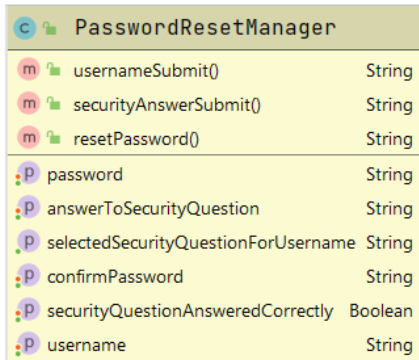


Figure 8. UML diagram of Managers

Chapter 5: Evaluator Functionality

The functionalities of Evaluator are presented in this chapter.

5.1 Account Creation

A user must enter the necessary data to create an account, which includes the first and last names, first line of the address, the city, state, ZIP code, the security question and its response, email, the username, and the password (Figure 9). The username must include 6 to 32 characters, including uppercase and lowercase letters, special characters, and numerals. The password must be between 8 and 32 characters long and include at least one special character, one number, one uppercase and one lowercase letter. The email must adhere to accepted email standards. If not, an email format error is displayed on the sign-up page. The security question which is used to change passwords is selected from a list.

Evaluator has two different user types: the super user and regular users. In order to register as the super user, the username must be selected as SuperUser. All users with any username other than SuperUser are registered as regular users.

Users can enable TFA if they wish to make their account login more secure. They have two options: TFA via email and TFA via SMS. If one of them is enabled, when the user wants to login an authentication code is sent to the user as an email or SMS depending on the method chosen.

shark.cs.vt.edu/Evaluator/userAccount/CreateAccount.xhtml

Hello, Guest

Menu

Create a User Account

First Name: *	<input type="text"/>
Middle Name:	<input type="text"/>
Last Name: *	<input type="text"/>
Address Line 1: *	<input type="text"/>
Address Line 2:	<input type="text"/>
City: *	<input type="text"/>
State:	AK ▾
Zip Code: ? *	<input type="text"/>
Security Question:	In what city or town did your mother and father meet? ▾
Security Answer: *	<input type="text"/>
Email: *	<input type="text"/>
Username: ? *	<input type="text"/>
Password: ? *	<input type="password"/>
Confirm Password: *	<input type="password"/>

Two-Factor Authentication

Two-Factor Authentication via Email:	<input type="checkbox"/>
Two-Factor Authentication via SMS:	<input type="checkbox"/>
Cell Phone Number:	<input type="text"/>
Cell Phone Carrier:	Select a Carrier ▾

[Privacy Statement](#)
[Principles of Community](#)
[Acceptable Use](#)

Virginia Polytechnic Institute and State University (Virginia Tech)

Figure 9. Account Creation Interface

5.2 User Sign-In

The user sign-in functionality consists of username, password, and the reCAPTCHA checkbox (Figure 10). After entering the correct login information, clicking the reCAPTCHA checkbox, and the Sign In button, the user is directed to the profile page by logging in if TFA is not enabled. If it is enabled, a verification code is sent to the user via email or SMS, and the user will be directed to the TFA page where they will enter this code (Figure 11).

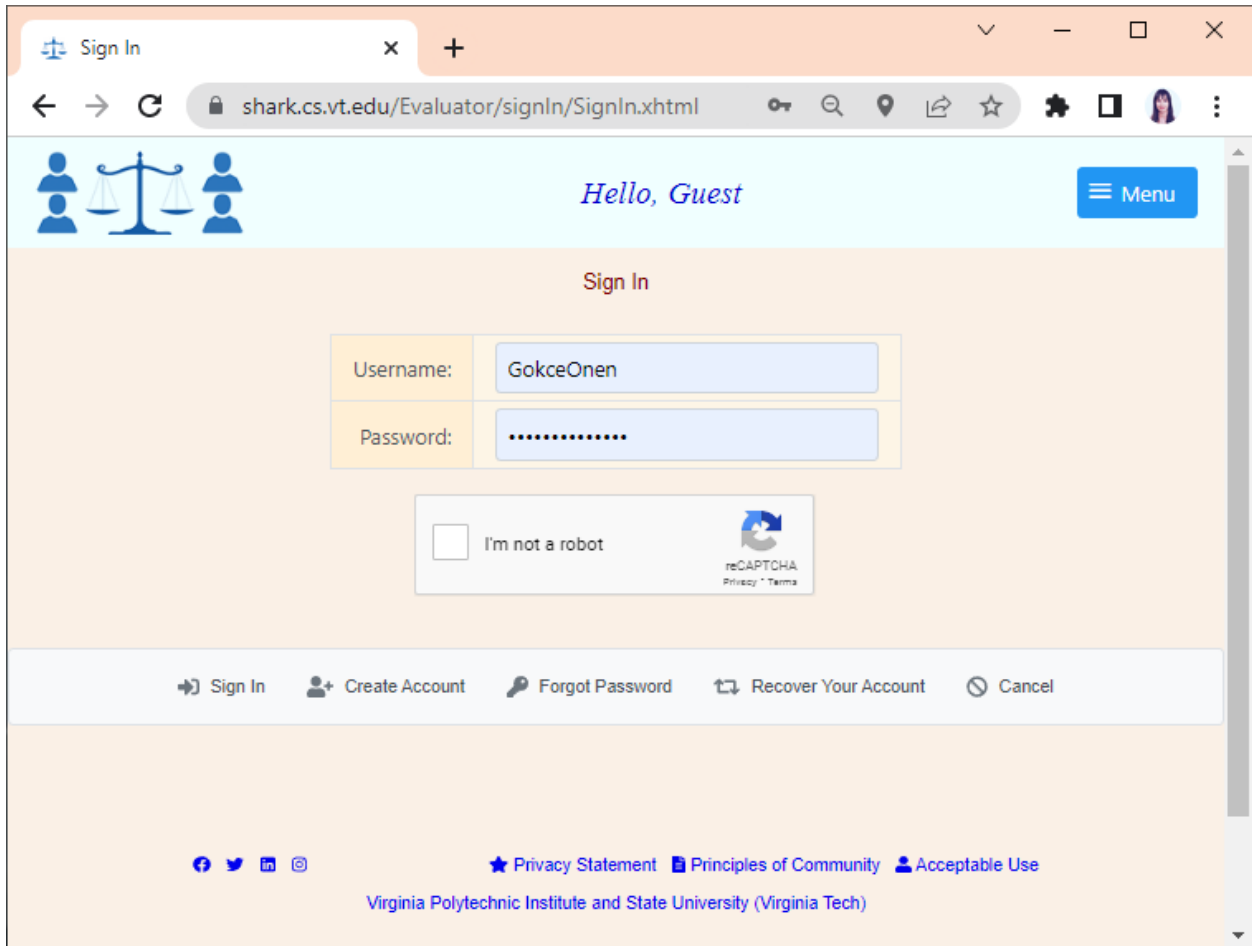


Figure 10. User Sign-In Interface

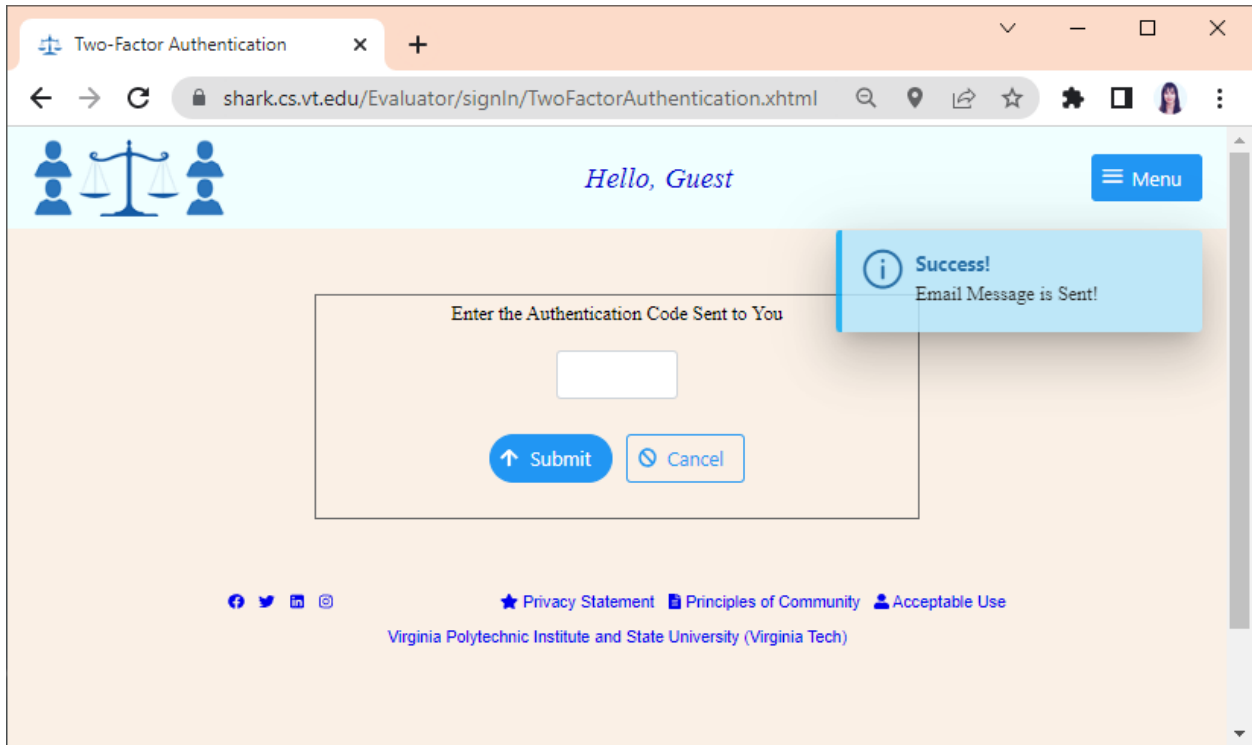


Figure 11. TFA Interface

5.3 Password Change

There are three stages in password change. The user must first provide their username in order to log in. The user must then respond to the security question for the account that is currently logged in. The security question is selected during account creation. The user must then enter and validate the new password. (Figure 12, Figure 13, Figure 14).

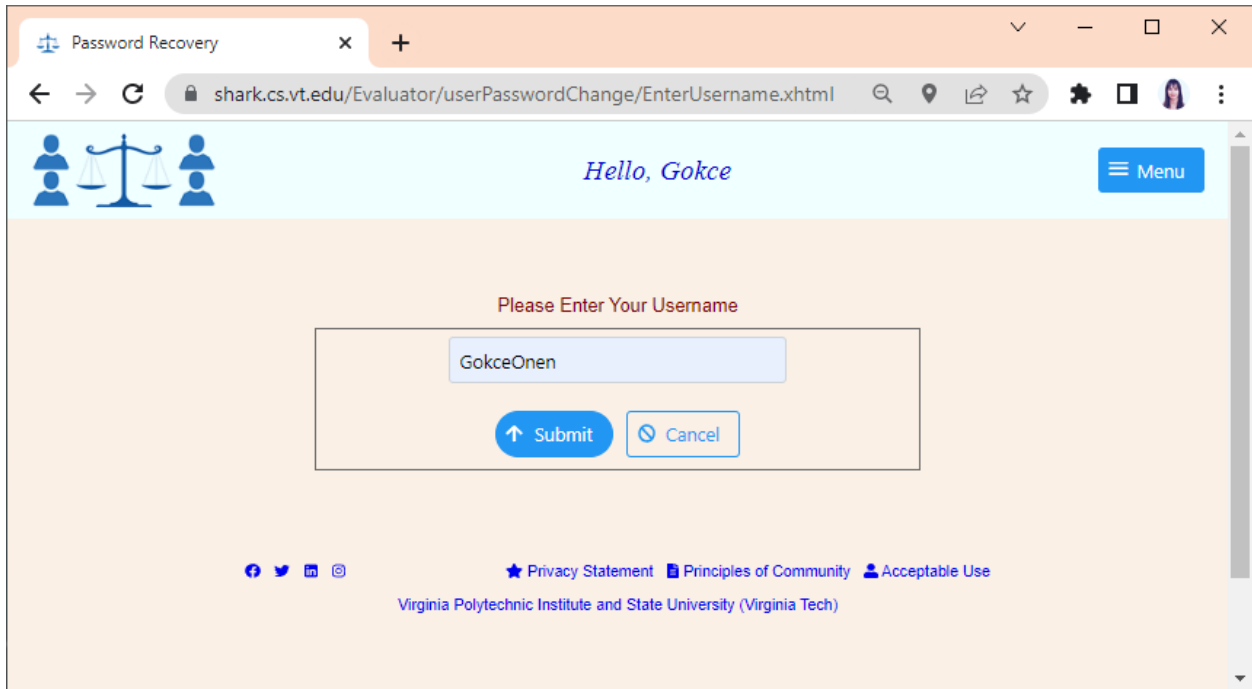


Figure 12. Password Change Interface

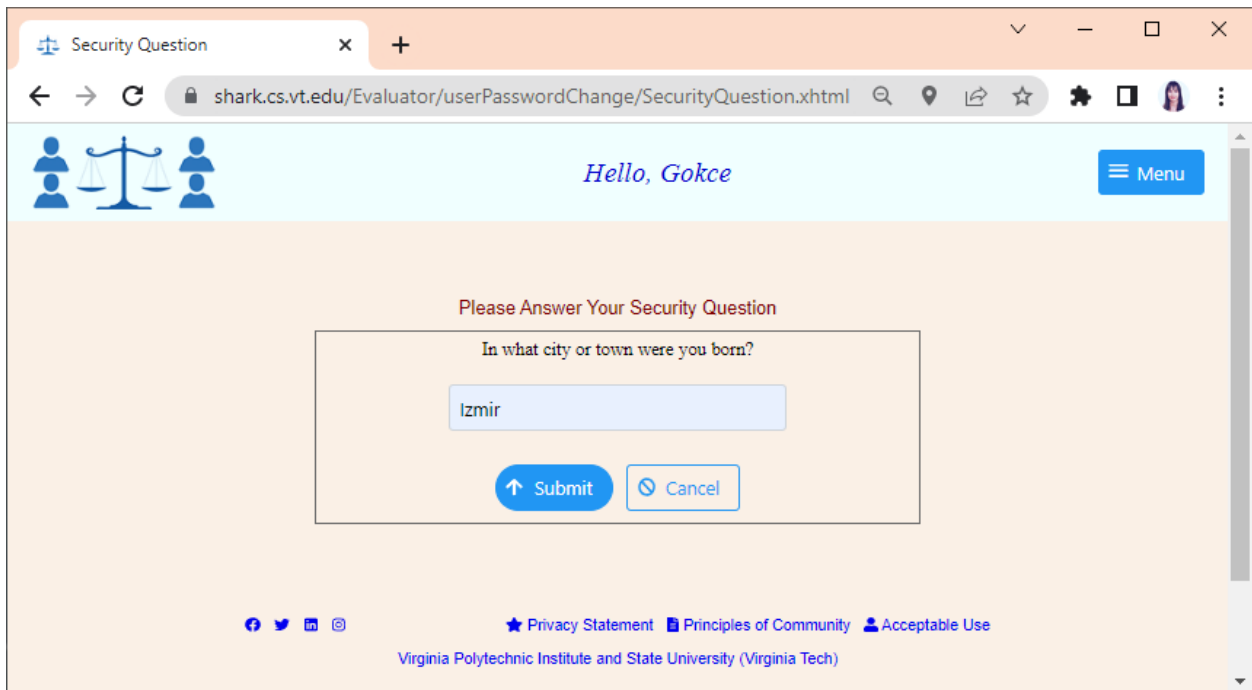


Figure 13. Password Change Interface

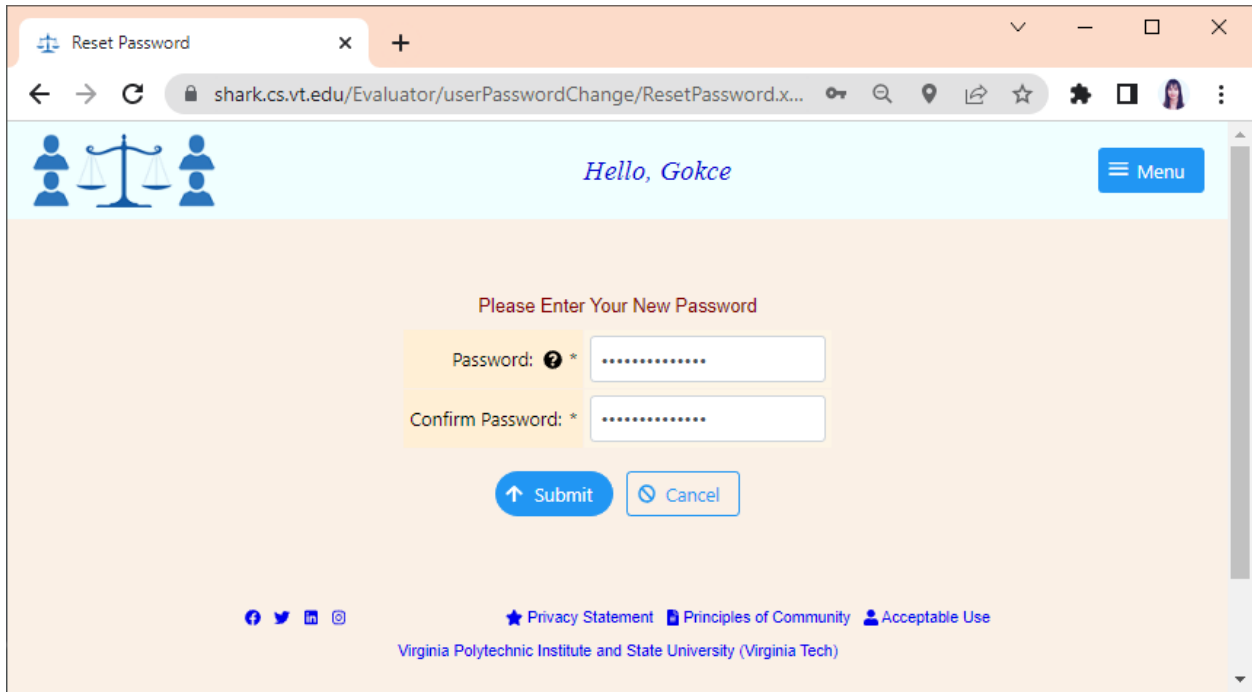


Figure 14. Password Change Interface

5.4 Account Recovery

Account recovery is provided for the users who have activated TFA for the cases of losing access to the phone number or email address they use to receive a verification code. Users enter their username, password, and the answer to their security question in order to regain access to their accounts by bypassing TFA (Figure 15, Figure 16).

