Abstract
This document contains the software requirements for the Delta extension of the APD tool, developed by the Delta team. The requirements in this Software Requirements Document (SRD) satisfy the requirements set in the User Requirements Document [1]. This document complies with the Software Engineering Standard, as specified by the European Space Agency [2].
1 Introduction
1.1 Purpose ............................................. 6
1.2 Scope .............................................. 6
1.3 List of definitions and abbreviations ................. 6
   1.3.1 List of definitions ................................. 6
   1.3.2 List of abbreviations ............................. 8
1.4 List of references ................................... 8
1.5 Overview ........................................... 9

2 General Description .................................. 10
2.1 Relation to current projects .......................... 10
2.2 Relation to predecessor and successor projects ....... 10
2.3 Function and purpose ................................ 10
2.4 Environment ....................................... 11
2.5 Relation to other systems ............................ 12
2.6 General constraints .................................. 12
   2.6.1 Environment .................................... 12
   2.6.2 Privacy .......................................... 12
   2.6.3 Performance .................................... 12
2.7 Model description .................................... 12
   2.7.1 Class diagram ................................... 12
   2.7.2 Data model ...................................... 23
   2.7.3 Sequence diagrams ............................... 28

3 Specific requirements ................................. 60
3.1 Functional requirements .............................. 60

4 Requirements traceability matrix ....................... 78
4.1 Software requirements to user requirements ........... 78
4.2 User requirements to software requirements .......... 80

A User interface mock-ups ............................... 83
A.1 Login page .......................................... 83
A.2 Registration page ................................... 83
A.3 Home page ......................................... 84
A.4 Project page ........................................ 86
A.5 Results page ....................................... 89
A.6 User information page ............................... 90
A.7 Admin tools ........................................ 90

B Transitions ............................................ 92
Document Status

General

Title of document  Software Requirements Document
Document identifier Delta.SRD/1.0
Version 1.0
Authors
D.P. van den Berg
R.T. van Bergen
D.J.C. Dekker
S. van den Eerenbeemt
J. Mols
B.F. Rongen
B.W.M. van Rooijen
R.P. Schellekens
A.A. Vast
G. Walravens
S. Wessel

History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author(s)</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>2018-06-08</td>
<td>D.J.C. Dekker, S. van den Eerenbeemt, B.W.M. van Rooijen, S. Wessel</td>
<td>Third draft of SRD for feedback.</td>
</tr>
<tr>
<td>0.4</td>
<td>2018-06-15</td>
<td>D.J.C. Dekker, S. van den Eerenbeemt, B.W.M. van Rooijen, A.A. Vast</td>
<td>Fourth draft of SRD for feedback.</td>
</tr>
<tr>
<td>1.0</td>
<td>2018-06-28</td>
<td>D.J.C. Dekker, S. van den Eerenbeemt, B.W.M. van Rooijen</td>
<td>Final version of SRD.</td>
</tr>
</tbody>
</table>
## Document Change Records

<table>
<thead>
<tr>
<th>Section</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Section included some spelling mistakes.</td>
</tr>
<tr>
<td>2</td>
<td>Section included some spelling mistakes.</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Purpose

This Software Requirements Document (SRD) provides a translation from the user requirements specified in the User Requirement Document (URD) [1] to the software requirements for the Delta extension of the APD tool. Whereas the URD states the user requirements as specified by our customer, Laura Genga, this document state the software requirements that the developers of the Delta team place upon the system in order to fulfill the user requirements. These software requirements dictate, in developer terms, what the Delta extension of the APD tool must do, and not how. In other words, this document is implementation-independent. The software requirements specified in this document are modeled in a logical model, which provides a simplified view of the systems content and behavior.

1.2 Scope

The APD tool is designed to extract anomalous patterns together with their correlations. These patterns are extracted from historical logging data from past process executions. Users can upload event logs and process models on which experiments can be run. The results of these experiments can be shown after running [3].

The purpose of the Delta extension is to extend the currently existing APD tool with several features. First of all, this extension will support multiple users, which means the tool will have to have a solid workflow and proper user management controls. The user management system will be created and several improvements have to be made to the user interface in order to realize these goals.

Next to this, reporting on the progress of the experiments done in the tool is important, as well as the visualization of the results. The tool is currently able to discover patterns and subgraphs from a business process model and an event log. These patterns and subgraphs give information on the business process and anomalous traces in the event log. In the current version, however, comparing different result types is cumbersome. Delta will provide a more convenient way to manage experiments and their results as well as the projects they belong to.

Furthermore, the administrator will be able to manage the users and the projects created by users, as well as monitor the user activity tracking of each individual user. These controls will also be provided by Delta.

1.3 List of definitions and abbreviations

1.3.1 List of definitions

Administrator
A registered user with the highest available access rights who manages both the tool and its users.

Anomalous subgraph discovery
The extraction of recurrent subgraphs involving one or more deviations from the process model [3].

APD tool
The APD tool is an extension of the Esub tool designed to extract anomalous patterns together with their correlations. These patterns are extracted from historical logging data from past process executions. Users can upload event logs and process models on which experiments can be run. The results of these experiments can be shown after running [3].

Business process
A set of activities performed in an organization and technical environment that are coordinated to a certain product or service [4].
**Comparative results**  
After a batch of experiments has been run, the comparative results of this batch of experiments can be calculated. These comparative results include the average support value of the discovered anomalous subgraphs, the average support value of the discovered partial orders and the number of discovered partial orders per run experiment.

<table>
<thead>
<tr>
<th><strong>Component</strong></th>
<th>A part of a phase of an experiment.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Esub tool</strong></td>
<td>An online tool supporting the visualization and exploration of the outcome of the frequent subgraph mining algorithm SUBDUE [5].</td>
</tr>
<tr>
<td><strong>Experiment</strong></td>
<td>Both the anomalous subgraph discovery and partial order discovery together.</td>
</tr>
<tr>
<td><strong>Experiment log</strong></td>
<td>A file that tracks all activities performed within an experiment.</td>
</tr>
<tr>
<td><strong>Experiment phase</strong></td>
<td>Anomalous subgraph discovery and partial order discovery are the two phases of one experiment.</td>
</tr>
<tr>
<td><strong>Event log file</strong></td>
<td>A file that consists of traces [3].</td>
</tr>
<tr>
<td><strong>Intermediate result</strong></td>
<td>The outcome of either a component or an experiment phase.</td>
</tr>
<tr>
<td><strong>Partial order discovery</strong></td>
<td>An experiment phase creating patterns from anomalous subgraphs and partially ordering them based on their location in the log traces [3].</td>
</tr>
<tr>
<td><strong>Petri net</strong></td>
<td>A mathematical model used for the specification and the analysis of parallel processes [6].</td>
</tr>
<tr>
<td><strong>Process model</strong></td>
<td>A representation of the prescribed behavior of a business process [3].</td>
</tr>
<tr>
<td><strong>Project</strong></td>
<td>A combination of an event log, a process model, and a unique project name. The project is stored together with any experiments run under that project name.</td>
</tr>
<tr>
<td><strong>Project owner</strong></td>
<td>The user who created the project.</td>
</tr>
<tr>
<td><strong>Registered user</strong></td>
<td>A user with a registered account on the APD tool.</td>
</tr>
<tr>
<td><strong>Responsive</strong></td>
<td>A website is responsive when dynamic changes are made to the appearance of the site depending on the screen size and orientation of the device being used to view it [7].</td>
</tr>
<tr>
<td><strong>Result</strong></td>
<td>Either an intermediate result or final result.</td>
</tr>
<tr>
<td><strong>Support</strong></td>
<td>The support of a subgraph/pattern is equal to the fraction of graphs which involve the subgraph/pattern at least once [3].</td>
</tr>
<tr>
<td><strong>Subgraph</strong></td>
<td>A graph whose vertices and edges are subsets of the vertices and edges of another graph [8].</td>
</tr>
<tr>
<td><strong>Trace</strong></td>
<td>A trace in a business process model is a sequence of events generated during a process execution.</td>
</tr>
<tr>
<td><strong>Unregistered user</strong></td>
<td>A user who does not have an account on the APD tool.</td>
</tr>
<tr>
<td><strong>User</strong></td>
<td>A person who is currently using the APD tool or who has previously used the APD tool.</td>
</tr>
</tbody>
</table>
User activity

Creating a project, viewing a project, viewing or downloading a project’s files, deleting project files, sharing a project, starting or stopping an experiment phase, viewing results and status of an experiment phase, logging in, or logging out.

User activity log

A file containing information on the past user activities on the APD tool.

User tracking

The act of tracking the behavior of the user on the APD tool in the form of the user activity log.

1.3.2 List of abbreviations

APD
Anomalous pattern discovery

GDPR
General Data Protection Regulation

G file
Graph file

PNG
Portable Network Graphics

SRD
Software requirements document

TU/e
Eindhoven University of Technology

URD
User requirements document

1.4 List of references


1.5 Overview

The remainder of this document consists of three sections. In Section 2, a general description of the Delta extension of the APD tool. Sections 2.1 and 2.2 discuss the project's relation to current, predecessor, and successor projects. Section 2.3 describes the general function and purpose that the Delta extension of the APD tool fulfills. The environment in which the tool operates is described in Section 2.4 and Section 2.5 discusses the relation of the APD tool to other systems. Section 2.6 describes the general constrains that the Delta team must comply with. Lastly, Section 2.7 provides a description of the logical model of the Delta extension of the APD tool.

Afterwards, in Section 3 of this document, a list of the functional requirements the Delta team must comply with, is provided. The non-functional requirements can be found in the URD [1]. Section 4 provides a traceability matrix that maps the user requirements specified in Section 3 of the URD[1] to the software requirements specified in Section 3 of this SRD.
2 General Description

2.1 Relation to current projects

Delta is an extension of the APD tool, a tool for process mining. Other free tools for process mining could therefore be considered as competitors. Noteworthy examples of this include Disco [9] and ProM [10].