5.5 Project Creation

Project creation page consists of the project title, project description, project administrators, and project evaluators fields (Figure 17). Project administrators and evaluators are selected among the users registered in the system. After a project is created, it becomes visible on the My Projects page of the project's administrators.

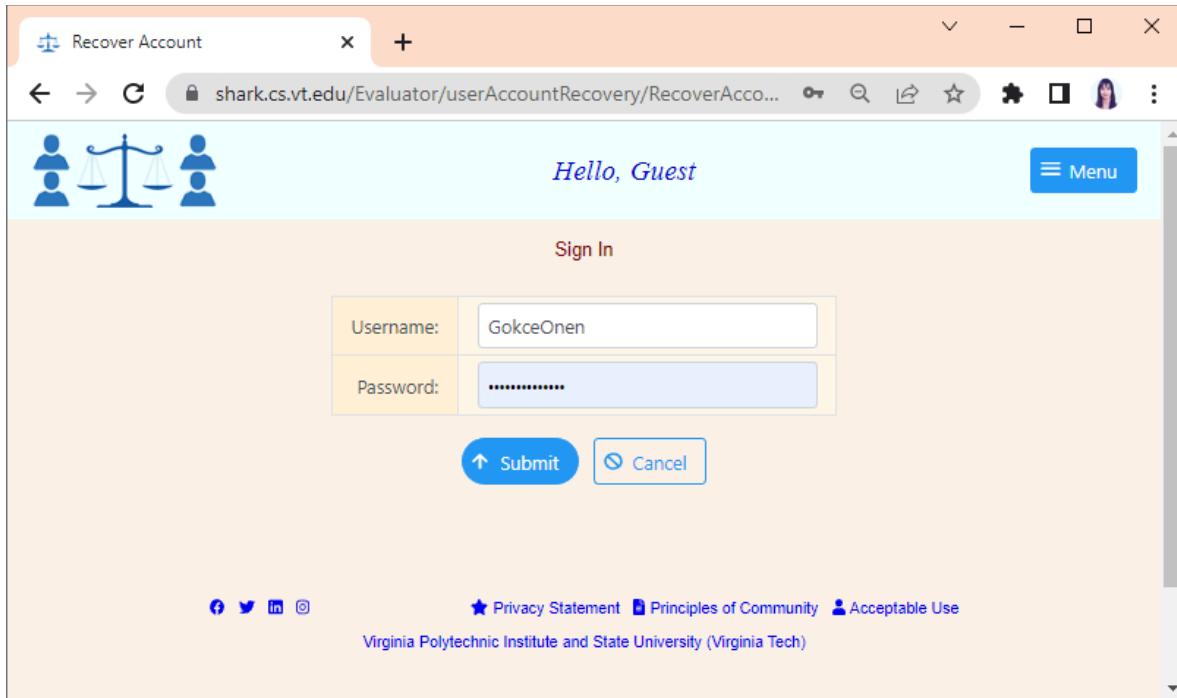


Figure 15. Account Recovery Interface

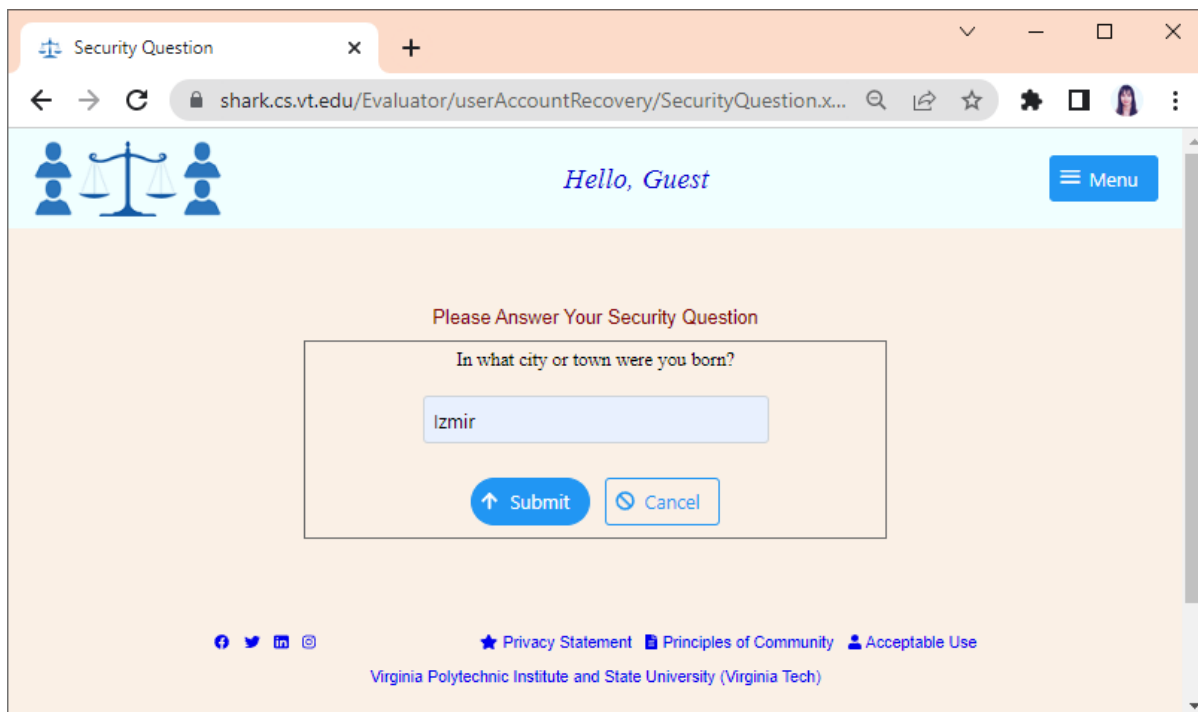


Figure 16. Account Recovery Interface

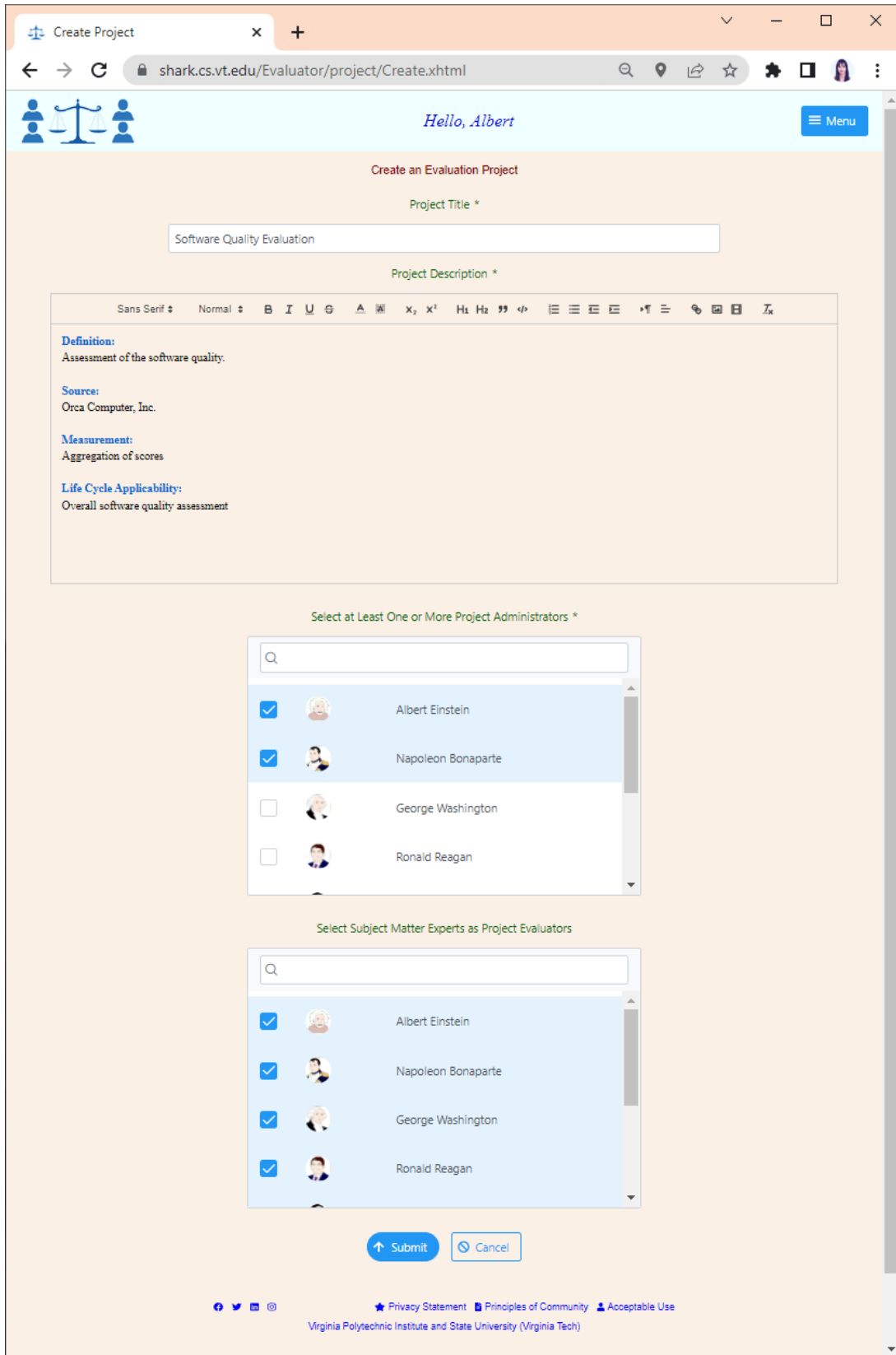


Figure 17. Project Creation Interface

5.5.1 Project Administrators

Project administrators can see a list of all the evaluation projects which they are the administrator of. An evaluation project can be viewed, edited, deleted, and opened only by its administrators (Figure 18). After administrators open a project, they can create its indicator hierarchy, specify parameters of this hierarchy, and assign the project's evaluators to evaluate the project in terms of certain indicators.

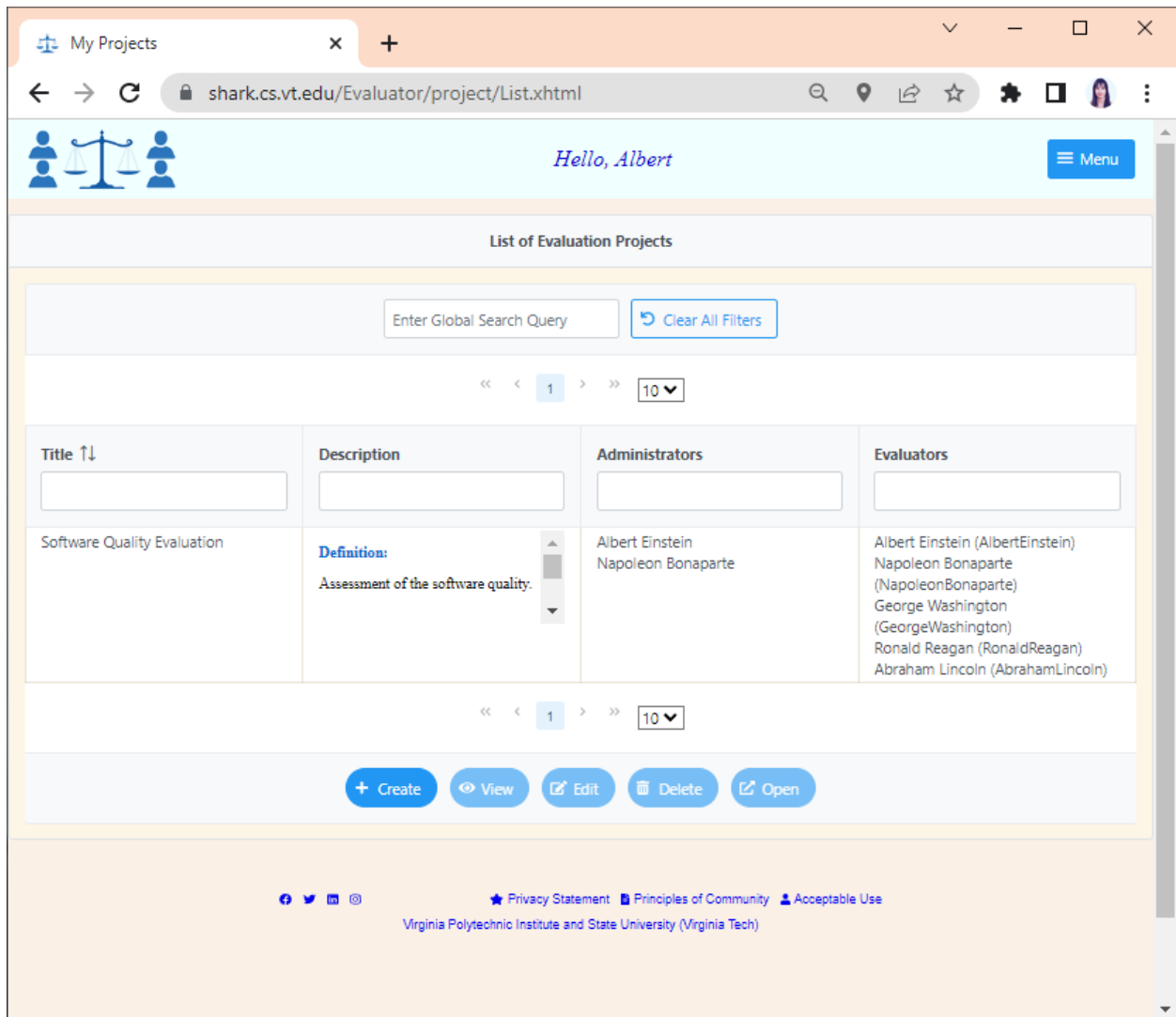


Figure 18. Administrator's Evaluation Projects

5.5.2 Project Evaluators

The evaluators of a project are assigned by the administrators to evaluate the project in terms of some indicators according to their area of expertise. These evaluators can see a list of the projects which they are the evaluator of (Figure 19). By selecting one of these projects, they can see the indicators which they are assigned to in that project (Figure 20). Therefore, they can evaluate the project in terms of these indicators.