With Disco, a user can examine their event logs by converting them into process models. These process models show a variety of statistics, such as the percentage of traces that follow a certain path. The goal of Disco, examining an event log, is very different from that of the APD tool, comparing an event log and process model.

ProM is an entirely different approach to process mining, offering a platform with a large variety of plug-ins. On ProM a user can select a process model, event log, or both and then run any plug-ins which are available for the files they selected. Users can also contribute to the ProM platform by creating their own plug-ins. Technically it might be possible for the functionality of the APD tool to be offered through such a plug-in, but there are reasons not to do this. Creating a plug-in on ProM means users would be expected to run it on their own computer. This could be a problem since the subgraph discovery of the APD tool can take days due to the size of event logs it handles. The APD tool runs on a server, however, thus allowing users to simply start an experiment and return when it is completed. Users would also need to share their files and results to cooperate on a project with other users, instead of the sharing functionality added by Delta.

2.2 Relation to predecessor and successor projects

The APD tool is a predecessor project of the Delta extension. The APD tool was built by researchers of the Università Politecnica delle Marche and the Eindhoven University of Technology. Delta will extend the APD tool by using it as a black box, which means the APD functionality of the tool will not be changed. Instead, an interface will be built around it, adding the functionality described in this document, and calling on the tool when its functionality is required.

No concrete plans for successor projects exist yet. However, it is possible that, after the extension has been delivered, the original team of the APD tool will continue to extend it. The customer intends to use the tool more widely with the Delta extension and this could result in further changes or functionality being required. Applying these changes would be a successor project of Delta. It is possible that plugins will be attached to the APD tool in the future.

2.3 Function and purpose

The APD tool is used for conformance checking, specifically by discovering anomalous subgraphs first and then deriving anomalous patterns from those subgraphs [3]. Delta will leave this functionality intact while extending the tool with useful features to improve usability and user management. Among other things, the Delta team will add a user management system, improved management of projects and experiments, and, possibly, the customization of the subgraph discovery algorithm.

Once these features are added, users of the tool will be able to register for an account on the APD tool. Creating a project will no longer be available to all users of the tool and the project can no longer be accessed by all other users of the tool. Instead, a user can choose to share his projects with other registered users of their choosing. They can specify the permissions that other users have on the project. Sharing a project helps users cooperate with others on a single project while still controlling other user’s access rights to their project.

Users will also have much more insight into the status of their experiments. The new user interface will show information about running experiment phases. For example, the user will be able to view whether any error has occurred while executing the experiment phase. Users can also see the running status of their running experiment phases. Once an experiment has completed, results can
be viewed on a single page. This simplifies the process of comparing results and analyzing the user’s business process.

Besides functions for registered and unregistered users, the administrators of the tool will also be given more control. Administrators will be able to add or remove users, view and edit all projects that were created by registered users and access user tracking data of specific registered users of the tool. More details on the user roles are given in Section 2.4.

The optional change of customizing the subgraph discovery algorithm would allow administrators to upload a different implementation for the anomalous subgraph discovery phase than SUBDUE [11]. Any registered user will then be able to choose to use the uploaded algorithm in the anomalous subgraph discovery phase, instead of SUBDUE. The main advantage of this is that, because the SUBDUE algorithm does not work well on every input, having multiple algorithm options allows the registered users of the tool to analyze more types of input.

All these new functions serve the purpose of this project: to simplify the process of working with the APD tool and to make it a more attractive option to use for conformance checking.

2.4 Environment

Users can make use of the web interface to access the tool. The web interface can be accessed using Chrome version 66 and is stored on the server. The interface is designed to also run on mobile devices.

The following types of users of the tool can be distinguished:

- **Academics**
  The most important users of the tool are academics. Academics will use the tool to perform conformance checking between traces in an event log and a process model. These academics are expected to be active in the field of anomalous traces discovery, either as researchers or as students. This prior knowledge ensures that the tool does not need to explain all the details of what each experiment phase involves, instead relying on the users to have experience in the required fields.
  Several types of academic users can be identified:
    - **Unregistered users**
      A user who does not have an account on the APD tool. These academics will be users who access the tool for the first time and are curious to get a taste of what it has to offer. To help these users understand what the tool can be used for, they will be able to view a demo project together with its results. This will allow them to see what kind of results they could generate using their own data, without having to register first.
    - **Registered users**
      A user who has an account on the APD tool. These users can create new projects, run experiments in those projects, share their projects with other users, and view results. Registered users can also change their account details, such as their password or email address.
  
- **Administrator**
  An administrator is a special user who is responsible for maintaining the tool with regards to user and project management. They can manually add or remove user accounts and they have access to all projects on the tool. Administrators can also access the user activity log, which keeps track of all user activities on the tool. An administrator can enable and disable tracking for specific users or delete the log entries of a specific user from the user activity log. A custom algorithm used in the anomalous subgraph discovery phase can be uploaded by an administrator. This algorithm can then be used by registered users running the relevant experiment phase.

Further details of these user types and their subroles are specified in Section 2.4 of the URD [1].
2.5 Relation to other systems

The Delta extension of the APD tool is not related to any other systems, either external or internal, except for the APD tool itself. The Delta extension will add functionality to the existing APD tool and will become a replacement for the original tool. Therefore, the Delta extension relies heavily on the functionality of the existing APD tool, in particular the execution of the anomalous subgraph discovery and partial order discovery processes. The results of the execution of these processes will be used directly by the Delta extension in the visualization for the user of the system.

2.6 General constraints

Researchers with an academic background are the main target audience of the tool. Academics and students are expected to have knowledge about conformance checking between traces in an event log and a process model. This means it is not necessary to design the user interface in such a way that it can be fully utilized without any prior knowledge in this field. This does not mean usability is of no concern. Instead it means the focus needs to be on improving the experience and possibilities for more advanced users. This will be achieved by creating an interface which allows for an easier monitoring of running experiments and more in-depth analysis of results.

2.6.1 Environment

Delta requires the APD tool to be usable from a desktop or laptop computer running the Google Chrome version 66 browser. Secondary browsers which the tool might function on are Firefox version 59, Internet Explorer version 11, and Safari version 8, but these do not fall within the constraints of the project. The current version of the tool is available in English and the extension will continue to use English. To deal with a wide range of device types the tool should be responsive.

2.6.2 Privacy

One of the features added to the tool by the Delta team requires the storage of user information and logging of user activity. The GDPR [12] is a new European law surrounding the collection, storage, and processing of personal data. Since email addresses are considered personal data and Delta requires email addresses to function properly, it needs to be compliant with GDPR. Not complying with the rules laid out by the GDPR could result in high fines depending on the articles infringed, therefore the extension needs to be designed with GDPR in mind.

2.6.3 Performance

The new user access system should support a minimum of 200 registered users, as the target audience is not expected to exceed this number. To ease access to the tool, a request for registering a new account should be processed within 1 minute and login request should be validated within 10 seconds.

2.7 Model description

In this section we describe the class diagram, data model, and sequence diagrams to give an overview of the system and its intended functionality from a logical viewpoint.

2.7.1 Class diagram

This section describes the logical model of the Delta extension of the APD tool. This model is depicted in the class diagram in Figures 1 and 2. All entities, together with their attributes and operations, are presented in the model and described below. The relationships between these entities are also described. Note that the class diagram is spread across two pages. The two pages...
are connected via the relationship drawn between the Project entity in Figure 1, and the Experiment entity in Figure 2.

UnregisteredUser
The UnregisteredUser class represents a guest user of the tool. Guest users only have permission to view the DemoProject and they can register for an account.

- `register(username: string, email: string, password: string, homeCountry: string, homeAddress: string, firstName: string, lastName: string, organization: string, organizationRole: string, heardAbout: string): void`
  Adds a new user with the given username, email address, and password, to the database. The provided username needs to be unique, the email address needs to be valid and unique, and the password needs to contain at least one number, one capital letter, one lower case letter and one special character. When a new user has registered, the administrators are notified.

RegisteredUser
The RegisteredUser class represents a registered user of the tool. The RegisteredUser is associated with several UserLogEntries. Every RegisteredUser has a unique username, a unique and valid email address, and a password for authentication purposes. The RegisteredUser class also has several optional attributes, which are optional fields in the registration process of the user. The remaining attributes of a RegisteredUser concern the user’s consent with regard to the privacy notice, the tracking of the user’s activities and the resetting of the user’s password. A RegisteredUser can log in and out, they can access and change their account information and change or reset their password. A RegisteredUser can also create Projects and search for projects by name or date.

- `id: integer`
  The unique identifier of the user.

- `username: string`
  The username of the user.

- `email: string`
  The email address of the user.

- `password: string`
  The hashed password of the user.

- `homeCountry: string`
  The home country of the user.

- `homeAddress: string`
  The home address of the user.

- `firstName: string`
  The first name of the user.

- `lastName: string`
  The last name of the user.

- `organization: string`
  The organization of the user.

- `organizationRole: string`
  The organization role of the user.

- `heardAbout: string`
  The way the user heard about the tool.

- `trackingEnabled: boolean`
  Boolean that is true if the tracking of this user’s data is enabled and false otherwise.

- `consent: boolean`
  True if the user has given consent with regard to the privacy notice and false otherwise.
Figure 1: Class diagram of the Delta extension, part 1
Figure 2: Class diagram of the Delta extension, part 2
• `login(password : string) : void`
  Checks the combination of the given password and the username of the user and sets the user to logged in if the combination was correct.

• `logout() : void`
  Logs out the registered user.

• `resetPassword() : void`
  Resets the password of the registered user by sending an email to the registered user.

• `changePassword(newPassword : string) : void`
  Changes the password of the user to the provided new password.

• `changeAccountInfo(field : string, value : string) : void`
  Updates the provided field of the user’s account information in the database, by storing the provided value in the provided field.