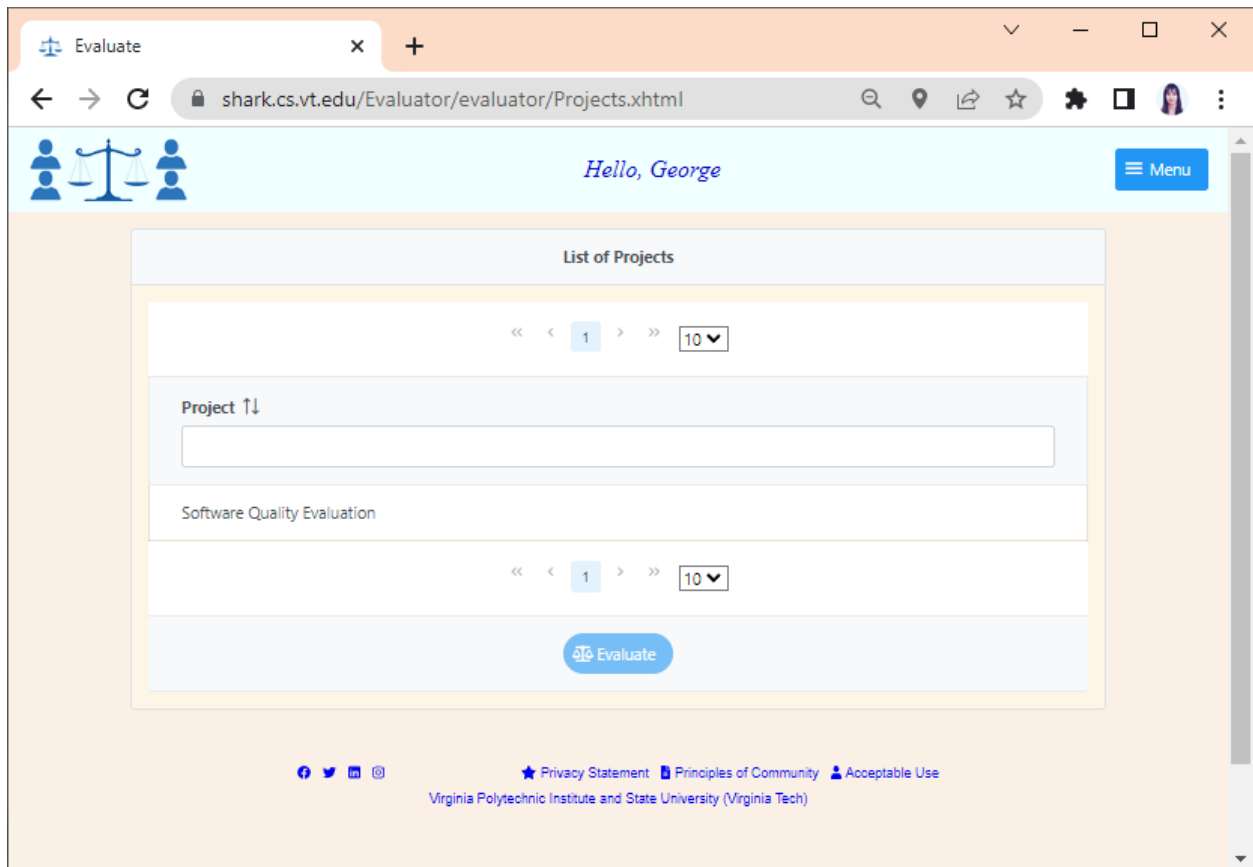


Figure 19. Evaluator's Evaluation Projects

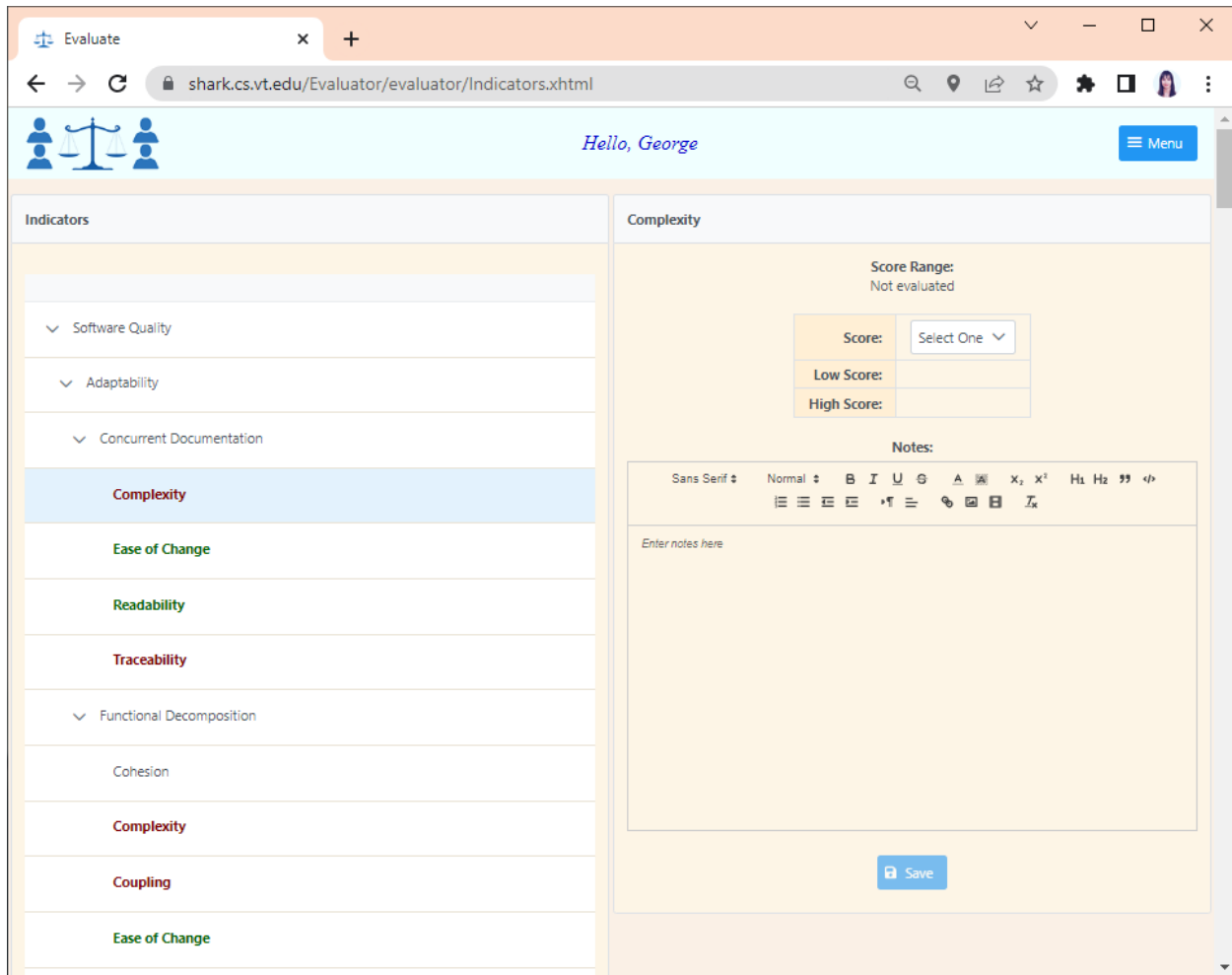


Figure 20. Indicators the Evaluator is Assigned to

5.5.3 Creation of Indicators Hierarchy

Creation of indicators hierarchy consists of the following functionalities:

- Creating the root indicator
- Adding a child, sibling, or parent to an indicator
- Deleting an indicator
- Exporting and importing a complete hierarchy, a branch, or a single indicator.

Administrators use these functionalities to determine which indicators the project will be evaluated (Figure 21). All indicators, except for the leaf indicators which are at the lowest level in hierarchy, are also evaluated in terms of their own children (sub-indicators).

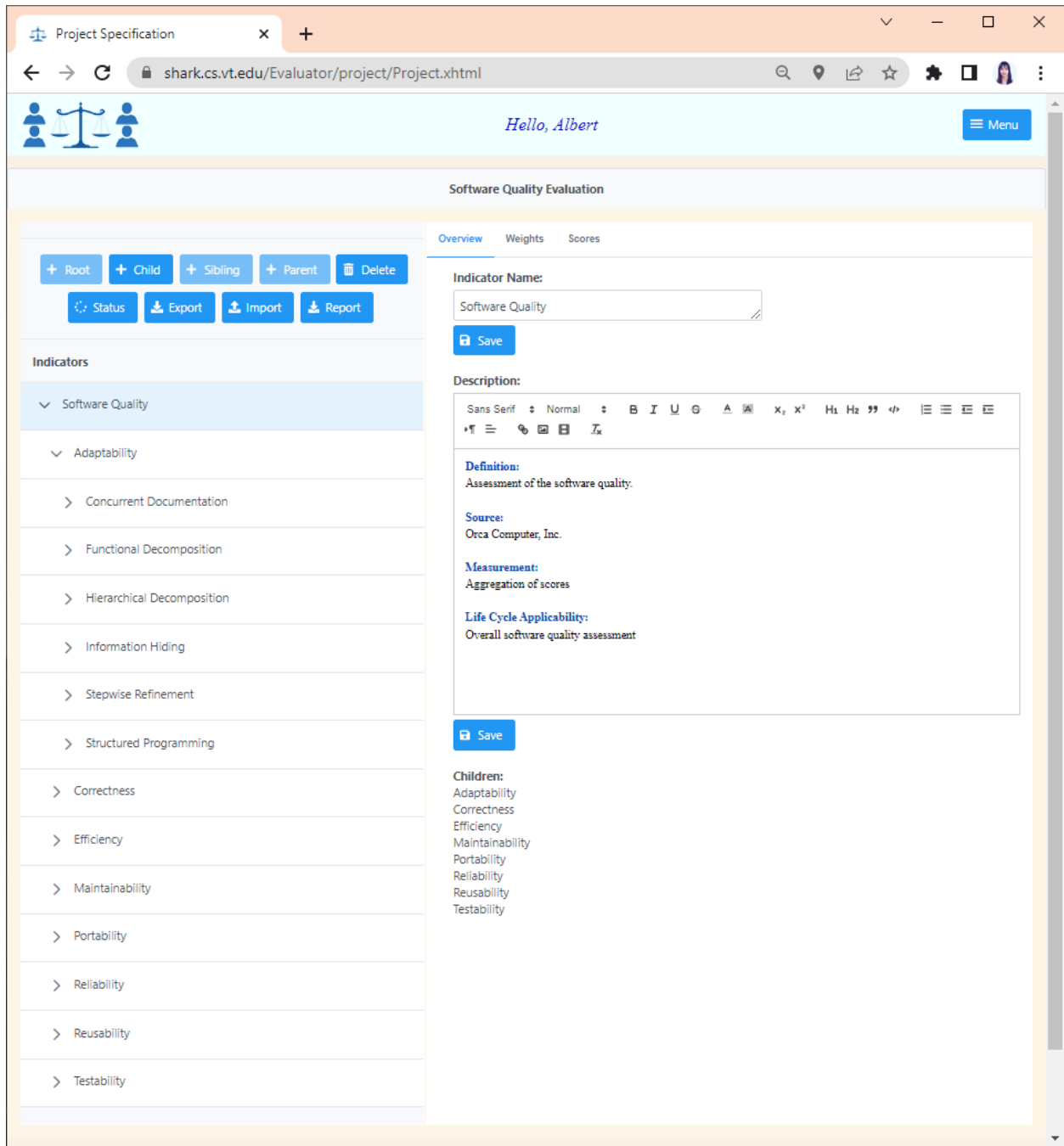


Figure 21. Part of the Indicators Hierarchy of Software Quality Evaluation

By selecting an indicator from the hierarchy of indicators, information about that indicator can be viewed and edited in the Overview section (Figure 22).

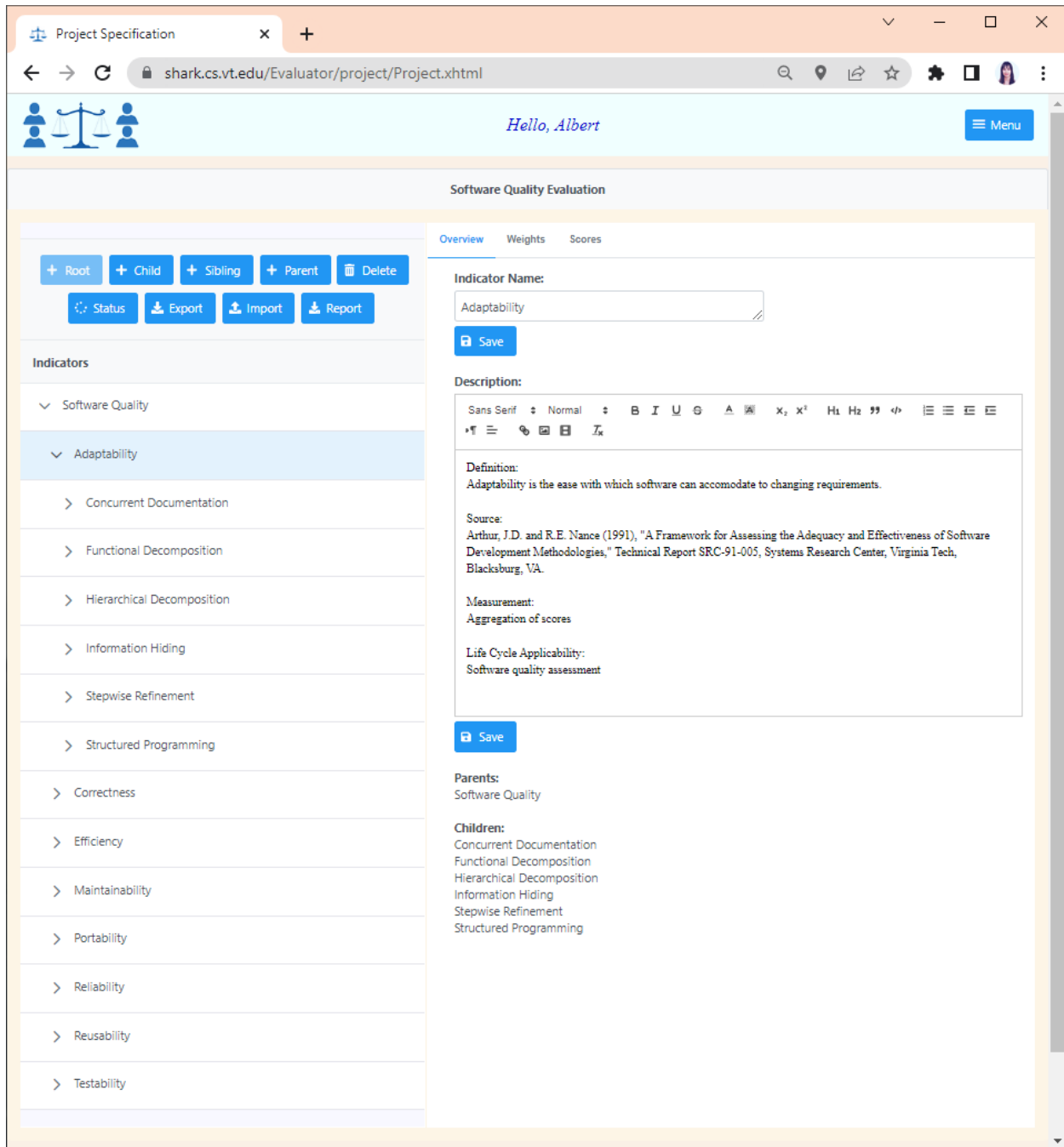


Figure 22. Overview of the Adaptability indicator

After selecting an indicator from the hierarchy of indicators, administrators can observe the relative child indicators weights of the selected indicator in the Child Relative Weights table. The weights in this table are also visualized with a radar chart (Figure 23).

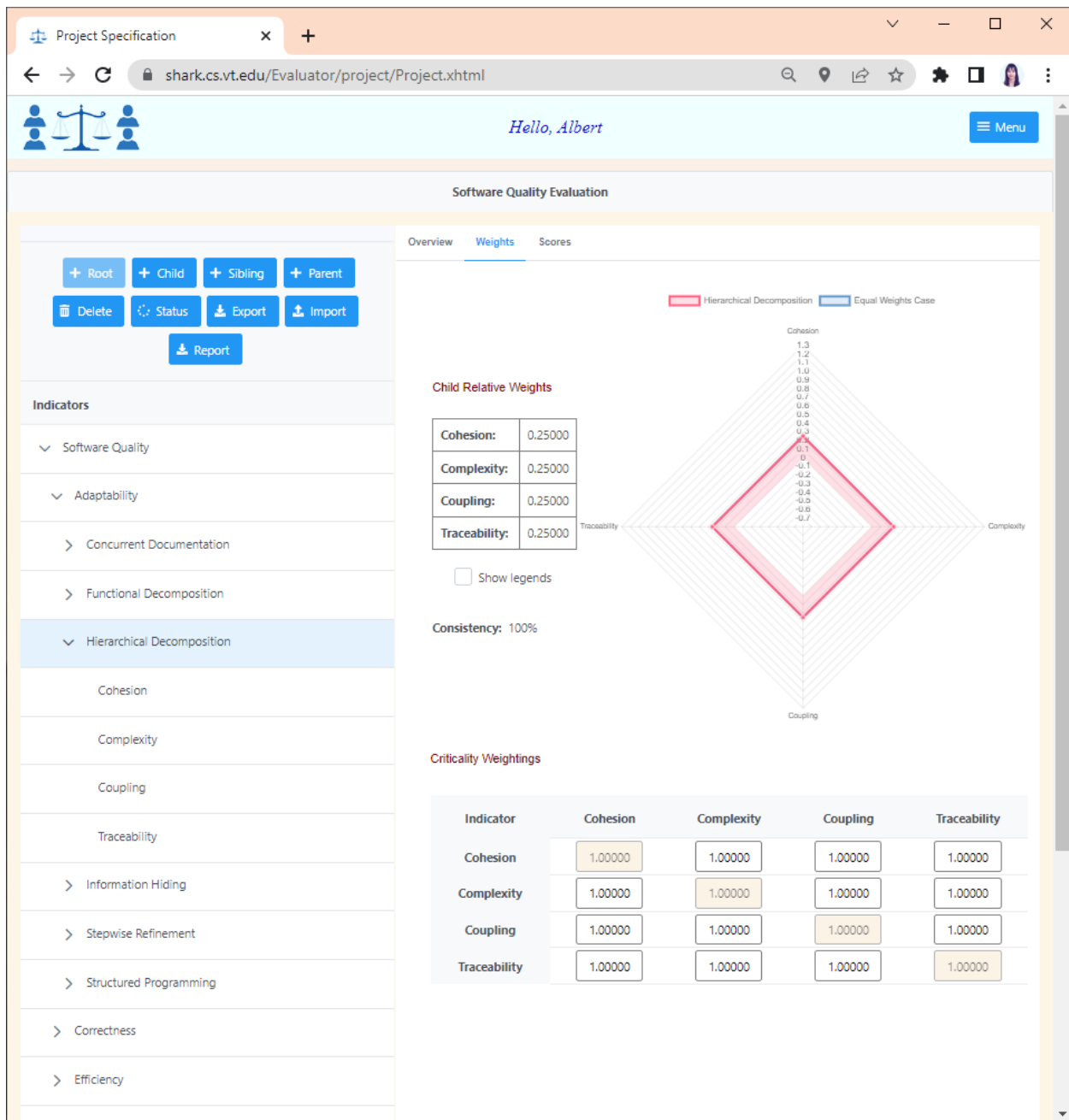


Figure 23. Weights of the Hierarchical Decomposition Indicator

The administrator can change the child relative weights by editing the pairwise comparisons in the criticality weightings table. In order to do this, they determine how important an indicator is compared to the other in a selected pairwise comparison. To determine the degree of importance, they select the appropriate value on a scale from 1 to 9 divided into options of equally, moderately, strongly, very strongly, and extremely (Figure 24).