• `requestTrackingData() : UserLogEntry[]`
  Requests all tracking data of the user from the user log. All user log entries of the user are returned.

• `requestAccountDeletion() : void`
  Requests the deletion of the user’s account from the database.

• `requestDeleteUserLog() : void`
  Requests the deletion of all the user’s tracking data from the database.

• `createProject(name : string, processModel : file, eventLog : file) : void`
  Creates a new project with the given name, process model and event log, of which the user is the project owner.

• `searchProject(name : string) : Project[]`
  Returns the list of projects where the provided name is a substring of the project name.

• `searchProjectByDate(startDate : date, endDate : date) : Project[]`
  Returns the list of projects where the creation date lies between the provided start and end date.

**ProjectObserver**

A **ProjectObserver** is a special kind of **RegisteredUser** that is an observer of one or more **Projects**. Being an observer grants the user the rights to view project files and results.

**ProjectMember**

A **ProjectMember** is a special kind of user that is a member of one or more **Projects**. Being a **ProjectMember** allows the user to start new **Experiments** within the project or delete files of the project. A member can also upload files for several **Components** in order to skip certain components of an **ExperimentPhase**. A **ProjectMember** is automatically also a **ProjectObserver** of the same project, thus granting him the same rights.

**ProjectOwner**

A **ProjectOwner** is a special kind of user that is an owner of one or more **Projects**. Being a **ProjectOwner** allows the user to add and remove other members and observers from the project.

• `addMember(user : RegisteredUser, project : Project) : void`
  Adds the provided user as a member of the provided project.

• `addObserver(user : RegisteredUser, project : Project) : void`
  Adds the provided user as an observer of the provided project.

• `removeMember(user : RegisteredUser, project : Project) : void`
  Removes the provided user as a member of the provided project.
• `removeObserver(user : RegisteredUser, project : Project) : void`
  Removes the provided user as an observer of the provided project.

**Administrator**
The Administrator is a special kind of `RegisteredUser` that has full access to all functionality of the tool. They can enable or disable user activity tracking of a `RegisteredUser`. An administrator is a member of every `Project` on the APD tool. Finally, administrators can create and delete user accounts and reset a password of a `RegisteredUser`.

• `createUser(username : string, email : string, password : string, homeCountry : string, homeAddress : string, firstName : string, lastName : string, organization : string, organizationRole : string, heardAbout : string) : void`
  Adds a new user with the given username, email address, and password, to the database. The provided username needs to be unique, the email address needs to be valid and unique, and the password needs to contain at least one number, one capital letter, one lower case letter and one special character.

• `deleteUser(user : RegisteredUser) : void`
  Deletes the user account of the provided user from the database.

• `resetPassword(user : RegisteredUser) : void`
  Resets the password of the given registered user.

• `startTracking(user : RegisteredUser) : void`
  Starts tracking the user activity of the provided user.

• `stopTracking(user : RegisteredUser) : void`
  Stops the tracking of the user activity of the provided user.

**UserLogEntry**
A `UserLogEntry` represents an action performed by a `RegisteredUser`. All `UserLogEntries` are part of the `UserActivityLog`.

  • `id : integer`
    The unique identifier of the user log entry.

  • `user : RegisteredUser`
    The registered user that performed the activity.

  • `timestamp : time`
    Time at which the logged action occurred.

  • `action : string`
    Action that was logged.

**UserActivityLog**
In the `UserActivityLog` the activity of all `RegisteredUsers` is stored using `UserLogEntries`. Administrators have access to the user activity log.

  • `accessUserLogData(user : RegisteredUser) : UserLogEntry[]`
    Returns all the user log entries of the provided registered user.

  • `deleteUserLog(user : RegisteredUser) : void`
    Deletes all the user log entries of the provided registered user.

**Project**
A `Project` contains a process model and an event log, which are used when `Experiments` of this project are run. Every `Project` is associated with exactly one `ProjectOwner`, who has the right to delete the project. A `Project` can have multiple `ProjectMembers` and `ProjectObservers`, who all have different access rights with regards to the project. In a `Project`, a batch of experiments can be created and run, and the comparative results of this batch of experiments can be retrieved by project members.
Within a project, experiments can be searched for by status or date by project observers. The files of a project can be downloaded by project observers and deleted by project members.

- **id**: integer
  The unique identifier of the project.

- **name**: string
  The name of the project.

- **processModel**: file
  The input business process model of the project.

- **eventLog**: file
  The input event log of the project.

- **creationDate**: timestamp
  The date and time on which the project was created.

- **downloadFiles()**: file[]
  Downloads the files of this project.

- **deleteFile(fileName)**: void
  Deletes the file with the given file name from the project.

- **createExperiment(numIterationsRange, covRange, beamRange, divRange, itemsetThresholdRange, orderingThresholdRange)**: void
  Creates and runs an experiment for every parameter combination defined by the provided parameter ranges.

- **compareResults(experiments)**: Table
  Calculates the comparative results of the provided experiments and returns the results in table format.

- **searchExperimentByDate(runningDate)**: Experiment[]
  Returns the list of experiments that are part of this project, whose running date is equal to the provided running date.

- **searchExperimentByStatus(status)**: Experiment[]
  Returns the list of experiments that are part of this project, whose status is equal to the provided status.

- **delete()**: void
  Deletes the complete project from the database.

**DemoProject**
The DemoProject is a special Project that can be accessed by guest users of the tool. These UnregisteredUsers will have the rights of a ProjectObserver, in order to get a taste of what the tool has to offer.

**Experiment**
An Experiment is associated with one or two experiment phases, but at most one of each type. These experiment phases together complete the running of the experiment. Every Experiment is associated with exactly one Project. It represents a specific experiment that is run, using the process model and event log of the associated Project. An Experiment is also associated with exactly one ExperimentLog. The files of an experiment can be downloaded by project observers and administrators, and the experiment can be deleted by project members and administrators.

- **id**: integer
  The unique identifier of the experiment.

- **name**: string
  The name of the experiment.
• status : Status
  The status of an experiment describes the current phase of the experiment and whether the
  experiment is currently running or idle, or ran into an error.

• creationDate : timestamp
  The creation date of the experiment.

• downloadFiles() : file[]
  Downloads the files of this experiment.

• delete() : void
  Deletes the experiment.

ExperimentLog
An ExperimentLog represents the log of an Experiment. It contains information on the Experiment, all
user activities regarding the Experiment and the low level activities of the corresponding experiment
phases. The experiment log contains information on:

• The date and time an experiment phase is started.
• The date and time an experiment phase is finished.
• The components of an experiment phase that have been started, together with the date and
time they were started.
• The components of an experiment phase that are finished, together with the date and time
they were finished.
• Any errors that occurred while running the experiment, together with the date and time they
occurred.

An experiment log consists of several ExperimentLogEntries, each of which represents a recorded
activity. The experiment log can be deleted by project members.

• id : integer
  The unique identifier of the experiment log.

• delete() : void
  Deletes this experiment log.

ExperimentLogEntry
An ExperimentLogEntry represents a specific entry of an ExperimentPhase. It consists of an action
and a timestamp.

• id : integer
  The unique identifier of the experiment log entry.

• timestamp : timestamp
  Date and time at which the logged action occurred.

• action : string
  Action that was logged.

AnomalousSubgraphDiscoveryPhase
Every AnomalousSubgraphDiscoveryPhase belongs to exactly one Experiment. It represents the phase
of an experiment that performs anomalous subgraph discovery on the process model and event log
of the Project the Experiment belongs to. This experiment phase is associated with several Compo-
nents, which together perform the anomalous subgraph discovery. A project member can choose
which components to run, and can upload intermediate results of components he does not wish to
run. Four parameters can be specified in order to run the anomalous subgraph discovery phase: num-
iterations, beam, cov and div. An anomalous subgraph discovery phase can be cancelled or deleted
by project members and administrators.
• \textbf{id} : integer
  The unique identifier of the anomalous subgraph discovery phase.

• \textbf{numIterations} : integer
  The maximum number of iterations the subgraph discovery algorithm goes through. The algo-
  rithm will continue until no new subgraphs are found or until this threshold is reached; since a
  new subgraph is discovered at each iteration, this threshold represents the maximum number
  of subgraphs that can be found.

• \textbf{beam} : integer
  At each iteration, the algorithm extends a set of candidate subgraphs inferred during previous
  iterations to determine the one(s) which provide the best compression for the graphs dataset.
  The beam determines the maximum number of candidates to consider for the expansion at
  each iteration. The algorithm considers the top candidates based on the description length
  of the subgraphs. Set to 4 by default.

• \textbf{cov} : boolean
  Cov stands for coverage. If it is set to true the algorithm calculates the coverage, which is the
  fraction of the graph that can be reconstructed from the set of subgraphs returned by the
  algorithm.

• \textbf{div} : boolean
  Div stands for diversity. This is a variable which represents the difference between two sub-
  graphs based on how many edges and nodes need to be changed to move from one subgraph
  to another. If set to true, the algorithm calculates the average diversity of the entire set of
  subgraphs in the result.

• \textbf{status} : Status
  Describes the current status of the anomalous subgraphy discovery phase.

• \textbf{creationDate} : timestamp
  The creation date of the experiment phase.

• \textbf{intermediateResults} : file[]
  An array of intermediate results, which replace the results of certain components that belong
  to the anomalous subgraph discovery phase.

• \textbf{uploadIntermediateResults(intermediateResults : file[])} : void
  Uploads the provided intermedate results that replace certain components in the execution
  of this anomalous subgraph discovery phase.

• \textbf{run}() : void
  Runs the anomalous subgraph discovery phase using the parameters of this anomalous sub-
  graph discovery phase, by running the appropriate components in order. Components for
  which an intermediate result was uploaded in the intermediateResults attribute will be skipped.

• \textbf{cancel}() : void
  Cancels running the anomalous subgraph discovery phase.

• \textbf{delete}() : void
  Deletes this experiment phase.

\textbf{PartialOrderDiscoveryPhase}
Every \textbf{PartialOrderDiscoveryPhase} belongs to exactly one \textit{Experiment}. It represents the phase of an
experiment that performs partial order discovery on the process model and event log of the \textit{Project}
the \textit{Experiment} belongs to. It uses as input the results of a \textbf{AnomalousSubgraphDiscoveryPhase}, as
well as two specified parameters. A \textbf{PartialOrderDiscoveryPhase} is associated with several \textit{Components}, which together perform the partial order discovery. A project member can choose which
components to run, and can upload intermediate results of components he does not wish to run. A
partial order discovery phase can be cancelled or deleted by project members.

• \textbf{id} : integer
  The unique identifier of the partial order discovery phase.
- **itemsetThreshold**: float
  The minimum percentage of traces in the event log in which a set of subgraphs need to appear together for them to be considered a potential pattern. The order in which these subgraphs appear is not relevant for this parameter.

- **orderingRelationThreshold**: float
  A pattern will only be shown as a result if all its ordering relations between two subgraphs in it occur in at least this percentage of the total set of traces.

- **status**: Status
  Describes the current status of the partial order discovery phase.

- **creationDate**: timestamp
  Creation date of the experiment phase.

- **intermediateResults**: file[]
  An array of intermediate results, which replaces the results of certain components that belong to the partial order discovery phase.

- **uploadIntermediateResults**(intermediateResults: file[]): void
  Uploads the provided intermediate results that replace certain components in the execution of this partial order discovery phase.

- **run()**: void
  Runs the partial order discovery phase using the parameters of the partial order discovery phase, by running the appropriate components in order. Components for which an intermediate result was uploaded in the intermediateResults attribute will be skipped.

- **cancel()**: void
  Cancels running the partial order discovery phase.

- **delete()**: void
  Deletes this experiment phase.

**Component**
A Component is associated with exactly one experiment phase. All Components have results, which are (intermediate) results of their associated experiment phase. These results often are input of other Components as well. Components can be run, cancelled and deleted by project members.

- **id**: integer
  The unique identifier of the component.

- **status**: Status
  Describes the current status of the component.

- **creationDate**: timestamp
  Creation date of the component.

- **run()**: file[]
  Runs the component with the possible input of previously run components and returns the result files.

- **cancel()**: void
  Cancels running the component.

- **delete()**: void
  Deletes this component.

**CustomASDComponent**
The CustomASDComponent represents a custom component that contains a subgraph discovery algorithm that was uploaded by an Administrator. The custom component can be used by RegisteredUsers in order to use an alternative subgraph discovery algorithm when running the anomalous subgraph discovery phase of an experiment. A custom anomalous subgraph discovery component can be run, cancelled and deleted by project members.
• **id**: integer
  The unique identifier of the custom subgraph discovery algorithm.

• **status**: Status
  Describes the current status of the custom component.

• **creationDate**: timestamp
  Creation date of the custom component.

• **algorithm**: file
  The uploaded file containing the custom algorithm.

• **run**(): file[]
  Runs the component with the possible input of previously run components and returns the result files.

• **cancel**(): void
  Cancels running the component.

• **delete**(): void
  Deletes this component.

**AnomalousSubgraphDiscoveryResult**

An **AnomalousSubgraphDiscoveryResult** represents the result of a run anomalous subgraph discovery phase. This result consists of a hierarchy of several subgraphs, each of which consists of several nodes. Subgraphs are initially displayed as an empty node, and can be expanded to display the full subgraph. The parent and child subgraph intersection of several subgraphs in the hierarchy can also be indicated. Next to this, every subgraph has a certain support value and a number of occurrences. This means the anomalous subgraph discovery result also has an average support value. One can filter the subgraphs by providing a support value and filter type. Subgraphs can be exported to a PNG or a .g file and specific subgraphs can be searched for by providing a (substring of a) subgraph name. Project members and administrators can delete anomalous subgraph discovery results.

• **id**: integer
  The unique identifier of the experiment phase result.

• **averageSupport**: float
  The average support value of the subgraphs this anomalous subgraph discovery result consists of.

• **getSupportValue**(subgraph: Subgraph): float
  Returns the support value of the provided subgraph.

• **getNumberOfOccurrences**(subgraph: Subgraph): integer
  Returns the number of occurrences of the provided subgraph.

• **expand**(subgraphs: Subgraph[]): Subgraph[]
  Returns the fully expanded version of all provided subgraphs.

• **filter**(supportValue: float, filterType: string): Subgraph[]
  Filters the subgraphs of the result using the provided support value and filter type and returns these subgraphs.

• **exportPNG**(subgraphs: Subgraph[]): file
  Export the provided subgraphs to a PNG file.

• **exportG**(subgraphs: Subgraph[]): file
  Export the provided subgraphs to a .g file.

• **getParents**(depth: integer, subgraphs: Subgraph[]): Subgraph[]
  Returns the intersection of parents of the provided subgraphs up to the provided depth level.

• **getChildren**(depth: integer, subgraphs: Subgraph[]): Subgraph[]
  Returns the intersection of children of the provided subgraphs up to the provided depth level.
PartialOrderDiscoveryResult
A PartialOrderDiscoveryResult is the result of a run partial order discovery phase. This result consists of several patterns that consist of subgraphs, in which each subgraph consists of several nodes. A partial order discovery result has an average support value and a number of patterns. All subgraphs are initially displayed as an empty node, which can be expanded to display the full subgraph.

- **id**: integer
  - The unique identifier of the experiment phase result.

- **numberOfPatterns**: integer
  - The number of patterns this partial order discovery result consists of.

- **averageSupport**: float
  - The average support value of this partial order discovery result.

- **expand(patterns: Pattern[]): Pattern[]**
  - Returns the fully expanded version of all the provided patterns.

- **delete(): void**
  - Deletes the experiment phase result.

2.7.2 Data model
This section describes our data model. The data model illustrates the organization of the data used by the Delta extension. This model is described in an Entity-Relationship diagram, which is depicted in Figure 3. The diagram uses the Crow’s Foot notation, with entities depicted as rectangles and relations depicted as lines. Entities can have three different types of attributes: primary keys, required attributes and optional attributes. These attribute types are indicated illustrated in the legend of Figure 3. In this section the entities and relations in the diagram will also be further described.

User
Each User represents an account of a registered user or administrator of the system. A User can be an owner, member or observer of a Project. A User has an id as primary key, and also has several optional attributes.

- **id**: integer
  - Integer that uniquely identifies the user.

- **username**: string
  - The unique username that identifies the user for login purposes.

- **email**: string
  - The email address of the user.

- **password**: string
  - The hashed password associated with this user, used for authentication.

- **isAdmin**: boolean
  - True if this user is an administrator and false otherwise.

- **trackingEnabled**: boolean
  - True if user activity tracking is enabled for this user and false otherwise.

- **consent**: boolean
  - True if the user has given consent with regard to the privacy notice and false otherwise.

- **homeCountry**: string
  - An optional attribute that describes the home country of the user.
Figure 3: Entity-Relationship diagram of our data model
- `homeAddress : string`
  An optional attribute that describes the home address of the user.

- `firstName : string`
  An optional attribute that describes the first name of the user.

- `lastName : string`
  An optional attribute that describes the last name of the user.

- `organization : string`
  An optional attribute that describes the organization of the user.

- `organizationRole : string`
  An optional attribute that describes the organization role of the user.

- `heardAbout : string`
  An optional attribute that describes the way the user heard about the tool.

**UserLogEntry**

A `UserLogEntry` represents an action performed by a `User` on the tool. The data stored in a `UserLogEntry` contains information on this action.

- `id : integer`
  Integer that uniquely identifies the user log entry.

- `userId : integer`
  The unique identifier of the user that performed the action.

- `action : string`
  The action performed by the user.

- `timestamp : timestamp`
  An attribute that specifies when the action was performed.

**Project**

A `Project` is specified by an eventLog, a processModel, a project name, and a creation date. A `Project` has exactly one project owner, and any number of members and observers.

- `id : integer`
  Integer that uniquely identifies the project.

- `name : string`
  The name of the project.

- `processModel : file`
  The process model associated with this project.

- `eventLog : file`
  The event log associated with this project.

- `creationDate : timestamp`
  This attribute specifies the creation date of the project.

- `ownerId : integer`
  The unique user identifier that specifies which user is the owner of the project.

**Experiment**

Every `Experiment` is associated with exactly one `Project`. Every `Experiment` also has exactly one creator. Note that this creator could be a different `User` than the owner of the `Project` this `Experiment` belongs to. It is also associated with `ExperimentLogEntries` and one or two experiment phases. The `Experiment` contains the creation date, name, and running status of the experiment it represents.

- `id : integer`
  Integer that uniquely identifies the experiment.
• **name**: string
  The name of the experiment.

• **status**: Status
  The running status of the experiment.

• **creationDate**: timestamp
  The creation date and time of the experiment.

• **projectId**: integer
  The unique project identification that specifies which project this experiment belongs to.

**ExperimentLogEntry**
An ExperimentLogEntry is associated with exactly one Experiment. The ExperimentLogEntry contains the action being performed in the experiment it is associated to and the timestamp of this action.

• **id**: integer
  Integer that uniquely identifies the experiment log entry.

• **experimentId**: integer
  The unique identifier of the experiment this log belongs to.

• **timestamp**: timestamp
  An attribute that specifies when the action was performed.

• **action**: string
  The action that was performed.

**ASDPhase**
Every ASDPhase is associated with exactly one Experiment. It is also associated with several components that together perform the running of this anomalous subgraph discovery phase. The ASDPhase can also be associated with a CustomASDComponent, which contains an alternative subgraph discovery algorithm. This custom component can be used in the running of this anomalous subgraph discovery phase. The ASDPhase contains the creation date and the running status of the anomalous subgraph discovery phase it represents.