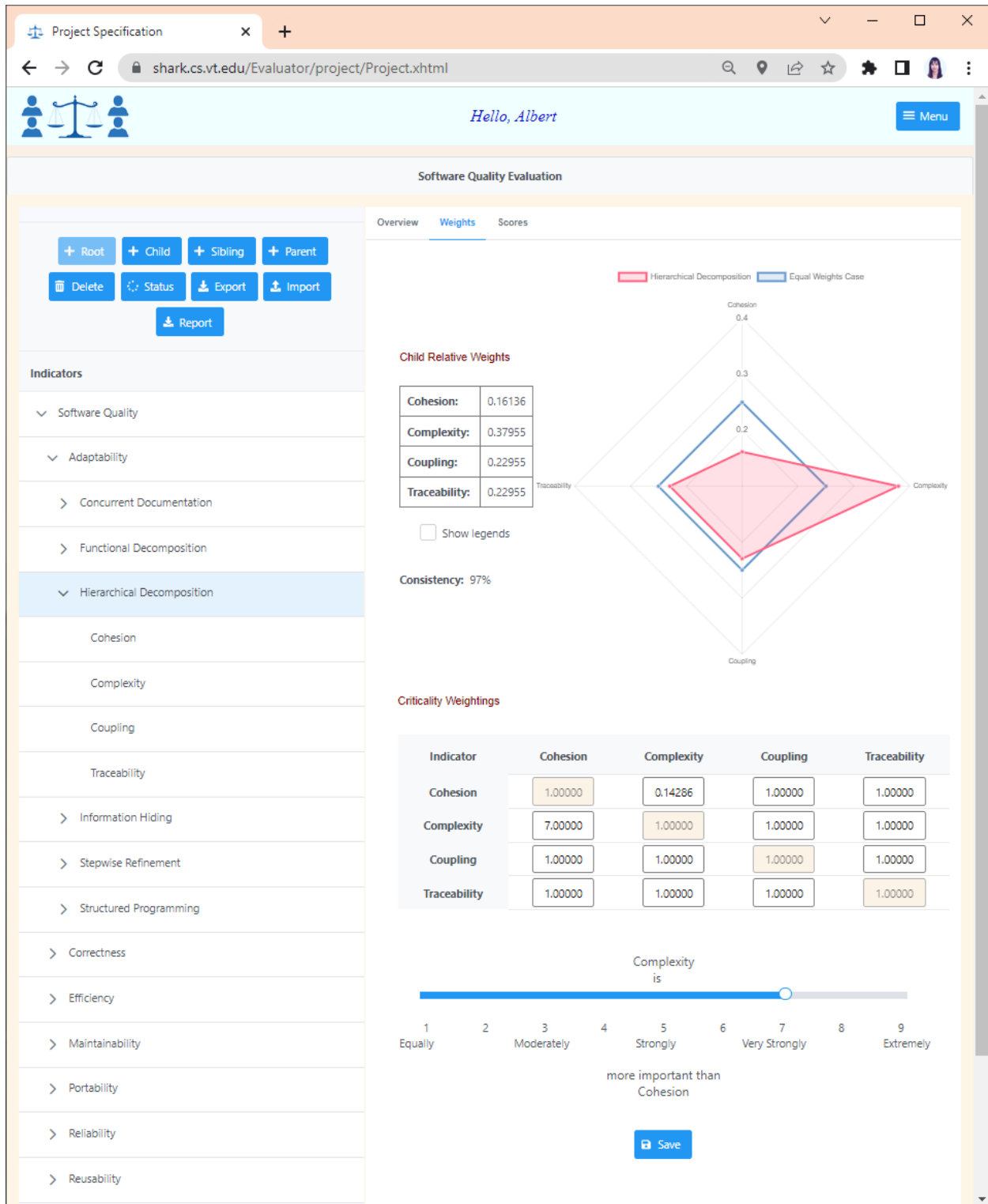


Figure 24. Editing Pairwise Comparison of Complexity and Cohesion

Editing any pairwise comparison in the Criticality Weightings table and clicking the Save button opens a dialog showing how consistent the new values are with each other according to the AHP

algorithm. Accordingly, the administrator can choose to continue editing the table with the current weights, save the current weights, or reset the table to equate all weights (Figure 25).

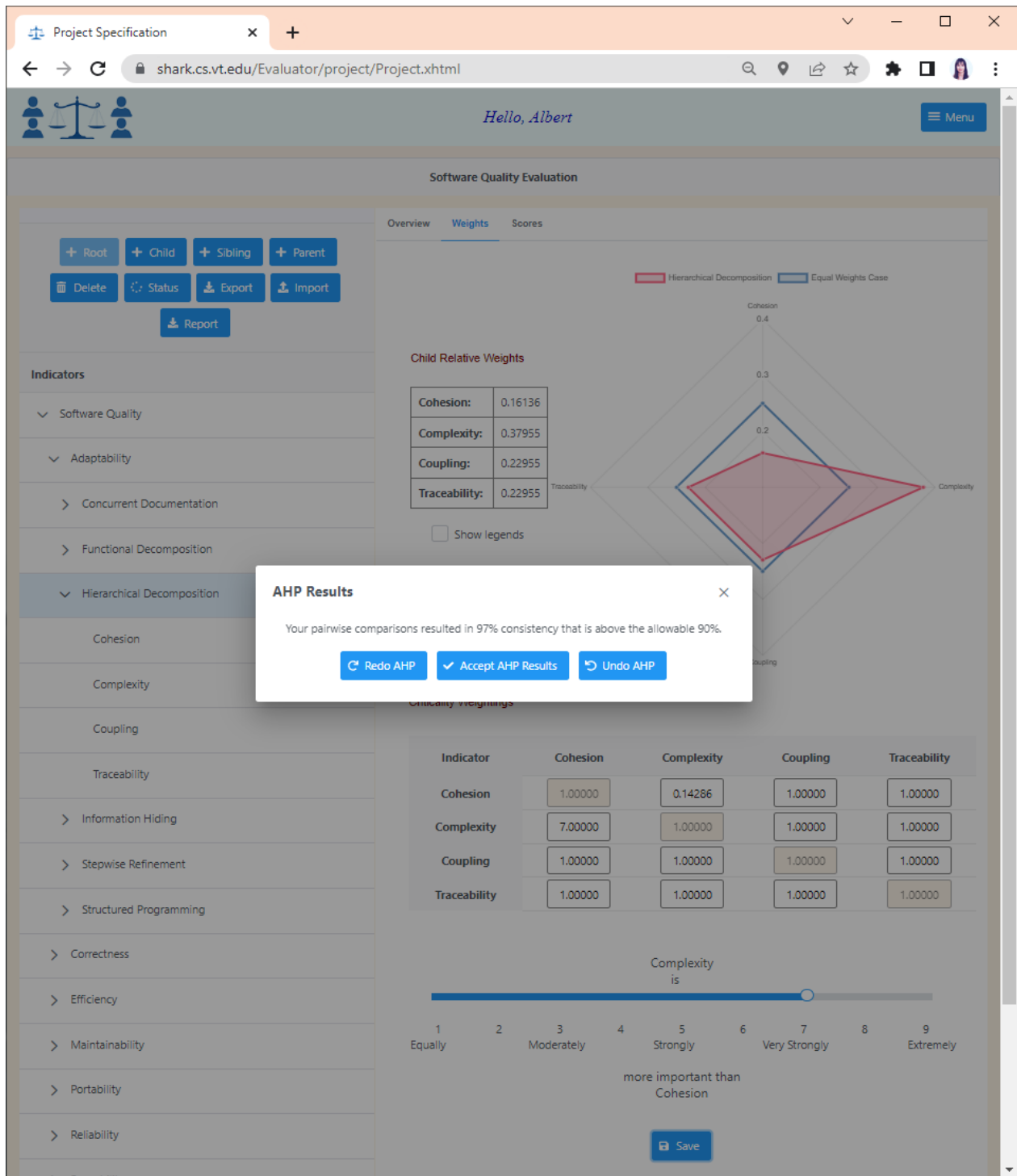


Figure 25. AHP Consistency Results

If the selected indicator is a leaf indicator, which is an indicator with no children at the lowest level of the hierarchy, the administrator can assign an evaluator to the selected indicator (Figure 26) or remove an assigned evaluator from the selected indicator (Figure 27).

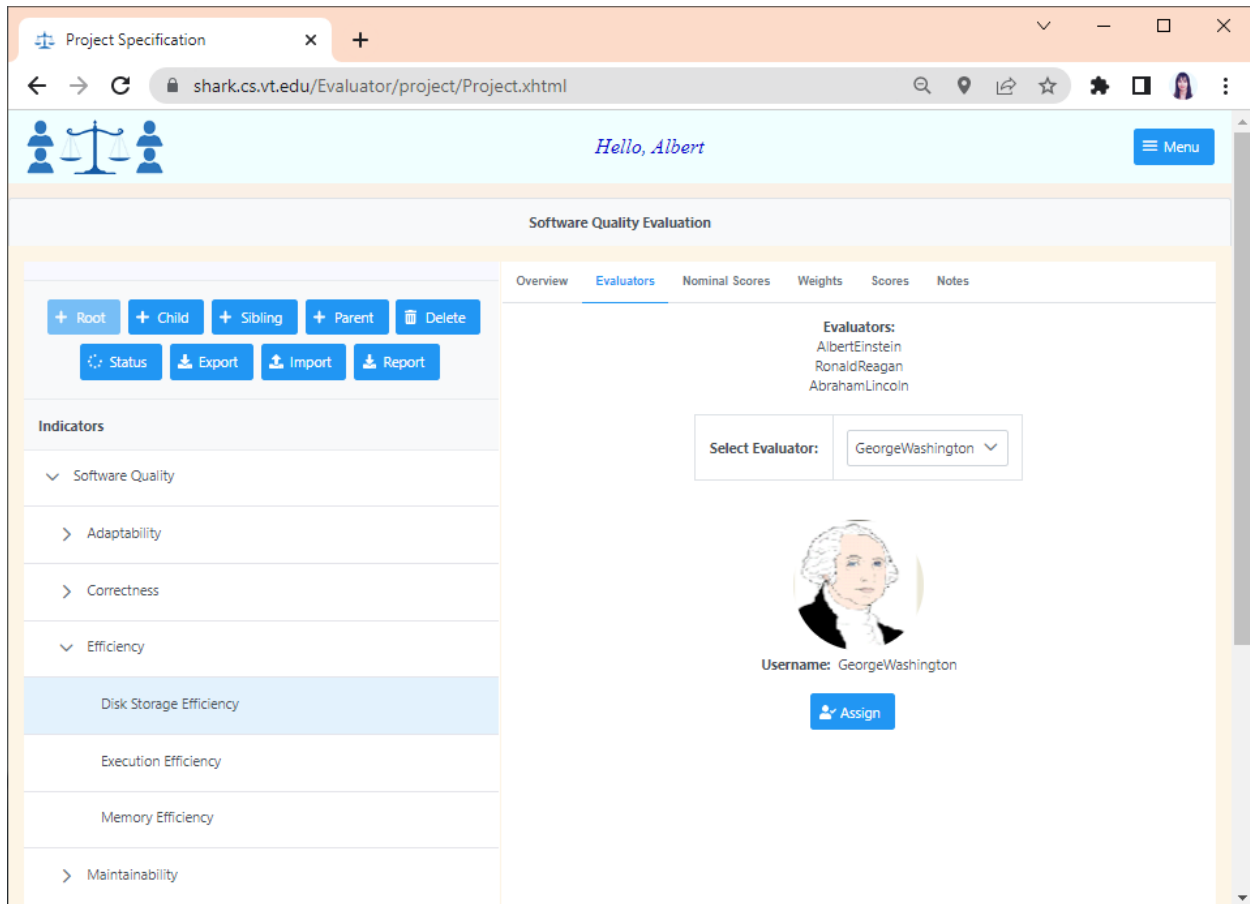


Figure 26. Assigning an Evaluator to a Leaf Indicator

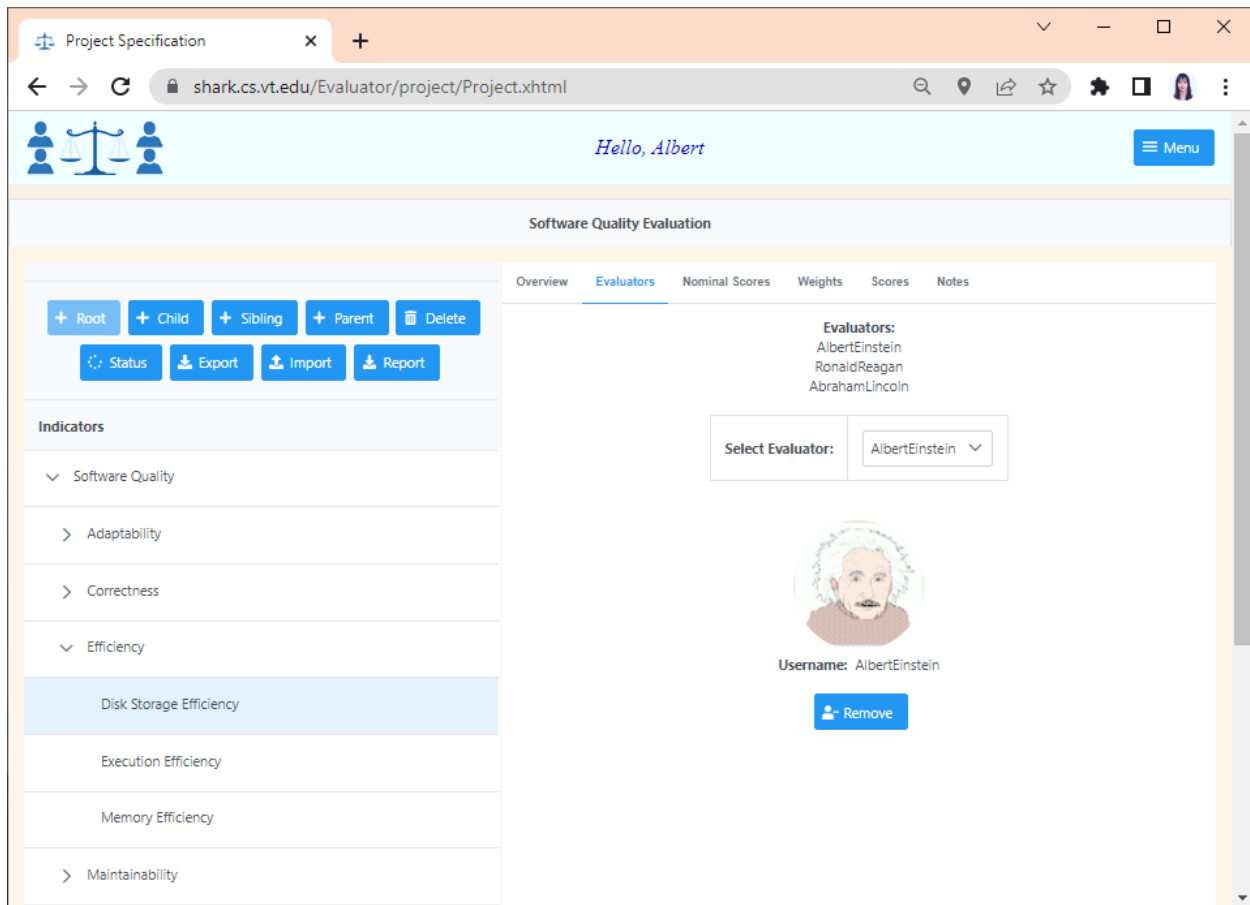


Figure 27. Removing an Evaluator from a Leaf Indicator

If the selected indicator is a leaf indicator, the administrator assigns one of the predefined nominal score sets to it (Figure 28). A list of all nominal score sets can be seen on the List of Nominal Score Sets page by all types of users (Figure 29). This page contains 11 different score sets. Within each score set, there are nominal scores corresponding to a certain numerical score range. The administrator assigns one of these score sets to all leaf indicators. Hence, the evaluators who are assigned to these leaf indicators evaluate the project in terms of that leaf indicator by selecting a score from the assigned score set.

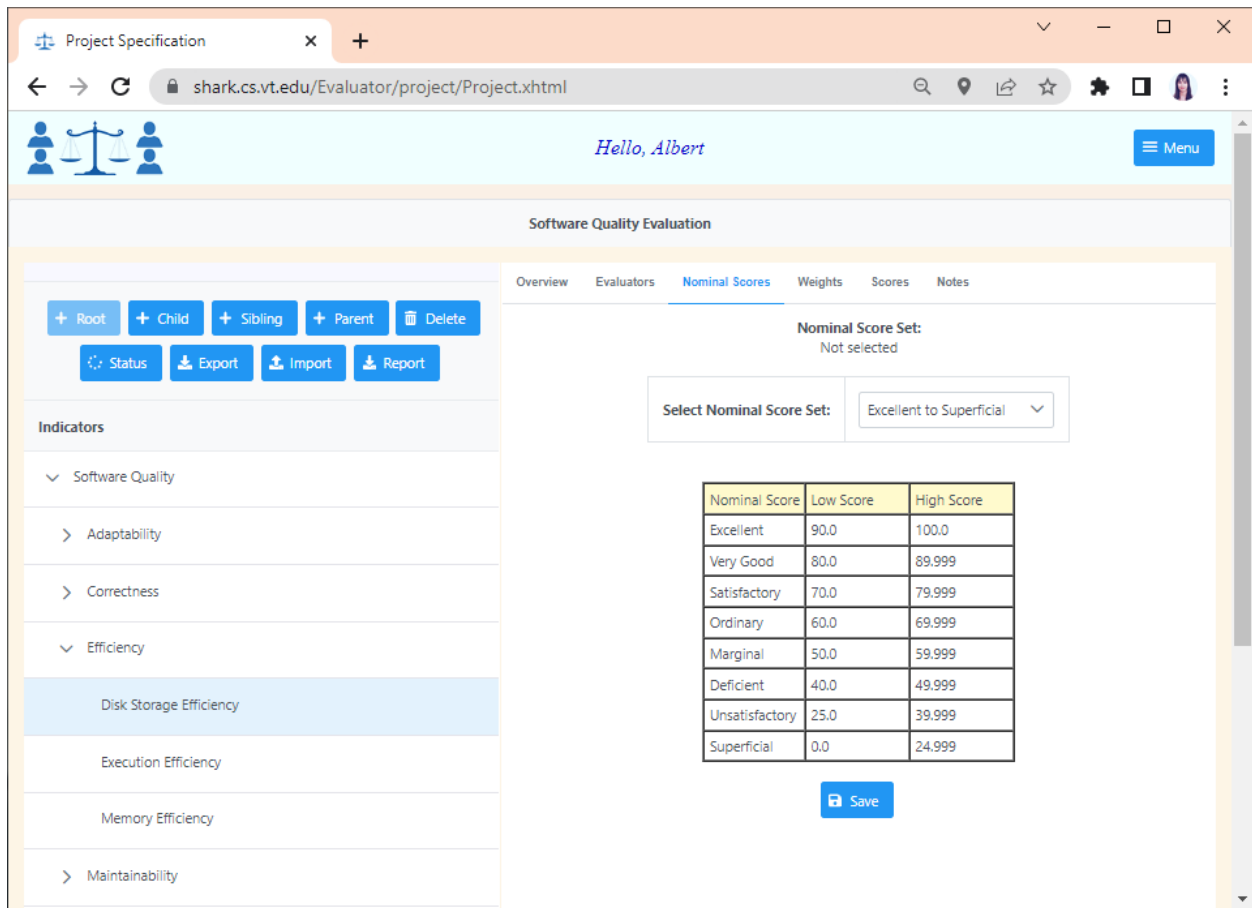


Figure 28. Assigning One of the Score Sets to a Leaf Indicator

Score Sets List

shark.cs.vt.edu/Evaluator/scoreSet/List.xhtml

Hello, Albert

Menu

List of Nominal Score Sets

Enter Global Search Query [Clear All Filters](#)

« < 1 2 3 > » 5 ▾

Title ↓	Definition																																																
Excellent to Unacceptable	<table border="1"> <thead> <tr> <th>Nominal Score</th> <th>Low Score</th> <th>High Score</th> </tr> </thead> <tbody> <tr><td>Excellent +</td><td>97.0</td><td>100.0</td></tr> <tr><td>Excellent</td><td>94.0</td><td>96.999</td></tr> <tr><td>Excellent –</td><td>90.0</td><td>93.999</td></tr> <tr><td>Good +</td><td>87.0</td><td>89.999</td></tr> <tr><td>Good</td><td>84.0</td><td>86.999</td></tr> <tr><td>Good –</td><td>80.0</td><td>83.999</td></tr> <tr><td>Satisfactory +</td><td>77.0</td><td>79.999</td></tr> <tr><td>Satisfactory</td><td>74.0</td><td>76.999</td></tr> <tr><td>Satisfactory –</td><td>70.0</td><td>73.999</td></tr> <tr><td>Poor –</td><td>67.0</td><td>69.999</td></tr> <tr><td>Poor</td><td>64.0</td><td>66.999</td></tr> <tr><td>Poor +</td><td>60.0</td><td>63.999</td></tr> <tr><td>Unacceptable –</td><td>40.0</td><td>59.999</td></tr> <tr><td>Unacceptable</td><td>20.0</td><td>39.999</td></tr> <tr><td>Unacceptable +</td><td>0.0</td><td>19.999</td></tr> </tbody> </table>	Nominal Score	Low Score	High Score	Excellent +	97.0	100.0	Excellent	94.0	96.999	Excellent –	90.0	93.999	Good +	87.0	89.999	Good	84.0	86.999	Good –	80.0	83.999	Satisfactory +	77.0	79.999	Satisfactory	74.0	76.999	Satisfactory –	70.0	73.999	Poor –	67.0	69.999	Poor	64.0	66.999	Poor +	60.0	63.999	Unacceptable –	40.0	59.999	Unacceptable	20.0	39.999	Unacceptable +	0.0	19.999
Nominal Score	Low Score	High Score																																															
Excellent +	97.0	100.0																																															
Excellent	94.0	96.999																																															
Excellent –	90.0	93.999																																															
Good +	87.0	89.999																																															
Good	84.0	86.999																																															
Good –	80.0	83.999																																															
Satisfactory +	77.0	79.999																																															
Satisfactory	74.0	76.999																																															
Satisfactory –	70.0	73.999																																															
Poor –	67.0	69.999																																															
Poor	64.0	66.999																																															
Poor +	60.0	63.999																																															
Unacceptable –	40.0	59.999																																															
Unacceptable	20.0	39.999																																															
Unacceptable +	0.0	19.999																																															
Excellent to Superficial	<table border="1"> <thead> <tr> <th>Nominal Score</th> <th>Low Score</th> <th>High Score</th> </tr> </thead> <tbody> <tr><td>Excellent</td><td>90.0</td><td>100.0</td></tr> <tr><td>Very Good</td><td>80.0</td><td>89.999</td></tr> <tr><td>Satisfactory</td><td>70.0</td><td>79.999</td></tr> <tr><td>Ordinary</td><td>60.0</td><td>69.999</td></tr> <tr><td>Marginal</td><td>50.0</td><td>59.999</td></tr> <tr><td>Deficient</td><td>40.0</td><td>49.999</td></tr> <tr><td>Unsatisfactory</td><td>25.0</td><td>39.999</td></tr> <tr><td>Superficial</td><td>0.0</td><td>24.999</td></tr> </tbody> </table>	Nominal Score	Low Score	High Score	Excellent	90.0	100.0	Very Good	80.0	89.999	Satisfactory	70.0	79.999	Ordinary	60.0	69.999	Marginal	50.0	59.999	Deficient	40.0	49.999	Unsatisfactory	25.0	39.999	Superficial	0.0	24.999																					
Nominal Score	Low Score	High Score																																															
Excellent	90.0	100.0																																															
Very Good	80.0	89.999																																															
Satisfactory	70.0	79.999																																															
Ordinary	60.0	69.999																																															
Marginal	50.0	59.999																																															
Deficient	40.0	49.999																																															
Unsatisfactory	25.0	39.999																																															
Superficial	0.0	24.999																																															
Excellent to Poor	<table border="1"> <thead> <tr> <th>Nominal Score</th> <th>Low Score</th> <th>High Score</th> </tr> </thead> <tbody> <tr><td>Excellent</td><td>80.0</td><td>100.0</td></tr> <tr><td>Good</td><td>60.0</td><td>79.999</td></tr> <tr><td>Average</td><td>40.0</td><td>59.999</td></tr> <tr><td>Marginal</td><td>20.0</td><td>39.999</td></tr> <tr><td>Poor</td><td>0.0</td><td>19.999</td></tr> </tbody> </table>	Nominal Score	Low Score	High Score	Excellent	80.0	100.0	Good	60.0	79.999	Average	40.0	59.999	Marginal	20.0	39.999	Poor	0.0	19.999																														
Nominal Score	Low Score	High Score																																															
Excellent	80.0	100.0																																															
Good	60.0	79.999																																															
Average	40.0	59.999																																															
Marginal	20.0	39.999																																															
Poor	0.0	19.999																																															

Figure 29. Some Nominal Score Set Examples

The Scores section shows the names of the evaluators assigned to the selected indicator and the score they have given to the project in terms of this indicator if the selected indicator is a leaf indicator (Figure 30). If assigned evaluators have not yet scored the project for this indicator, it shows the “not evaluated” expression. If the selected indicator is not a leaf indicator, which means a branch indicator, scores propagated from leaf indicators are displayed in the Scores section. Each of these scores is the evaluation of the project in terms of the selected indicator, as in the leaf indicators. If the scores in the leaf indicators have not been propagated yet, “not evaluated” is displayed in the scores section of the branch indicators as well.

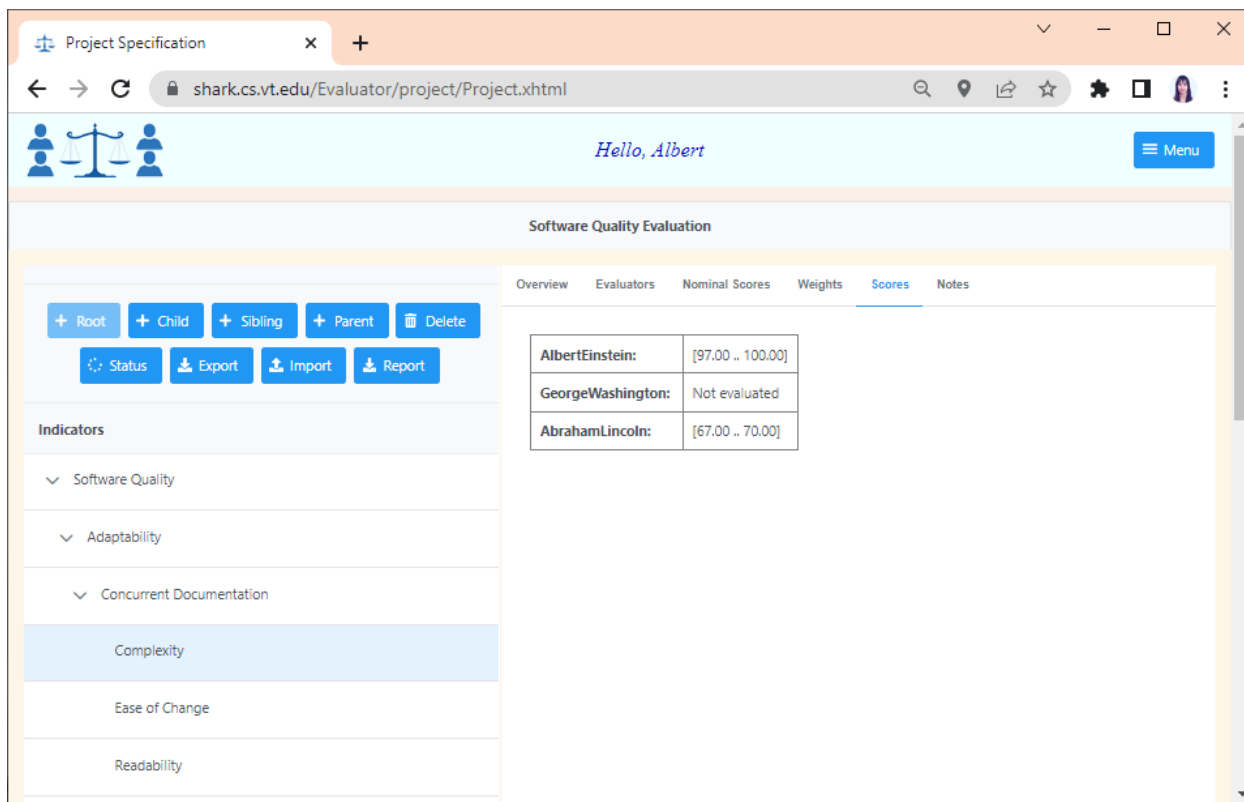


Figure 30. Scores of the Complexity indicator

If the selected indicator is a leaf indicator and the project is scored in terms of this indicator by the assigned evaluators, the Notes section displays the names of the evaluators assigned to the selected indicator and their notes on the score they gave to the project in terms of this indicator (Figure 31).

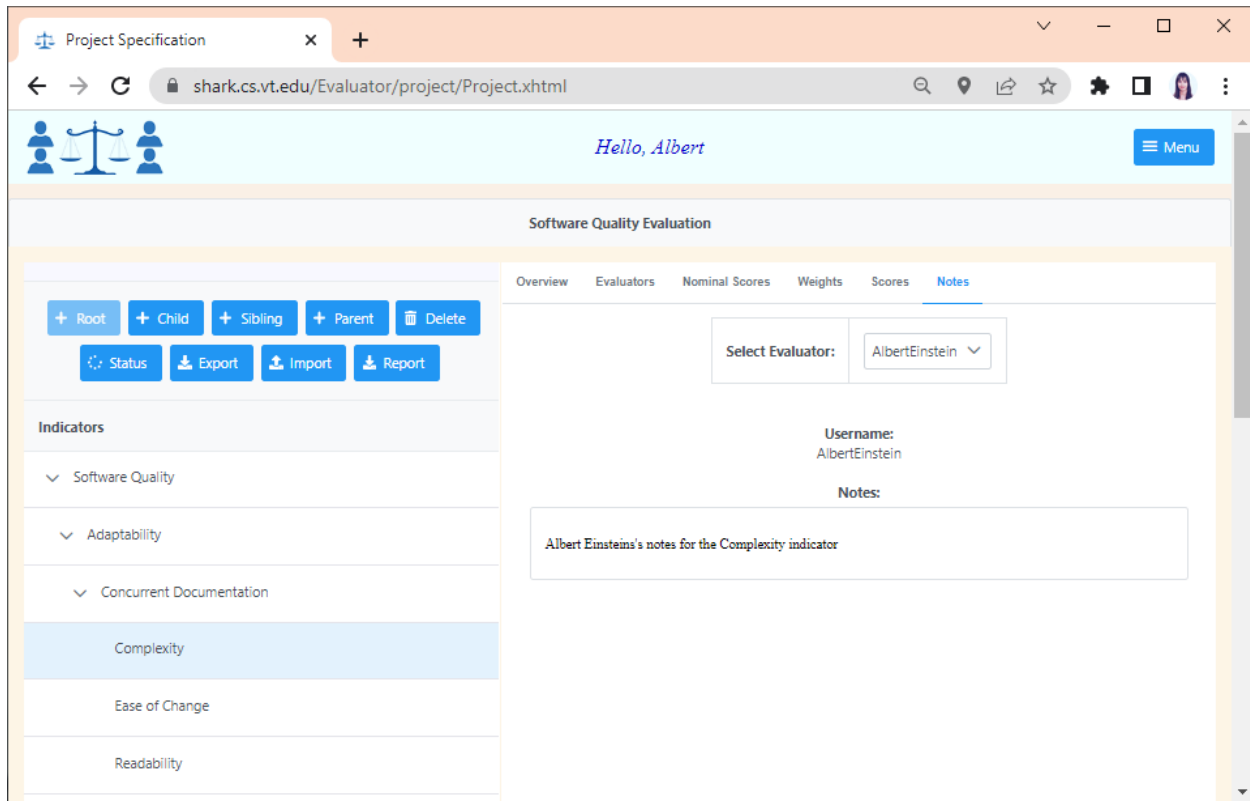


Figure 31. Notes of the Complexity indicator

The Evaluation Status section shows whether the project is ready to propagate scores. If at least one evaluator has not been assigned to all leaf indicators, it shows "Some indicators have not been assigned any evaluators" (Figure 32). If a leaf indicator has not been scored for by all the evaluators assigned to it, the status is displayed as "Some indicators have not yet been scored. Following evaluators have not completed scoring the indicators assigned to them" followed by the names of the evaluators who have not completed scoring yet (Figure 33). Project is ready to propagate the scores if at least one evaluator is assigned to all leaf indicators and all assigned evaluators have completed scoring. In this case, the Status section shows "All indicators have been scored by all assigned evaluators. The indicators graph is ready to propagate the scores" (Figure 34). Therefore, the administrator can compute the scores for all indicators and the final score for the project by pressing the Propagate Scores button.