• **id**: integer
  Integer that uniquely identifies the anomalous subgraph discovery phase.

• **status**: Status
  The running status of this anomalous subgraph discovery phase.

• **creationDate**: timestamp
  The creation date and time of this anomalous subgraph discovery phase.

• **intermediateResults**: file[]
  An array of intermediate results, which replace the results of certain components that belong to the anomalous subgraph discovery phase.

• **results**: file[]
  The final results of this anomalous subgraph discovery phase.

• **experimentId**: integer
  The unique identifier of the experiment this anomalous subgraph discovery phase is associated with.

**PODPhase**
Every PODPhase is associated with exactly one Experiment. It is also associated with several components that together perform the running of the partial order discovery phase. The PODPhase contains the creation date and the running status of the partial order discovery phase it represents.

• **id**: integer
  Integer that uniquely identifies the partial order discovery phase.
- **status**: `Status`  
  The running status of this partial order discovery phase.

- **creationDate**: `timestamp`  
  The creation date and time of this partial order discovery phase.

- **intermediateResults**: `file[]`  
  An array of intermediate results, which replaces the results of certain components that belong to the partial order discovery phase.

- **results**: `file[]`  
  The final results of this partial order discovery phase.

- **experimentId**: `integer`  
  The unique identifier of the experiment this partial order discovery phase is associated with.

### Component

Every **Component** is associated with exactly one experiment phase. It is responsible for running a part of the experiment phase it is associated with. The **Component** contains information on the creation date and running status of the component it represents.

- **id**: `integer`  
  Integer that uniquely identifies the component.

- **status**: `Status`  
  The running status of this component.

- **creationDate**: `timestamp`  
  The creation date and time of this component.

- **phaseld**: `integer`  
  The unique identifier of the experiment phase this component is associated with.

### CustomASDComponent

The **CustomASDComponent** is an alternative component, associated with exactly one anomalous subgraph discovery phase. It contains an alternative subgraph discovery algorithm that can be used in the running of an anomalous subgraph discovery phase.

- **id**: `integer`  
  Integer that uniquely identifies the custom anomalous subgraph discovery component.

- **status**: `Status`  
  The running status of this component.

- **creationDate**: `timestamp`  
  The creation date and time of this component.

- **algorithm**: `file`  
  The file that contains the custom subgraph discovery algorithm.

- **phaseld**: `integer`  
  The unique identifier of the experiment phase this component is associated with.
2.7.3 Sequence diagrams

User access

View demo - Figure 4  An unregistered user can access the tool without loggin in. They do have the option to continue using the tool as an unregistered user, which allows them to view a demo project. Unregistered users can only view this demo project. They do not have member rights to the demo project.

Goal: An unregistered user accesses a demo project.

Preconditions: There exists a demo project. The user is not logged in.

Summary: An unregistered user views the demo project made available to them by the tool.

Priority: Should have.

Registration - Figure 5 Users can register for an account. To do this, they have to navigate to the registration page and fill in all necessary information. Some fields in this registration form are optional and may be skipped. After the registration form is correctly filled in and submitted, the user is now registered and can log in to the tool.

Goal: An unregistered user creates an account for the APD tool.

Preconditions: The user is unregistered.

Postconditions: A new account is registered.

Summary: A new user creates an account for the APD tool.

Priority: Must have.

Logging in - Figure 6 A registered user can log in to the tool by navigating to the login page. The user has to fill in their credentials (username and password). The tool checks the validity of the
given credentials and the user is logged in if they are correct. If the given credentials do not match, the tool feeds this information back to the user and remains on the login page.

**Goal:** A registered user logs in to the APD tool.

**Preconditions:** The user is a registered user.

**Postconditions:** The user is logged in.

**Summary:** A registered user logs in to the APD tool.

**Priority:** Must have.

**Logging out - Figure 7** If a registered user is logged in, this user can log out. The user is logged out and is navigated back to the home page of the tool.

**Goal:** Registered user logs out from the APD tool.

**Preconditions:** The user is a registered user. The user is logged in.

**Postconditions:** The user is logged out.

**Summary:** A registered user logs out from the APD tool.

**Priority:** Must have.

**Access account information - Figure 8** A registered user can access their account information when logged in to the tool. The tool retrieves and displays their account information.

**Goal:** Registered user accesses their account information.

**Preconditions:** The registered user is logged in.

**Summary:** A registered user accesses his account information.

**Priority:** Must have.

**Edit account information - Figure 9** A registered user can edit their account information, if the user is logged in. The user navigates to the page that displays their account information and edits their account information. After the user confirms the changes, these are saved and the new account...
Figure 6: Logging in

Figure 7: Logging out
information is displayed to the user. If the user cancels the changes, the changes are not saved and the original account information is displayed.

**Goal:** Registered user changes their account information.

**Preconditions:** The user is a registered user. The user is logged in. The user has completed sequence 2.7.3.

**Postconditions:** The account information of the registered user is changed.

**Summary:** A registered user changes their account information.

**Priority:** Must have.
**Change password - Figure 10**  A registered user can change their password through a reset process.

**Goal:** Change the password of a registered user.

**Preconditions:** The user is a registered user. The user is logged in. The user has completed sequence 2.7.3.

**Postconditions:** The password of the user account is changed.

**Summary:** A registered user changes his password.

**Priority:** Should have.

---

![Diagram of Change password process](image_url)
Create account as administrator - Figure 11

Goal: Administrator creates a new account.

Preconditions: The user is administrator. The user is logged in.

Postconditions: A new account is registered.

Summary: An administrator creates a new account.

Priority: Should have.

Delete account as administrator - Figure 12

Administrators can delete registered accounts via the tool. A logged in administrator navigates to the registered accounts page on the tool and selects the account that is to be deleted. After the administrator confirms the deletion, the registered account is deleted from the database.

Goal: Delete a registered account.

Preconditions: The user is an administrator. The user is logged in. The to-be-deleted account is a registered user account.

Postconditions: The registered account is deleted from the database.

Summary: The administrator deletes a registered user account.

Priority: Must have.
Figure 12: Delete account as administrator
User tracking

Tracking user activity - Figure 13  User activity is tracked in a user activity log. This log receives entries for a specific user only when user tracking is enabled for the respective user. Every time this registered user performs an activity on the tool and this user is logged in, the action is added to the user activity log.

**Goal:** Keep track of what users do with the APD tool.

**Preconditions:** User tracking is enabled by the administrator. The user is a registered user. The registered user is logged in.

**Summary:** The user activity of a user is tracked.

**Priority:** Could have.

![Diagram of user activity](image)

Figure 13: Tracking user activity

Accessing tracking data - Figure 14  If logged in, an administrator can access the user activity data of a selected user. The administrator navigates to the user tracking environment and selects a specific user. The activity log of the respective user is retrieved and displayed.

**Goal:** Access the user tracking data of a registered user.

**Preconditions:** The user is an administrator. The user is logged in.

**Summary:** The administrator accesses the user tracking data of a registered user.

**Priority:** Could have.

Enabling tracking - Figure 15  In order for the tool to track a user's data, user tracking should be enabled for that particular user. An administrator can enable tracking via the tool, after logging in. The administrator navigates to the user tracking environment and enables activity tracking for a selected user.

**Goal:** Administrator enables the tracking of a registered user's activity.

**Preconditions:** The user is an administrator and is logged in. The registered user's activity tracking is disabled.

**Postconditions:** The activity of a registered user is tracked.

**Summary:** An administrator enables the tracking of a registered user's activity.

**Priority:** Could have.
Figure 14: Accessing tracking data

Figure 15: Enabling tracking
Disabling tracking - Figure 16  The activity tracking of a specific user can be disabled. If user activity tracking is disabled, no user activity log entries will be created for this particular user. An administrator can disable tracking by navigating to the user tracking environment, and disabling tracking for a selected user.

Goal: Disable the tracking of a registered user’s activity.

Preconditions: The user is an administrator and is logged in. The registered user’s activity tracking is enabled.

Postconditions: The activity of a registered user is not tracked.

Summary: The administrator disables the tracking of a registered user's activity.

Priority: Could have.

Project management

Creating a new project - Figure 17  A registered user who is logged in can create new projects. New projects need a project name and two input files in order to be created. After confirmation, the project is created.

Goal: Creating a new project.

Preconditions: The user is a registered user. The user is logged in.

Postconditions: A new project is created. The registered user is a project owner.

Summary: A registered user creates a new project.

Priority: Must have.

Access files of a project - Figure 18  A logged in, registered user who is an observer of a project can access the project's files. The user selects the particular project and next selects the project file he wishes to access.

Goal: Access files of a project.

Preconditions: The user is a registered user and is logged in. The user is at the list of experiments of a project. The user is project observer.

Summary: A registered user accesses files of a project they are an observer of.

Priority: Could have.
Figure 17: Creating a new project

Figure 18: Access files of a project
Delete files of a project - Figure 19  A registered user can delete files of a project they are a member of. The project has to be selected by the user, after which the files can be deleted.

Goal: Delete files of a project.

Preconditions: The user is a registered user and is logged in. The user is at the list of experiments of a project. The user is a project member.

Postconditions: The files are removed from the project.

Summary: A registered user deletes files of a project associated to him.

Priority: Could have.

Share a project - Figure 20  A registered user who is a project's owner can share that project with other registered users. The registered user has to be logged in. The user navigates to the project and selects the setting to share the project. The other registered user can be made a project member or a project observer. After confirming the sharing of the project, the project rights are updated.
**Goal:** Share a project with another registered user.

**Preconditions:** The caller of the action is a registered user and is the project’s owner. The user is logged in. The user is at the list of projects.

**Postconditions:** The project is shared with the other registered user. The registered user is logged in.

**Summary:** A registered user shares a project with another registered user.

**Priority:** Should have.

![Diagram of project sharing process](image)

**Figure 20: Share a project**

**Delete a project - Figure 21** A registered user who is a project’s owner can delete that project. The user simply deletes the selected project. After confirming the deletion, the project is deleted from the database and the deletion is confirmed.

**Goal:** Delete an entire project.

**Preconditions:** The user is a registered user and is logged in. The user is at the list of projects. The user is project owner.

**Postconditions:** The project is deleted.

**Summary:** A project owner deletes their project.

**Priority:** Should have.

**Change project rights - Figure 22** A logged in project owner who has shared a project with other users can change the access rights of those users. The project owner navigates to the access rights of the selected project and changes the access rights of the other users. Access rights of a user can be changed from project member to project observer and vice versa.
Figure 21: Delete a project
**Goal:** Change the project rights of a registered user.

**Preconditions:** The caller of the action is a registered user and is logged in. The user is at the list of projects. This user is the project owner. The registered user is either an observer or a member of the project.

**Postconditions:** The project rights of a registered user are changed.

**Summary:** The project owners changes the project rights for a registered user.

**Priority:** Should have.

![Diagram](image)

**Figure 22: Change project rights**

**Remove project rights - Figure 23** Access rights of a user to a project can be removed by the project owner, if the project owner is logged in. The project owner selects the users and removes their access rights to the project.

**Goal:** Remove the project rights of a registered user.

**Preconditions:** The caller of the action is a registered user and is logged in. The user is at the list of projects. This user is project owner. The registered user is either an observer or a member of the project.

**Postconditions:** The registered user has no project rights.

**Summary:** The project owner removes the project rights of a registered user.

**Priority:** Should have.

![Diagram](image)

**Figure 23: Remove project rights**

**Experiment management**
Run anomalous subgraph detection - Figure 24  A registered user can run anomalous subgraph detection on a project they are a member of, if they are logged in. The registered user runs an experiment and selects the anomalous subgraph detection phase. The anomalous subgraph detection is started and the user is notified.

**Goal:** Start the anomalous subgraph detection.

**Preconditions:** The user is a registered user and is logged in. The user is at the list of experiments of a project. A project has been created and the user is project member.

**Postconditions:** The phase is running.

**Summary:** The user starts anomalous subgraph discovery.

**Priority:** Must have.

Run anomalous subgraph detection with custom parameters - Figure 25  A registered user can run anomalous subgraph detection with custom parameters on a project they are a member of, if they are logged in. The registered user runs an experiment and selects the anomalous subgraph detection phase. The user sets the custom parameters and after confirmation, the anomalous subgraph detection is started using the custom parameters. The user is notified.

**Goal:** Run the anomalous subgraph detection with custom parameters.

**Preconditions:** The user is a registered user and is logged in. The user is at the list of experiments of a project. The user is project member.

**Postconditions:** The parameters for anomalous subgraph detection are set and the experiment phase is running.

**Summary:** The user sets the parameters for anomalous subgraph detection and starts the experiment phase.

**Priority:** Could have.

Run pattern detection - Figure 26  After the anomalous subgraph discovery phase is completed, a registered user can run the anomalous pattern discovery phase on a project they are a member
Figure 25: Run anomalous subgraph detection with custom parameters
of, if they are logged in. The registered user runs an experiment and selects the pattern detection phase. After confirmation, the pattern detection is started and the user is notified.

**Goal:** Run the pattern detection experiment phase.

**Preconditions:** The user is a registered user and is logged in. The user is project member. The user is at the list of experiments of a project.

**Postconditions:** The pattern detection experiment phase is started.

**Summary:** The user starts pattern detection.

**Priority:** Must have.

![Diagram of run pattern detection](image)

**Stop a running experiment phase - Figure 27** After an experiment phase has started, a registered user that is a member of the project that the experiment is part of, can stop the experiment phase while it is running, if the user is logged in. The user navigates to their running experiments and stops the selected experiment phase. After confirmation, the experiment phase is stopped and the user is notified.

**Goal:** Stop an experiment phase while it is running.
**Preconditions:** The user is a registered user and is logged in. The user is project member. The user is at the list of experiments of a project. An experiment phase is running on this project.

**Postconditions:** The experiment phase has stopped running.

**Summary:** A project member stops a running experiment phase.

**Priority:** Could have.

Figure 27: Stop a running experiment phase

**Accessing experiment information - Figure 28** A registered user can access experiment information in projects they are a member of. The user needs to be logged in. The user selects the specific experiment and the experiment information is displayed.

**Goal:** Access experiment information.

**Preconditions:** The user is a registered user and is logged in. The user is project observer. The user is at the list of experiments of a project. An experiment of this project is in progress or finished.

**Summary:** A registered user looks up information about their in progress experiments and previous experiments.

**Priority:** Must have.
Access experiment results as registered user - Figure 29  Experiment results can be viewed after an experiment phase has finished. A registered user can access the experiment results in projects they are a member of. The user needs to be logged in. Registered users can access these results when selecting the experiment in the project.

Goal: Access result of an experiment.

Preconditions: The user is a registered user and is logged in. The user is a project observer. The user is at the list of experiments of a project.

Postconditions: Experiment results are displayed.

Summary: A registered user displays results of an experiment.

Priority: Must have.

Delete experiment log file - Figure 30  Registered users can delete an experiment log after an experiment phase has finished, if they are logged in and they are a member of the project the experiment is a part of. The user navigates to the specific experiment and deletes the selected experiment log. After confirmation, the experiment log file is deleted from the database and the user is notified.

Goal: Delete the log file of an experiment.

Preconditions: The user is a registered user and is logged in. The user is project member. The user is at the list of experiments of a project. An experiment phase of this project is completed. No phase of this experiment is running.

Postconditions: The log file of the experiment is deleted.

Summary: The user deletes the log file of an experiment.

Priority: Should have.
Figure 30: Delete experiment log file
**Start multiple runs of an experiment - Figure 31**  
Registered users can start a batch of experiments in projects they are a member of. The user runs a batch of experiments and sets ranges for the experiment parameters. After confirmation, an experiment is created and started for each parameter combination in the specified ranges. The user is notified.

**Goal:** Start multiple runs of an experiment.

**Preconditions:** The user is a registered user and is logged in. The user is a project member of the project. The user is at the list of experiments of the project.

**Postconditions:** The experiment runs are performed.

**Summary:** The user starts multiple executions of an experiment.

**Priority:** Could have.
Comparing results of multiple runs of an experiment - Figure 32  After multiple experiment runs in a project have finished, a project observer can access the comparative results of the experiments. The user navigates to the project and selects the experiments he wishes to compare. The experiment results are retrieved and the comparative results are displayed.

Goal: Display comparative results between experiments.

Preconditions: The user is a registered user and is logged in. The user is a project observer. The user is at the list of experiments in a project. An experiment in this project has been run multiple times.

Postconditions: Comparative results of the experiment are displayed.

Summary: A registered user displays comparative results between runs of an experiment.

Priority: Could have.

Upload implementation of subgraph discovery algorithm - Figure 33  Administrators are allowed to upload custom implementations for the anomalous subgraph detection algorithm. All implementations uploaded by an administrator can be used by every registered user.

Goal: Upload implementation of the subgraph mining algorithm.

Preconditions: The user is an administrator and is logged in.

Postconditions: The implementation is uploaded to the server.

Summary: The administrator uploads a custom implementation of the subgraph mining algorithm which can be run on the input files.

Priority: Could have.
Start subgraph discovery with custom algorithm implementation - Figure 34  When creating an experiment that runs the anomalous subgraph detection phase, the user can select which implementation for the subgraph discovery algorithm should be used, after logging in. All implementations uploaded by an administrator can be selected. The user starts a new experiment and selects a custom algorithm for the anomalous subgraph detection phase. After confirmation, the anomalous subgraph detection phase is started, using the custom algorithm. The user is notified.

Goal: Start executing subgraph discovery with a custom implementation of the algorithm.

 Preconditions: The user is a registered user and is logged in. The user is project member and a custom implementation of the subgraph discovery algorithm has been uploaded by an administrator. The user is at the experiment list of a project.

Postconditions: The subgraph discovery phase is running, using the custom implementation.

Summary: The user starts subgraph discovery using a custom implementation of the algorithm.

Priority: Could have.

Figure 34: Start subgraph discovery with custom algorithm implementation

Component customization
Upload intermediate result - Figure 35  A logged in, registered user can upload their own results of a custom component of a project they are a member of. These results can then be used by the tool to start the next custom component in the experiment. The user navigates to the project and uploads the intermediate results of the custom component. After confirmation, the uploaded file is validated and saved if the result file is valid.

**Goal:** Upload an intermediate result of a component of an experiment.

**Preconditions:** The user is a registered user and is logged in. The user is project member. The user is at the list of experiments. The user has a file representing an intermediate result of a component.

**Postconditions:** A file is uploaded and can be used for the project.

**Summary:** A user uploads a file.

**Priority:** Could have.
Select components for the experiment setup - Figure 36  A logged in, registered user can run a select set of components, using previously uploaded intermediate results, when starting a new experiment in a project they are a member of. The user starts a new experiment and selects the custom components they wish to run. After confirmation, the experiment is started and the user is notified.

**Goal:** Select the components that the user wants to be executed.

**Preconditions:** The user is a registered user and is logged in. The user is project member. The user is at the list of experiments of a project. The user has uploaded intermediate result files of relevant components of the experiment.

**Postconditions:** The components that the user want to be run are selected.

**Summary:** The user selects the components that he wants to be run.

**Priority:** Could have.
Visualization

Search projects by name - Figure 37  A registered user who is logged in can search for projects by project name. The user enters the project name in the relevant search bar on their personalized home screen. If there exist projects with the given project name, to which the user has access rights, these projects are displayed. If such a project does not exist, the user is informed of this.

Goal: Search a project by name.