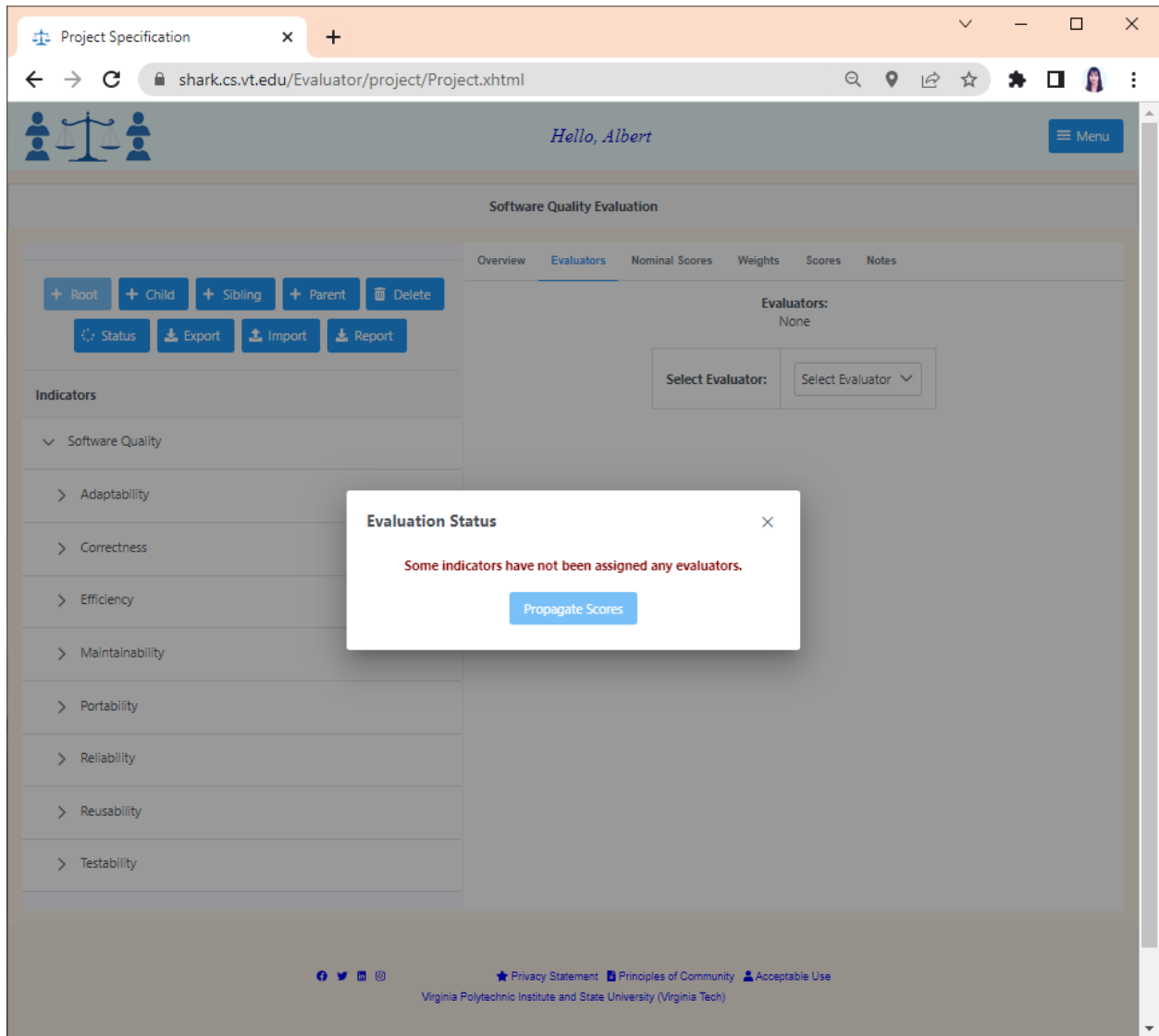


Figure 32. Evaluation Status Interface

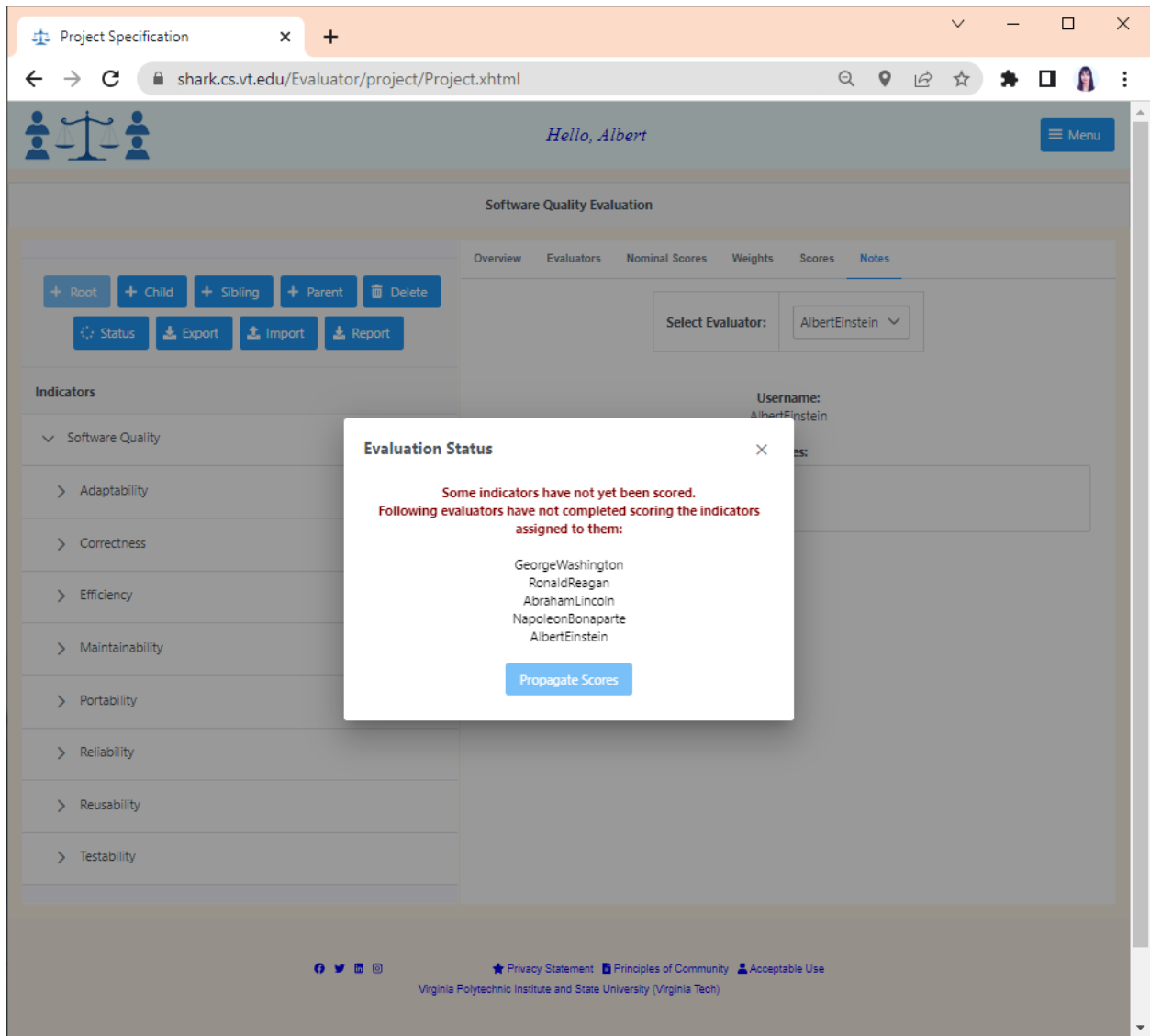


Figure 33. Evaluation Status Interface

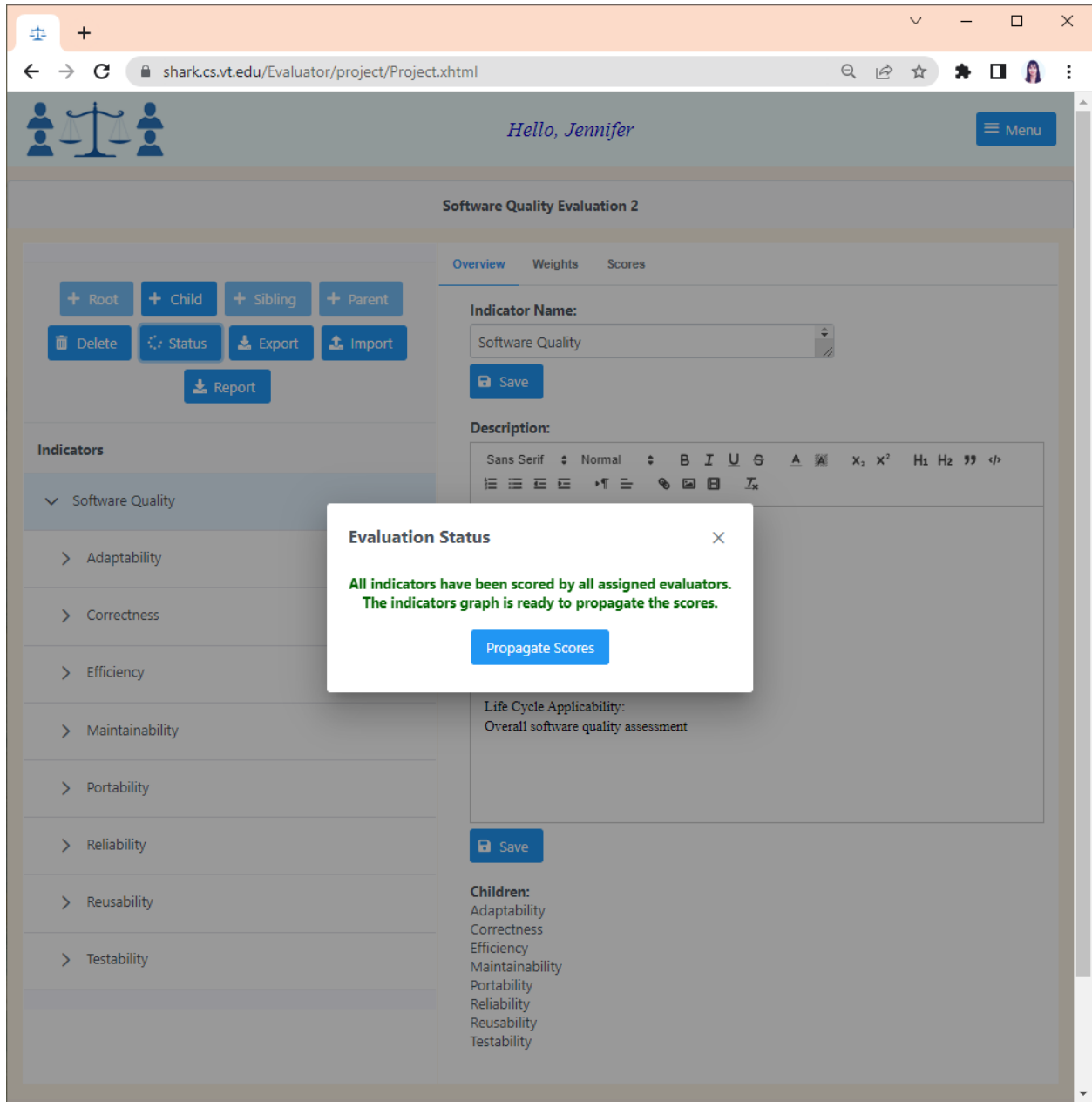


Figure 34. Evaluation Status Interface

The report functionality makes it possible to download a PDF file containing all the information related to the evaluation project and its indicators hierarchy (Figure 35). It consists of the following sections:

- Project name
- Project description
- Indicators hierarchy
- A list of all indicators with the details of their

- descriptions, child indicators, aggregate scores, weights, and parent indicators for the branch indicators,
- descriptions, evaluators' weights and scores, parent indicators, and notes for the leaf indicators.

Parent Indicators:

Parent Indicators
Adaptability
Correctness
Maintainability
Adaptability
Correctness
Reliability
Testability

Correctness:

Description:

Definition:
Correctness refers to strict adherence to specifications.

Source:
Arthur, J.D. and R.E. Nance (1991), "A Framework for Assessing the Adequacy and Effectiveness of Software Development Methodologies," Technical Report SRC-91-005, Systems Research Center, Virginia Tech, Blacksburg, VA.

Measurement:
Aggregation of scores

Life Cycle Applicability:
Software quality assessment

Child Indicators, Aggregate Scores, and Weights:

Child Indicator Name	Aggregate Score	Weight
Hierarchical Decomposition	0.250	[14.218 .. 14.723]
Life Cycle Verification	0.250	[0.000 .. 0.000]
Stepwise Refinement	0.250	[18.958 .. 19.631]
Structured Programming	0.250	[43.024 .. 44.536]

Figure 35. A Page from a Project Report

5.6 Evaluation

After an indicators hierarchy has been created by the project administrator, and one score set and at least one evaluator has been assigned to all leaf indicators in this hierarchy, the project is ready for conducting the evaluation. The first step of this evaluation is for the evaluators to evaluate the project by giving scores to it in terms of the leaf indicators to which they are assigned. In order to score the project in terms of these leaf indicators, evaluators select the project they will evaluate from the projects on their Evaluate page (Figure 19). Thus, they can see the leaf indicators they have been assigned in the indicator hierarchy of that project (Figure 20). Evaluators will see the leaf indicators to which they are assigned in dark red if they have not yet scored them, and in dark green if they have already scored them. Thus, the evaluators start the evaluation process by selecting the dark red indicators from this list.

The evaluator selects one of the scores of the assigned score set to evaluate the project in terms of the selected leaf indicator (Figure 36). After the score is given, the evaluator writes their evaluation notes for that indicator and saves the score (Figure 37).

Once all leaf indicators in the indicator hierarchy are scored by all the evaluators assigned to them, the project is ready to propagate the scores. In the propagation process, the scores given to the project in terms of leaf indicators are propagated up from the lowest level of the hierarchy to compute the final evaluation score of the root indicator. Administrators can see if the project is ready for score propagation using the project's Status functionality (Figure 34).

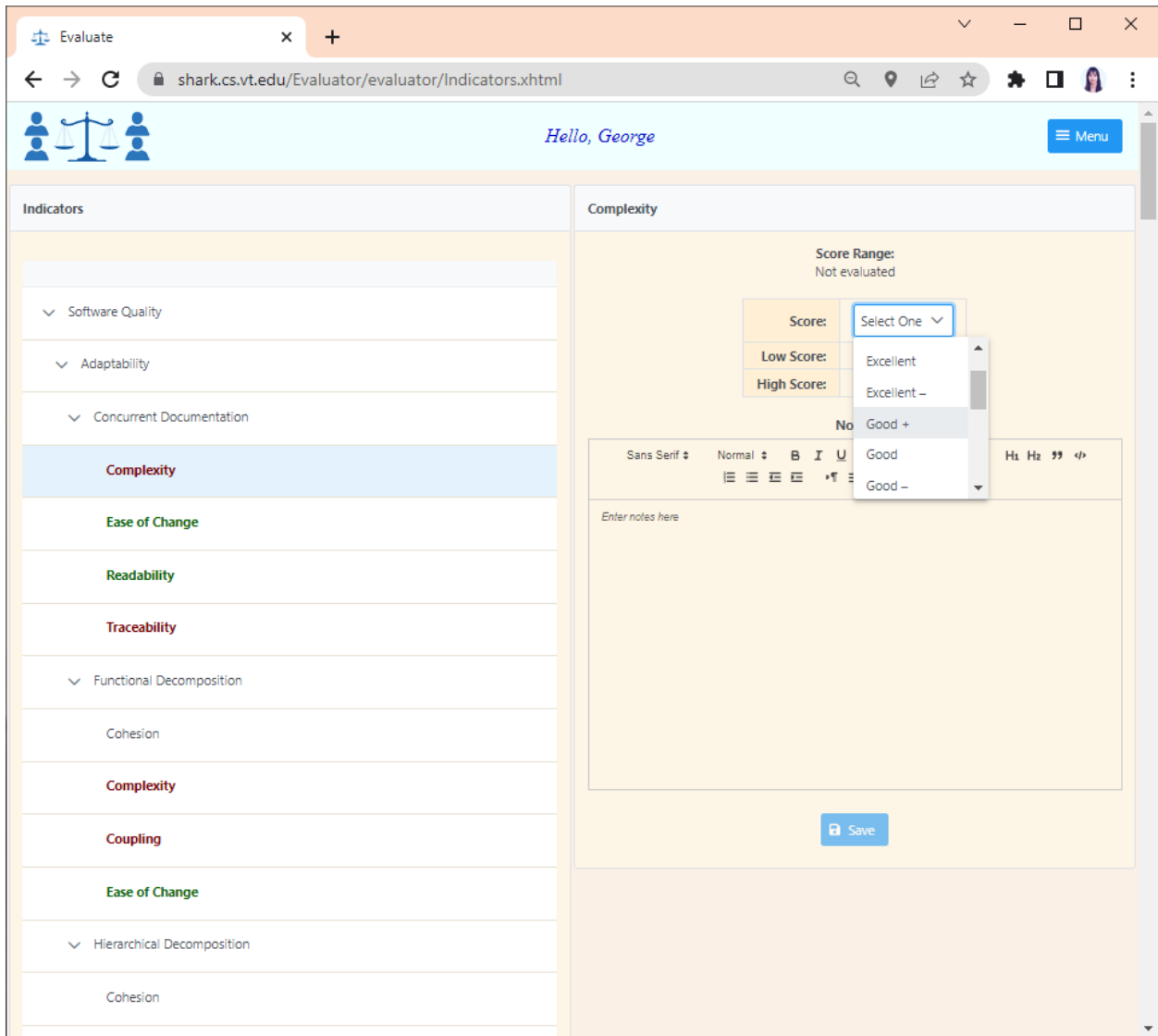


Figure 36. Evaluator Selects a Score

The screenshot shows a web browser window with the URL `shark.cs.vt.edu/Evaluator/evaluator/Indicators.xhtml`. The page title is "Evaluate" and the user is logged in as "George". The interface is divided into two main sections: "Indicators" and "Complexity".

Indicators List:

- Software Quality
- Adaptability
- Concurrent Documentation
- Complexity** (Selected)
- Ease of Change
- Readability
- Traceability
- Functional Decomposition
 - Cohesion
- Complexity
- Coupling
- Ease of Change
- Hierarchical Decomposition
 - Cohesion

Complexity Indicator Details:

- Score Range:** Not evaluated
- Score:** Good + (dropdown menu)
- Low Score:** 87.0
- High Score:** 89.999
- Notes:**
 - Rich text editor toolbar: Sans Serif, Normal, Bold, Italic, Underline, Text Color, Background Color, Text Size, Font Size, Bold, Italic, Underline, Link, Unlink, Undo, Redo.
 - Note content: George Washington's notes for the Complexity indicator
- Save** button

Figure 37. Evaluator Score and Notes

Chapter 6: Evaluator Self-Assessment

Self-assessment of Evaluator has been done using Evaluator by creating a project called Software Quality Evaluation. We created ten sample user accounts to represent the administrators and evaluators of this project. It has five project administrators and five project evaluators (Figure 38).

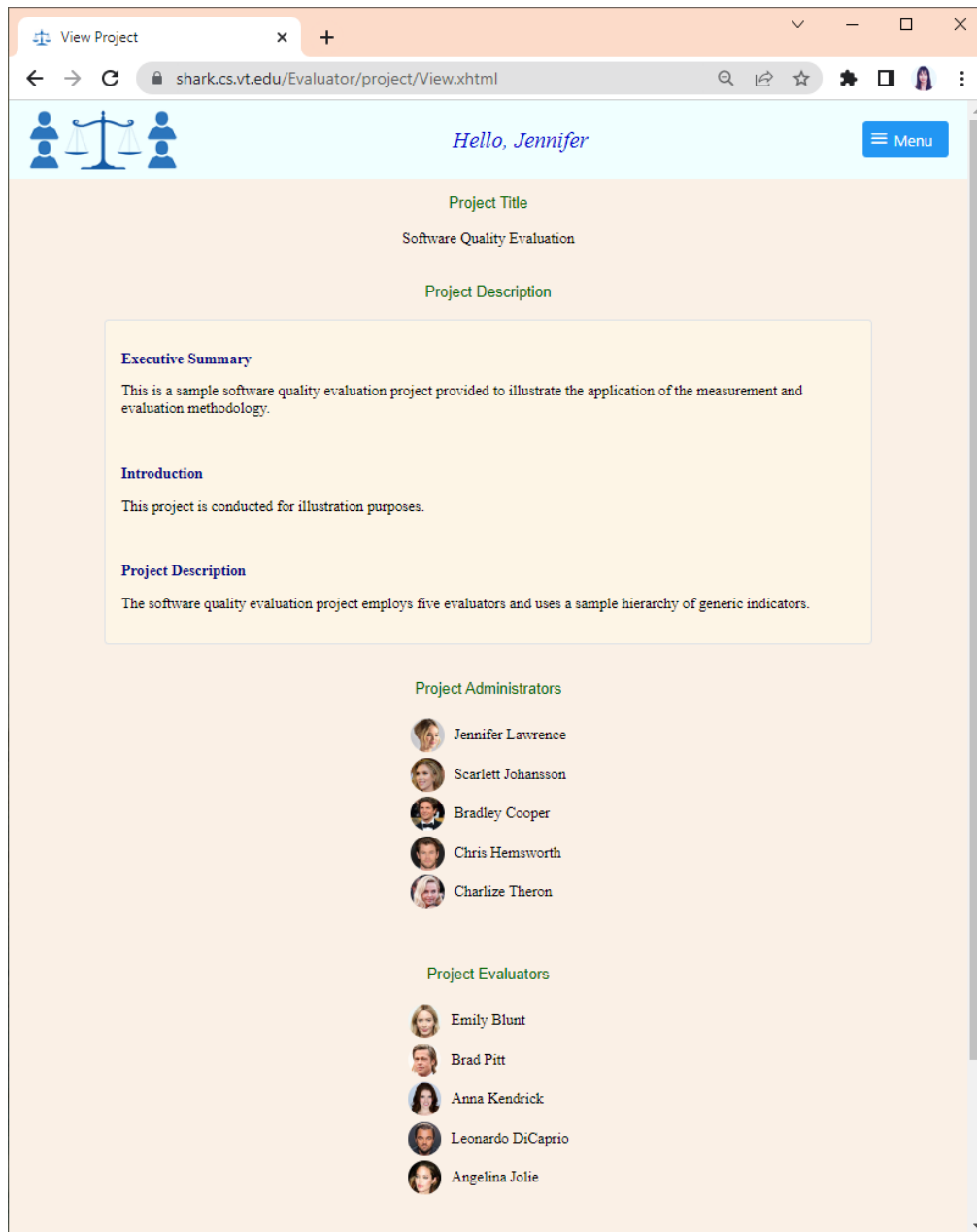


Figure 38. Project Documentation of Software Quality Evaluation

We created the project's indicators hierarchy using one of the project Administrator accounts. Accordingly, software quality is represented by eight different indicators with certain weights. Each of these indicators is divided into various sub-indicators. Hence, all indicators except leaf indicators consist of simpler sub-indicators that represent them (Figure 39).

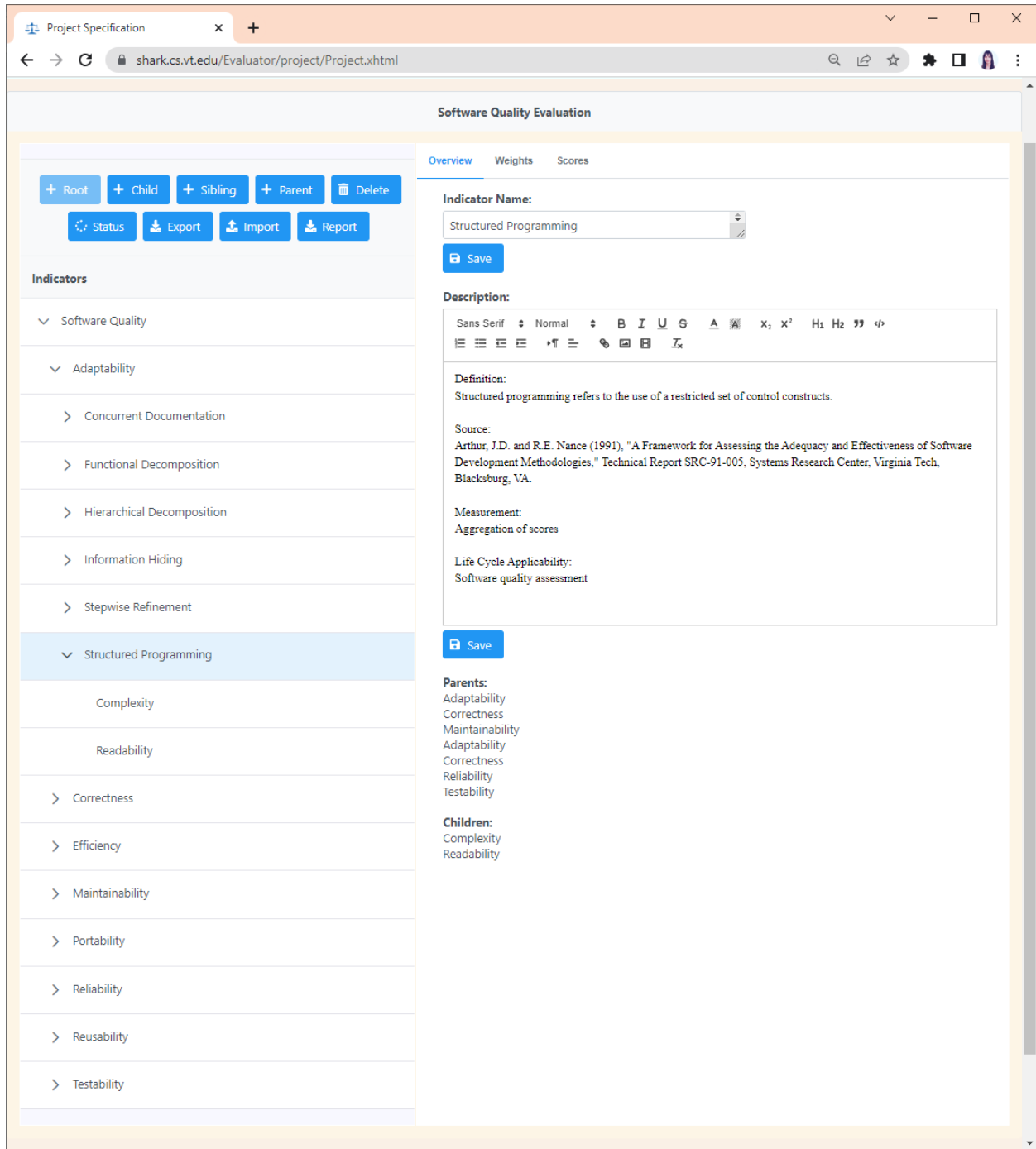


Figure 39. Indicators Hierarchy of Software Quality Evaluation

According to the hierarchy of indicators, the indicators that directly affect software quality are adaptability, correctness, efficiency, maintainability, portability, reliability, reusability, and correctness.

There are 12 leaf indicators that are not split into any sub-indicators. These are complexity, ease of change, readability, traceability, cohesion, coupling, well-defined interfaces, early error detection, visibility behavior, disk storage efficiency, execution efficiency, and memory efficiency [Arthur and Nance 1991]. At least one evaluator was assigned to each of these leaf indicators by administrators according to their area of expertise (Figure 40).

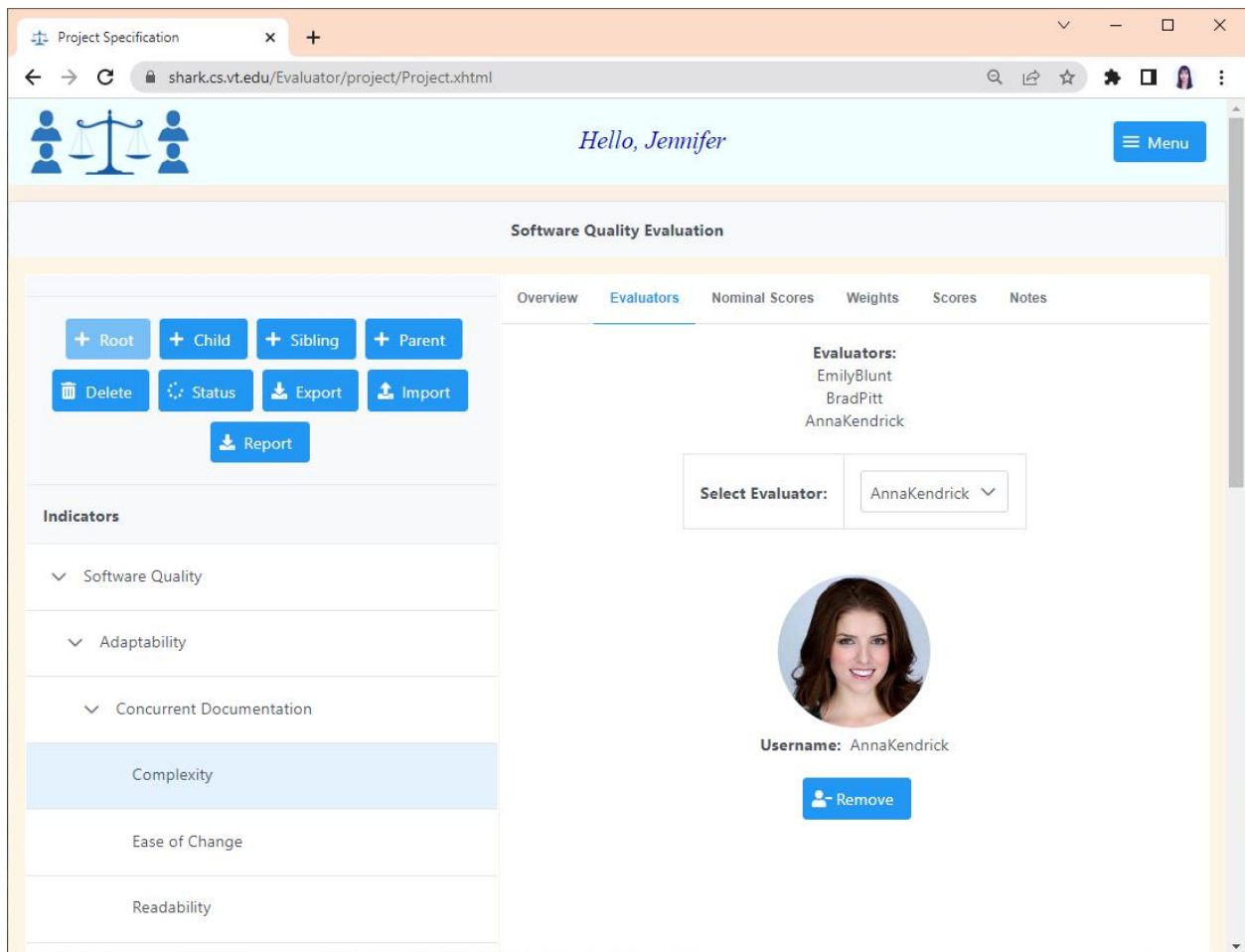


Figure 40. Evaluators Assigned to the Complexity Indicator

A score set is selected for all leaf indicators. As the lower complexity and coupling the more score should be given to a software, Undesirable Trait score set was selected for these two leaf indicators. Excellent to Poor score set was selected for all remaining leaf indicators in accordance with the evaluation strategy that aligns with them (Figure 41).

The screenshot shows a web application titled "Software Quality Evaluation". The user is logged in as "Jennifer". The interface includes a navigation menu on the left with the following items: Software Quality, Adaptability, Concurrent Documentation, Complexity, Ease of Change, Readability (selected), Traceability, Functional Decomposition, Cohesion, Complexity, and Coupling. The main content area shows the "Nominal Score Set" for "Readability" is "Excellent to Unacceptable". A table below this shows the mapping of nominal scores to low and high scores.

Nominal Score	Low Score	High Score
Excellent +	97.0	100.0
Excellent	94.0	96.999
Excellent -	90.0	93.999
Good +	87.0	89.999
Good	84.0	86.999
Good -	80.0	83.999
Satisfactory +	77.0	79.999
Satisfactory	74.0	76.999
Satisfactory -	70.0	73.999
Poor -	67.0	69.999
Poor	64.0	66.999
Poor +	60.0	63.999
Unacceptable -	40.0	59.999
Unacceptable	20.0	39.999
Unacceptable +	0.0	19.999

Figure 41. Excellent to Unacceptable Score Set Selected for Readability

Sub-indicators of all branch indicators were compared in pairs with each other in terms of their effects on their parent indicator. Thus, the weight of each sub-indicator on the parent indicator was obtained (Figure 43). Similarly, the evaluators assigned to the leaf indicators were compared in pairs with each other in terms of their expertise on that leaf indicator (Figure 43).

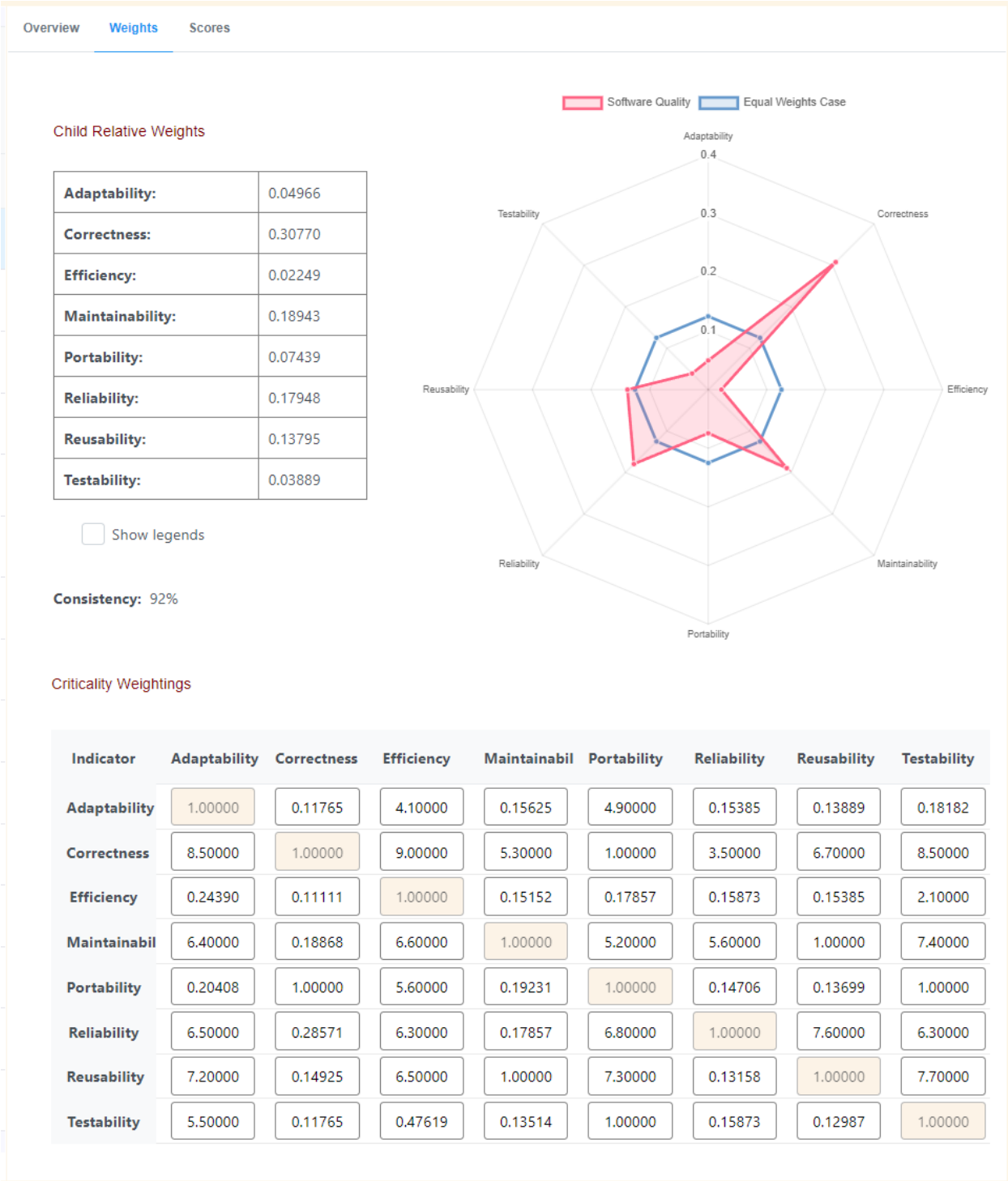


Figure 42. Weights of Main Indicators on Software Quality

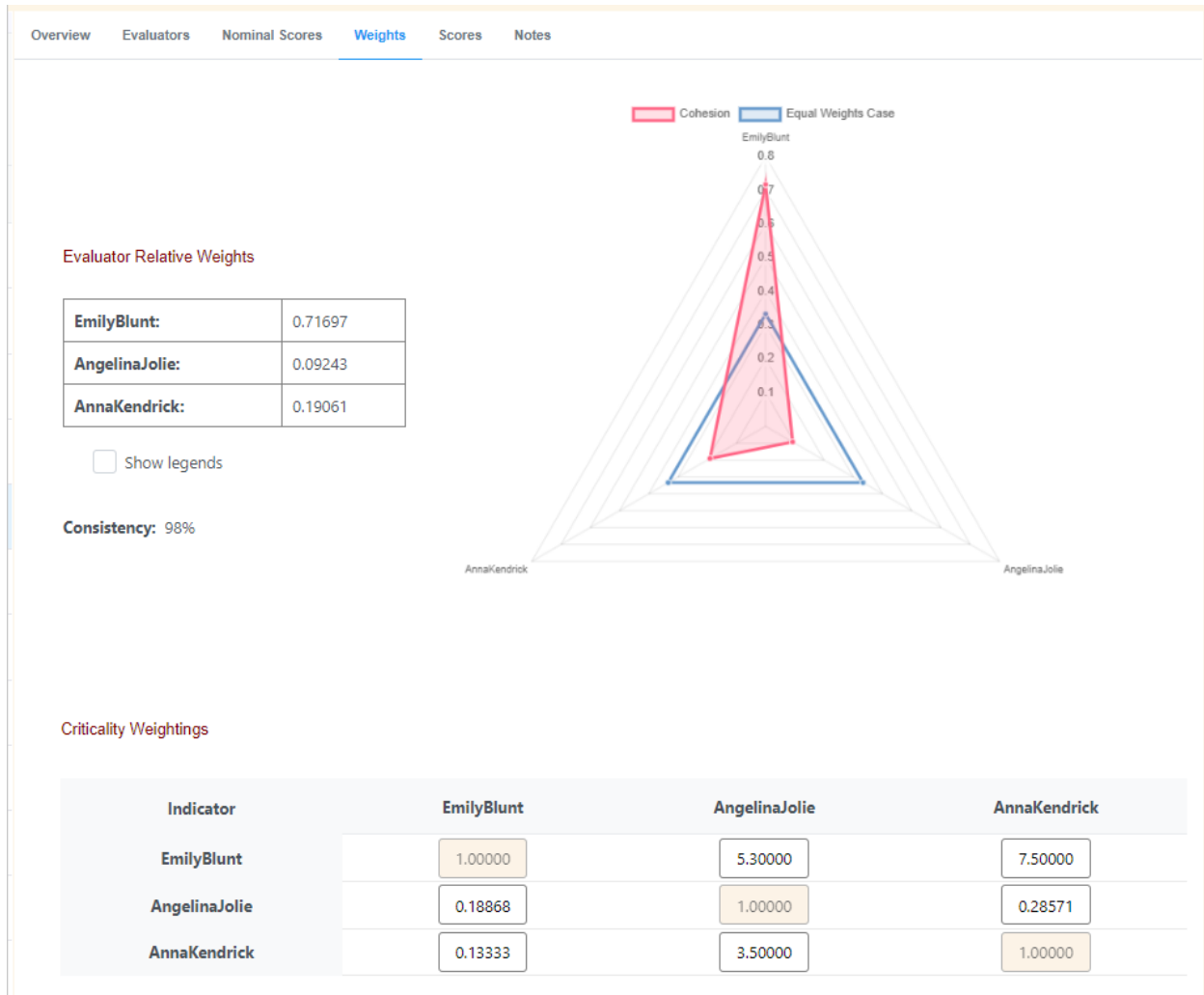


Figure 43. Weights of Assigned Evaluators on Cohesion

After the administrators determined the indicator hierarchy and evaluators, score sets, and weights of the indicators, the evaluators evaluated software quality in terms of each leaf indicator. In order to do this, each evaluator scored on all indicators to which they were assigned (Figure 44). While scoring the leaf indicators, the evaluation results described in the below section were considered.