Preconditions: The user is a registered user and is logged in. The user is at the list of projects.

Summary: A registered user searches a project by entering its name.

Priority: Should have.

Search projects by creation date - Figure 38  A registered user who is logged in can search for projects by creation date. The user enters the project creation date in the relevant search bar on their personalized home screen. If there exist projects with the given project creation date, to which the user has access rights, these projects are displayed. If such a project does not exist, the user is informed of this.

Goal: Search a project by date.

Preconditions: The user is a registered user and is logged in.

Summary: A registered user searches a project by entering its creation date.

Priority: Should have.

Search experiments by running date - Figure 39  A registered user who is logged in can search for experiments by running date. The user enters the experiment running date in the relevant search bar on their personalized home screen. If there exist experiments with the given experiment running date, to which the user has access rights, these experiments are displayed. If such an experiment does not exist, the user is informed of this.

Goal: Search an experiment by date.

Preconditions: The user is a registered user and is logged in. The user is at the experiment list of a project.
**Summary:** A registered user searches an experiment by entering its running date.

**Priority:** Should have.

![Figure 39: Search experiments by running date](image)

**Search experiments by state - Figure 40** A registered user who is logged in can search for experiments by status. The user enters the experiment status in the relevant search bar on their personalized home screen. If there exist experiments with the given experiment status, to which the user has access rights, these experiments are displayed. If such an experiment does not exist, the user is informed of this.

**Goal:** Search an experiment by state.

**Preconditions:** The user is a registered user and is logged in. The user is at the experiment list of a project.

**Summary:** A registered user searches an experiment by entering its state.

**Priority:** Should have.

![Figure 40: Search experiments by state](image)

**Expand a node - Figure 41** When viewing experiment results, a registered user can select a node to expand it. The node information is retrieved, and the full corresponding graph of the node is displayed.

**Goal:** Expand a node in the subgraph hierarchy to show the corresponding graph.

**Preconditions:** The user is a registered user and is logged in. Graph results of an experiment are displayed.

**Postconditions:** The subgraph of the selected node is shown.

**Summary:** A registered user selects a node for which they want to see the corresponding subgraph.

**Priority:** Must have.

**Filter subgraphs - Figure 42** When viewing experiment results, a registered user can filter the displayed subgraphs by providing a certain support value. The user enters the support value and selects either minimal or maximal filtering. Based on the settings entered by the user, the result subgraphs are filtered and the correct subgraphs are displayed.

**Goal:** To filter results to display subgraphs with a certain support value.
Preconditions: The user is a registered user and is logged in. Graph results of an experiment are displayed.

Postconditions: Subgraphs with corresponding support value are displayed.

Summary: The registered user filters on graph data with a certain support value to only display subgraphs with at least that support value.

Priority: Could have.
Export subgraphs - Figure 43  When viewing experiment results, a registered user can select a subgraph to export. The user exports the selected subgraph, after which the subgraph is converted to PNG and returned to the user.

Goal: Export a subgraph to a PNG or .g file.

Preconditions: The user is a registered user and is logged in. Graph results are of an experiment are displayed.

Postconditions: A subgraph is exported.

Summary: A registered user selects a subgraphs they want to export.

Priority: Could have.

Show subgraph support value - Figure 44  When viewing experiment results, the user can select a subgraph to view its support value.

Goal: Display the support value of a subgraph.

Preconditions: The user is project observer and is logged in. Subgraph discovery results of an experiment are displayed.

Summary: The user chooses to display the support value of a subgraph.

Priority: Could have.

Show the number of occurrences of a subgraph - Figure 45  When viewing experiment results, a registered user can select a subgraph to view its number of occurrences.

Goal: Display the number of occurrences of a subgraph.

Preconditions: The user is project observer and is logged in. Subgraph discovery results of an experiment in the project are displayed.

Summary: The user chooses to display the number of occurrences of a subgraph.

Priority: Could have.

Indicating children and parents - Figure 46  When viewing experiment results, a registered user can select a subgraph to view its children and parents. After selection, the direct children and parents of that subgraph are retrieved and indicated to the user.

Goal: Indicate the direct children and parents.
Select a subgraph
Display selection
Get support value

Get support value
Support value

Display support value

Figure 44: Show subgraph support value

Select a subgraph
Display selection
Get number of occurrences

Get number of occurrences
Number of occurrences

Display number of occurrences

Figure 45: Show the number of occurrences of a subgraph
**Preconditions:** The user is project observer and is logged in. Subgraph discovery results of an experiment of this project are displayed.

**Summary:** The user selects a subgraph. The direct parents and children are indicated.

**Priority:** Must have.

![Diagram](image)

**Showing parents or children up to 5 levels deep - Figure 47** When viewing experiment results, the user can select a subgraph to view its children and parents up to 5 levels deep. After selection, the user can specify how many levels deep these children and parents should be indicated. Based on the settings entered by the user, the proper parents and children are retrieved and indicated to the user.

**Goal:** Display parents or children of a subgraph up to 5 levels deep.

**Preconditions:** The user is project observer and is logged in. Subgraph discovery results of an experiment of this project are displayed.

**Summary:** The user selects a subgraph. The parents or children up to 5 levels deep are displayed.

**Priority:** Should have.

![Diagram](image)
3 Specific requirements

The requirements have been prioritized using the MoSCoW model [13].

The MoSCoW model consists of the following rules:

**Must have** These requirements are fundamental and have to be implemented to the product. All must havess together constitute the minimum viable product.

**Should have** These requirement are of high importance for the product, but are not essential for the business objective.

**Could have** These requirements are not essential, but could be implemented if there is time left.

**Won't have** These requirements will not be implemented in the current delivery, but can be added to later version.

The requirements in this section are constructed and grouped using the classes defined in our class diagram described in figures 1 and 2. Every class has two types of specified requirements: attribute requirements and operation requirements. Attribute requirements are described using the following template:

```
SR-0
attributeName: type
Description of the attribute
```

First, the unique identifier of the requirement is given, together with the priority of the requirement specified on the right. Next, the name of the attribute and the attribute type are defined. Lastly, a description of the attribute is provided.

Operation requirements are described using the following template:

```
SR-0
operationName(): returnType
Description of the functional requirement
Input: variable : type
Precondition: Precondition of the operation requirement
Postcondition: Postcondition of the operation requirement
```

First, the unique identifier of the requirement is given, together with the priority of the requirement specified on the right. Next, the name of the operation and the return type of the operation are defined, followed by a description of the operation. The input variables of the operation are specified, together with their types, and the preconditions and postconditions of the operation requirement are specified.

3.1 Functional requirements
SR-1 register(): void

Adds a new user with the given username, email address, and password, to the database. The provided username needs to be unique, the email address needs to be valid and unique, and the password needs to contain at least one number, one capital letter, one lower case letter and one special character. When a new user has registered, the administrators are notified.

Input: username : string
    email : string
    password : string
    homeCountry : string
    homeAddress : string
    firstName : string
    lastName : string
    organization : string
    organizationRole : string
    heardAbout : string

Precondition: The user is unregistered. The provided username is unique. The provided email address is valid and unique. The provided password contains at least one number, one capital letter, one lower case letter and one special character.

Postcondition: A new account is registered.

RegisteredUser

This class has the same specific requirements as the unregistered user class. This is because a registered user is a special type of unregistered user with additional permissions, attributes and operations. The registered user class has the following additional specific requirements.

SR-2 id: integer

The unique identifier of the user.

SR-3 username : string

The username of the user.

SR-4 email : string

The email address of the user.

SR-5 password : string

The hashed password of the user.

SR-6 homeCountry : string

The home country of the user.

SR-7 homeAddress : string

The home address of the user.
SR-8  
**firstName**: string  
The first name of the user.

---

SR-9  
**lastName**: string  
The last name of the user.

---

SR-10  
**organization**: string  
The organization of the user.

---

SR-11  
**organizationRole**: string  
The organization role of the user.

---

SR-12  
**heardAbout**: string  
The way the user heard about the tool.

---

SR-13  
**trackingEnabled**: boolean  
Boolean that is true if the tracking of this user's data is enabled and false otherwise.

---

SR-14  
**consent**: boolean  
True if the user has given consent with regard to the privacy notice and false otherwise.

---

SR-15  
**login()**: void  
Checks the combination of the given password and the username of the user and sets the user to logged in if the combination was correct.

**Input**:  
**password**: string  
**Precondition**: The user is a registered user.

**Postcondition**: The user is logged in.

---

SR-16  
**logout()**: void  
Logs out the registered user.

**Input**: -  
**Precondition**: The user is logged in.

**Postcondition**: The user is logged out.

---

SR-17  
**resetPassword()**: void  
Resets the password of the registered user by sending an email to the registered user.

**Input**: -  
**Precondition**: The user is registered.

**Postcondition**: An email is send to the user's email address.
SR-18

changePassword(): void

Changes the password of the user to the provided new password.

Input: newPassword : string

Precondition: The user is logged in.

Postcondition: The password of the user account is changed to the provided new password.

SR-19

changeAccountInfo(): void

Updates the provided field of the user’s account information in the database, by storing the provided value in the provided field.

Input: field : string

value : string

Precondition: The user is logged in.

Postcondition: The account information of the registered user of the provided field is changed to the provided value.

SR-20

requestTrackingData(): UserLogEntry[]

Requests all tracking data of the user from the user log. All user log entries of the user are returned.

Input: -

Precondition: The registered user is logged in.

Postcondition: The user log entries are returned.

SR-21

requestAccountDeletion(): void

Requests the deletion of the user’s account from the database.

Input: -

Precondition: The user is logged in.

Postcondition: The user account is deleted from the database.

SR-22

requestDeleteUserLog(): void

Requests the deletion of all the user’s tracking data from the database.

Input: -

Precondition: The user is logged in and user’s tracking data exists in the database.

Postcondition: The user’s tracking data is deleted from the database.