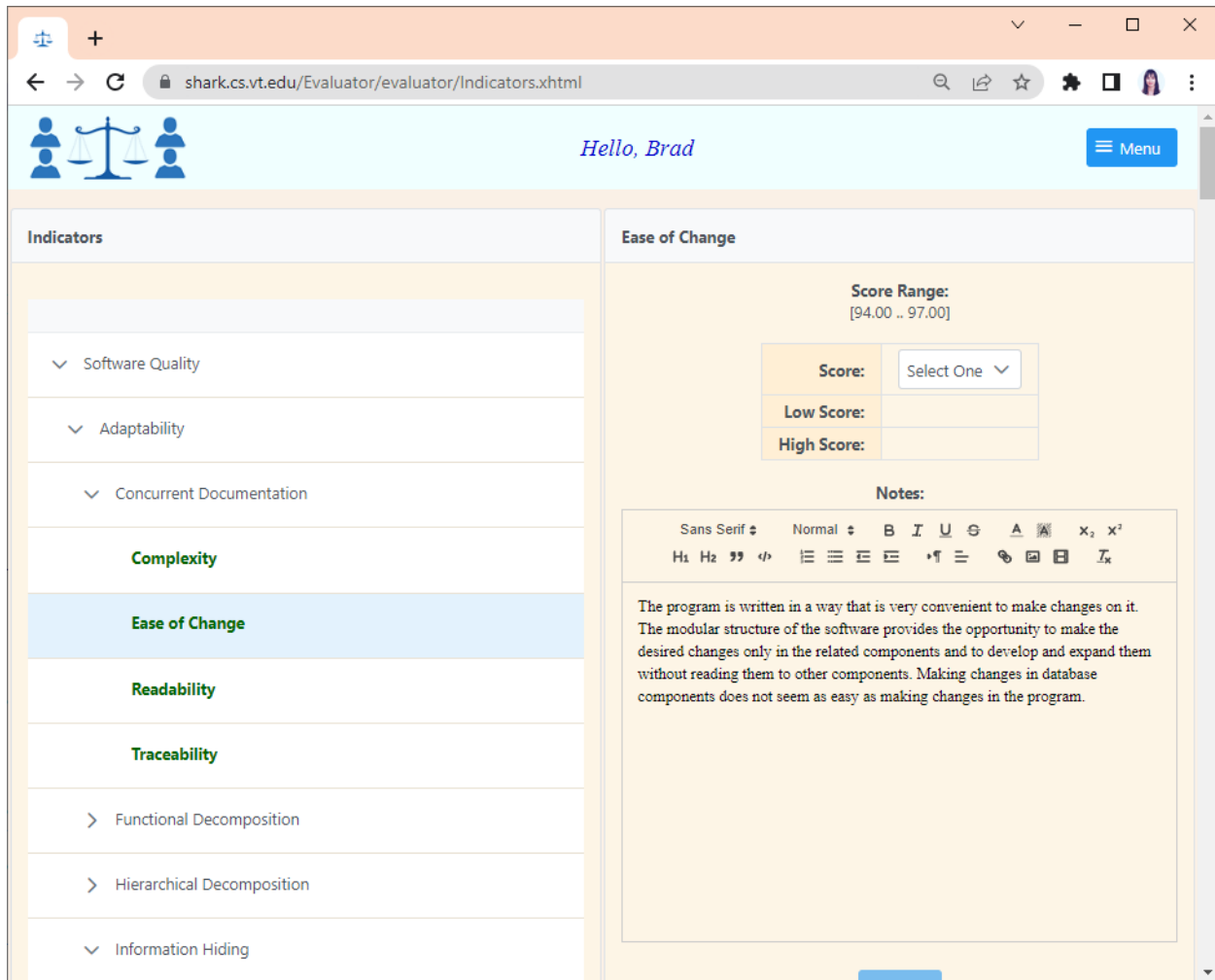


Figure 44. Evaluation of Leaf Indicators

6.1 Evaluations of Leaf Indicators

6.1.1 Complexity

Unnecessary complexity has been avoided and the code has been kept simple. This makes it very easy to understand the components of the program and adapt to the development process.

6.1.2 Ease of Change

The program is written in a way that is very convenient to make changes on it. The modular structure of the software provides the opportunity to make the desired changes only in the related components and to develop and expand them without reading them to other components. Making changes in database components does not seem as easy as making changes in the program.

6.1.3 Readability

Variables, classes, and functions are named in a meaningful way. Java Naming conventions were followed strictly. The comments are descriptive and in-depth. Code readability is very high.

6.1.4 Traceability

GitHub repository of the project has been used actively. Attention was paid to version control. Carefully written commit messages make it easy to follow the history of the project, the changes it has gone through, and the improvements made.

6.1.5 Cohesion

Functional cohesion was achieved in almost all parts of the project. Modules consist of elements that are meaningful to be together.

6.1.6 Coupling

Coupling between classes was very low. Still, some visual components were still coupled with functionality classes.

6.1.7 Well-Defined Interfaces

Inter-component interfaces communicated functional purposes very well. Only the necessary information was shared with the right components.

6.1.8 Early Error Detection

Very detailed and complete documentation of requirement specifications are provided prior to implementation of the project. Developers made sure the requirements were understood correctly.

6.1.9 Visibility Behavior

The development log shows periodic behavior reviews and feedback from the thesis advisor Dr. Osman Balci ensured the final product's behavioral consistency.

6.1.10 Disk Storage Efficiency

Disk usage of the developed software is minimal. Storing serialized bit code representations of the projects saves a lot of disk space.

6.1.11 Execution Efficiency

The execution efficiency of the application was very high. Even with the most complex evaluation graphs, the system responded within reasonable execution times.

6.1.12 Memory Efficiency

Using recursion in many functions because of the tree-like structure of the hierarchy of indicators comes with a stack overhead in the memory. It is not prominent but still present.

6.2 Evaluations of Branch Indicators

After the evaluation of all leaf indicators, the scores were propagated to the upper indicators in the hierarchy using the AHP algorithm. Thus, weighted scores were calculated for the “concurrent documentation, functional decomposition, hierarchical decomposition, information hiding, stepwise refinement, structured programming, and life cycle verification” indicators [Arthur and Nance 1991] indicators as well as the eight main indicators (Figure 45).

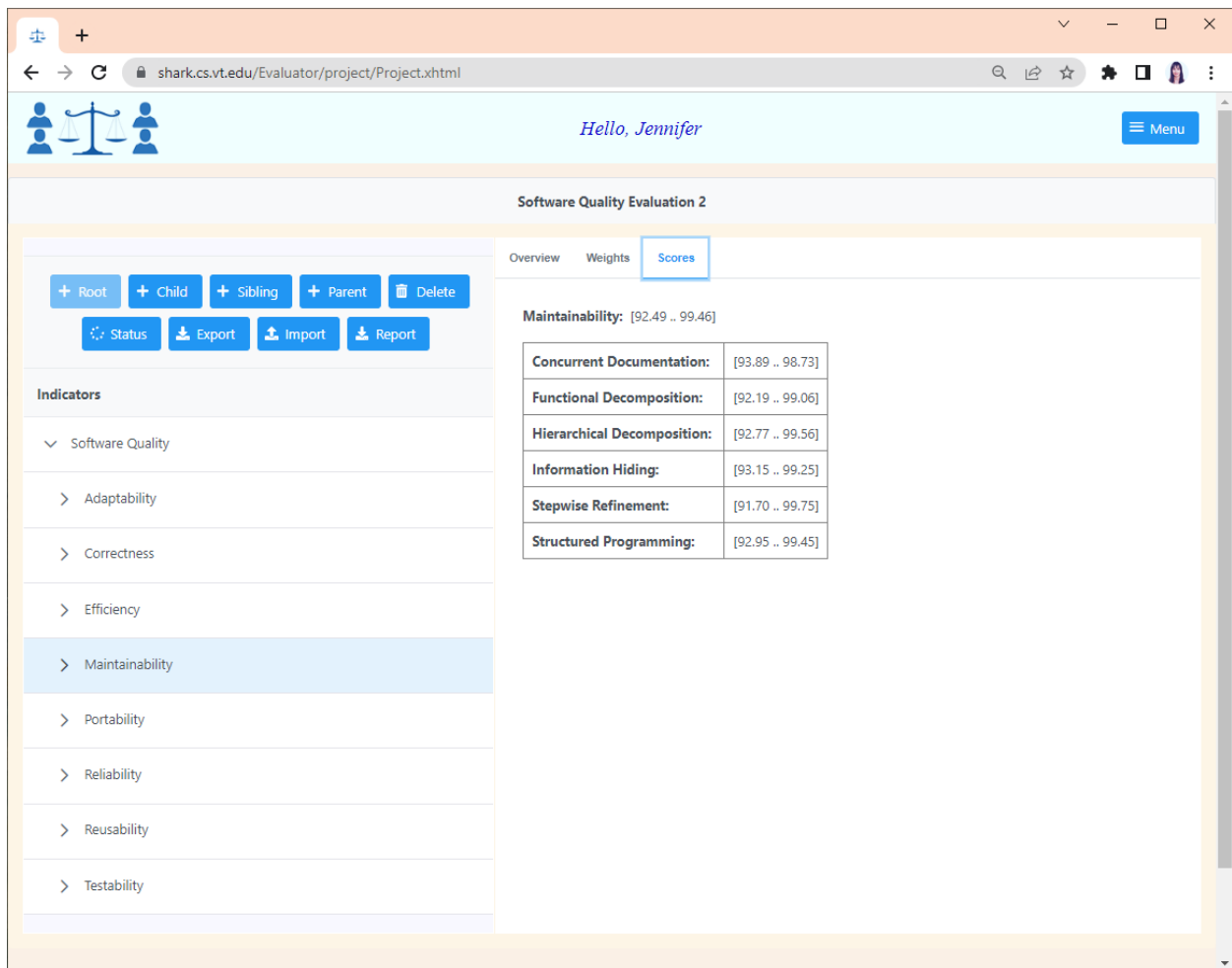


Figure 45. Computed Score of the Maintainability Indicator

6.3 Evaluation of Software Quality

After all scores are given for the leaf indicators and computed for the branch indicators, software quality was eventually evaluated. Scores given by the five different evaluators in accordance with the evaluations stated in 6.1 were used to calculate the ultimate software quality score. Accordingly, Evaluator has a score of 96.175 with a low score of 93 and a high score of 99.35 out of 100 in terms of software quality (Figure 46).

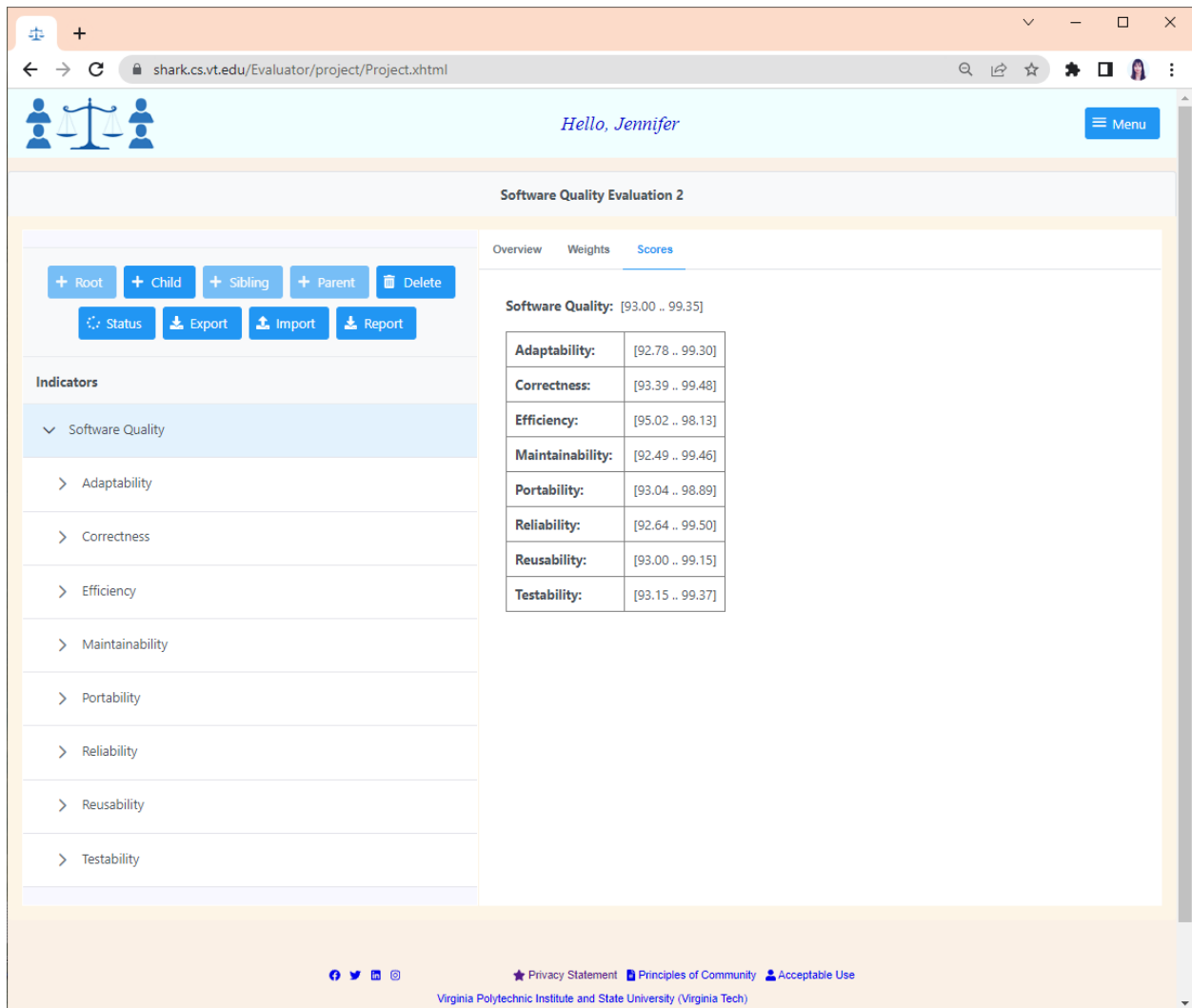


Figure 46. Software Quality Score of Evaluator

Chapter 7: Conclusions and Future Research

7.1 Conclusions

In this thesis, a cloud-based application of a collaborative evaluation methodology is presented. By utilizing the benefits of cloud software features, Evaluator offers a scientific assessment tool with cross platform capabilities and synchronization across multiple devices. It provides a paradigm that modifies AHP's hierarchical structure and weighted score propagation to allow multiple evaluators to evaluate the same system together. With this methodology, it is ensured that the evaluation process is carried out without errors by simplifying it for the evaluators. It enables the users construct an evaluation project, create the hierarchy of indicators, assess the overall project by creating the evaluator scores from bottom to the top, and generate the evaluation report according to the roles of the user accounts. Additionally, it safeguards the privacy of the user accounts via SHA-1 encryption while greatly simplifying the business processes.

The creation of the evaluator is based on the phases of the software development life cycle [Balci 2012], namely, “problem formulation, requirements engineering, architecture design, software components design, and development”. This procedure incorporates verification, validation, and quality testing. Evaluator is carefully designed, implemented, and tested by following this life cycle.

With evaluator, project administrators and evaluators are able to play distinct responsibilities while working on an evaluation project. A user-friendly user interface allows administrators to choose the indications, score sets, evaluators, and other specifics that will be used in the assessment of a project. Thus, evaluators can evaluate the project using simplified indicator scoring. Users may access all application data whenever and wherever they choose thanks to synchronization across all devices.

7.2 Future Research

The subject of this research is a cloud-based application implementing a collaborative evaluation methodology. Future research can focus on:

1. Provide a visualization of the hierarchy shown as a tree as an acyclic graph to show all the links more clearly between the indicators.
2. Include charts that visualize the comparisons of indicators and evaluators among themselves in project reports.
3. In addition to export-import functionality, provide conveniences such as copy-paste or drag-drop of a branch for modifying the hierarchy.
4. Refine the user interface as the User Interface/User Experience UI/UX design principles improve in order to provide users with a simplified experience for the complex evaluation processes.

REFERENCES

- Arthur, J.D. and R.E. Nance (1991), "A Framework for Assessing the Adequacy and Effectiveness of Software Development Methodologies," Technical Report SRC-91-005, Systems Research Center, Virginia Tech, Blacksburg, VA.
- Balci, O. (2001), "A methodology for certification of modeling and simulation applications," *ACM Transactions on Modeling and Computer Simulation*, Vol. 11, No. 4, pp.352–377.
- Balci, O. (2022), "CS5704 Software Engineering," <https://manta.cs.vt.edu/cs5704/>
- Balci, O., R. J. Adams, D. S. Myers, and R. E. Nance (2002), "A Collaborative Evaluation Environment for Credibility Assessment of Modeling and Simulation Applications," In *Proceedings of the 2002 Winter Simulation Conference* (San Diego, CA, Dec. 8-11). IEEE, Piscataway, NJ, pp. 214-220.
- Balci, O. and W.F. Ormsby (2008), "Network-centric military system architecture assessment methodology," *International Journal of System of Systems Engineering*, 1(1-2), 271-292.
- Booch, G., I. Jacobson, and J. Rumbaugh (1996), "The unified modeling language," *Unix Review*, 14(13), 5.
- Eastlake 3rd, D. and P. Jones (2001), *US secure hash algorithm 1 (SHA1)* (No. RFC 3174).
- Eder, J., G. Kappel, and M. Schrefl (1994), "Coupling and cohesion in object-oriented Systems," Technical Report, University of Klagenfurt.
- Oracle (2018), "The Client Tier," <https://docs.oracle.com/cd/E19226-01/820-7759/gcr1a/index.html>
- PrimeFaces (2022), "PrimeFaces for JSF," <https://www.primefaces.org/#primefaces>
- Saaty, T.L. (1980), *The Analytic Hierarchy Process*, New York, NY: McGraw-Hill.
- Schmidt, D. C., M. Stal, H. Rohnert, and F. Buschmann (2013), "Pattern-Oriented Software Architecture," *Patterns for Concurrent and Networked Objects*. John Wiley & Sons.
- Talbert, M.L. (1995), "A methodology for the measurement and evaluation of complex system designs," PhD Dissertation, Department of Computer Science, Virginia Tech, Blacksburg, VA, December.