SR-23

createProject(): void

Creates a new project with the given name, process model and event log, of which the user is the project owner.

Input: name : string

processModel : file

eventLog : file

Precondition: The user is logged in.

Postcondition: A new project is created and the registered user is the project owner.
SR-24

**searchProject(): Project[]**

Returns the list of projects where the provided name is a substring of the project name.

**Input:** name : string

**Precondition:** The user is logged in.

**Postcondition:** The projects of which the provided name is a substring of the project name are returned.

SR-25

**searchProjectByDate(): Project[]**

Returns the list of projects where the creation date lies between the provided start and end date.

**Input:** startDate : date
def
endDate : date
def

**Precondition:** The user is logged in.

**Postcondition:** The projects of which the creation date lies between the provided start and end date are returned.

ProjectObserver

This class has the same specific requirements as the registered user class. This is because a project observer is a special type of registered user with additional permissions.

ProjectMember

This class has the same specific requirements as the project observer class. This is because a project member is a special type of project observer with additional permissions.

ProjectOwner

This class has the same specific requirements as the project member class. This is because a project owner is a special type of project member with additional operations. The project owner class has the following additional specific requirements.

SR-26

**addMember(): void**

Adds the provided user as a member of the provided project.

**Input:** user : RegisteredUser

**project : Project**

**Precondition:** The provided registered user and project exist in the database. The project owner is logged in and is project owner of the provided project.

**Postcondition:** The provided user is project member of the provided project.

SR-27

**addObserver(): void**

Adds the provided user as an observer of the provided project.

**Input:** user : RegisteredUser

**project : Project**

**Precondition:** The provided registered user and project exist in the database. The project owner is logged in and is the project owner of the provided project.

**Postcondition:** The provided user is project member of the provided project.
SR-28
removeMember(): void
Should have
Removes the provided user as a member of the provided project.

Input: user: RegisteredUser
project: Project

Precondition: The provided registered user and project exist in the database. The project owner is logged in, and is project owner of the provided project. The provided user is a project member of the provided project.

Postcondition: The provided user has no access rights to the provided project.

SR-29
removeObserver(): void
Should have
Removes the provided user as an observer of the provided project.

Input: user: RegisteredUser
project: Project

Precondition: The provided registered user and project exist in the database. The user is logged in and is project owner of the provided project. The provided user is a project observer of the provided project.

Postcondition: The provided user has no access rights to the provided project.

Administrator
This class has the same specific requirements as the registered user class. This is because an administrator is a special type of registered user with additional permissions and operations. The administrator class has the following additional specific requirements.

SR-30
createUser(): void
Should have
Adds a new user with the given username, email address, and password, to the database. The provided username needs to be unique, the email address needs to be valid and unique, and the password needs to contain at least one number, one capital letter, one lower case letter and one special character.

Input: username: string
email: string
password: string
homeCountry: string
homeAddress: string
firstName: string
lastName: string
organization: string
organizationRole: string
heardAbout: string

Precondition: The administrator is logged in. The provided username is unique. The provided email address is valid and unique. The provided password contains at least one number, one capital letter, one lower case letter and one special character.

Postcondition: A new account is registered using the provided input variables.
SR-31
deleteUser(): void
Must have
Deletes the user account of the provided user from the database.

Input: user: RegisteredUser
Precondition: The administrator is logged in. The provided registered user exists in the database.
Postcondition: The provided registered user account is deleted from the database.

SR-32
resetPassword(): void
Must have
Resets the password of the given registered user.

Input: user: RegisteredUser
Precondition: The administrator is logged in. The provided registered user exists in the database.
Postcondition: The password of the provided registered user is reset.

SR-33
startTracking(): void
Could have
Starts tracking the user activity of the provided user.

Input: user: RegisteredUser
Precondition: The provided registered user exists in the database. The administrator is logged in and the registered user’s activity tracking is disabled.
Postcondition: The activity of a registered user is tracked.

SR-34
stopTracking(): void
Could have
Stops the tracking of the user activity of the provided user.

Input: user: RegisteredUser
Precondition: The provided registered user exists in the database. The administrator is logged in and the registered user’s activity tracking is enabled.
Postcondition: The activity of the registered user is not tracked.

UserLogEntry

SR-35
id: integer
Could have
The unique identifier of the user log entry.

SR-36
user: RegisteredUser
Could have
The registered user that performed the activity.

SR-37
timestamp: time
Could have
Time at which the logged action occurred.

SR-38
action: string
Could have
Action that was logged.
UserActivityLog

SR-39
accessUserLogData(): UserLogEntry[]

Could have

Returns all the user log entries of the provided registered user.

Input: user: RegisteredUser

Precondition: The provided user exists in the database. The user initiating this operation is an administrator, or the user initiating this operation is equal to the provided registered user. This user is logged in.

Postcondition: The user log entries of the provided user are returned.

SR-40
deleteUserLog(): void

Could have

Deletes all the user log entries of the provided registered user.

Input: user: RegisteredUser

Precondition: The provided user exists in the database. The user initiating this operation is an administrator and is logged in.

Postcondition: The user log entries of the provided user are deleted from the database.

Project

SR-41
id: integer

Must have

The unique identifier of the project.

SR-42
name: string

Must have

The name of the project.

SR-43
processModel: file

Must have

The input business process model of the project.

SR-44
eventLog: file

Must have

The input event log of the project.

SR-45
creationDate: timestamp

Must have

The date and time on which the project was created.

SR-46
downloadFiles(): file[]

Could have

Downloads the files of this project.

Input: -

Precondition: The user initiating this operation is project observer of the project. The user is logged in.

Postcondition: The project files are downloaded.
SR-47  
(deleteFile): void

Deletes the file with the given file name from the project.

Input: fileName : string

Precondition: The provided file exists in the project's files. The user initiating this operation is a project member of the project or is an administrator. The user is logged in.

Postcondition: The file is removed from the project.

SR-48  
(createExperiment): void

Creates and runs an experiment for every parameter combination defined by the provided parameter ranges.

Input: 
- numIterationsRange : Range
- covRange : Range
- beamRange : Range
- divRange : Range
- itemsetThresholdRange : Range
- orderingThresholdRange : Range

Precondition: The provided ranges are properly defined. The user initiating this operation is a project member of the project and is logged in.

Postcondition: A batch of experiments is created and started.

SR-49  
(compareResults): Table

Calculates the comparative results of the provided experiments and returns the results in table format.

Input: experiments : Experiment[]

Precondition: The provided experiments are part of the project.

Postcondition: Comparative results of the experiment are returned in table format.

SR-50  
(searchExperimentByDate): Experiment[]

Returns the list of experiments that are part of this project, whose running date is equal to the provided running date.

Input: runningDate : date

Precondition: The provided running date is properly defined. The user initiating this operation is a project observer of the project and is logged in.

Postcondition: The experiments that are part of this project, whose creation date matches with the provided running date are returned.

SR-51  
(searchExperimentByStatus): Experiment[]

Returns the list of experiments that are part of this project, whose status is equal to the provided status.

Input: status : Status

Precondition: The provided status is properly defined. The user initiating this operation is a project observer of the project and is logged in.

Postcondition: The experiments that are part of this project, whose status matches the provided state are displayed.
SR-52

`delete()`: void

Deletes the complete project from the database.

**Input:** -

**Precondition:** The user initiating this operation is a project owner of the project and is logged in.

**Postcondition:** The project is deleted.

---

**DemoProject**

This class has the same specific requirements as the project class. This is because a demo project is a special type of project with different permission settings.

---

**Experiment**

---

**SR-53**

`id`: integer

The unique identifier of the experiment.

**SR-54**

`name`: string

The name of the experiment.

**SR-55**

`status`: Status

The status of an experiment describes the current phase of the experiment and whether the experiment is currently running or idle, or ran into an error.

**SR-56**

`creationDate`: timestamp

The creation date of the experiment.

**SR-57**

`downloadFiles()`: file[]

Downloads the files of this experiment.

**Input:** -

**Precondition:** The user initiating this operation is a project observer of the project of this experiment. This user is logged in.

**Postcondition:** The files are downloaded.

**SR-58**

`delete()`: void

Deletes the experiment.

**Input:** -

**Precondition:** The user initiating this operation is a project owner of the project of this experiment or is an administrator. This user is logged in.

**Postcondition:** The experiment is deleted.

---

**ExperimentLog**
**SR-59**

`id`: integer

The unique identifier of the experiment log.

**SR-60**

`delete()`: void

Deletes this experiment log.

**Input:**

**Precondition:** At least one phase of the experiment this experiment log belongs to, has completed. The user initiating this operation is a project member of the project the experiment log is part of. This user is logged in.

**Postcondition:** The experiment log is deleted.

---

**ExperimentLogEntry**

**SR-61**

`id`: integer

The unique identifier of the experiment log entry.

**SR-62**

`timestamp`: timestamp

Date and time at which the logged action occurred.

**SR-63**

`action`: string

Action that was logged.

---

**AnomalousSubgraphDiscoveryPhase**

**SR-64**

`id`: integer

The unique identifier of the anomalous subgraph discovery phase.

**SR-65**

`numIterations`: integer

The maximum number of iterations the subgraph discovery algorithm goes through. The algorithm will continue until no new subgraphs are found or until this threshold is reached; since a new subgraph is discovered at each iteration, this threshold represents the maximum number of subgraphs that can be found.

**SR-66**

`beam`: integer

At each iteration, the algorithm extends a set of candidate subgraphs inferred during previous iterations to determine the one(s) which provide the best compression for the graphs dataset. The beam determines the maximum number of candidates to consider for the expansion at each iteration. The algorithm considers the top candidates based on the description length of the subgraphs. Set to 4 by default.
SR-67  
**cov**: boolean  
Could have

Cov stands for coverage. If it is set to true the algorithm calculates the coverage, which is the fraction of the graph that can be reconstructed from the set of subgraphs returned by the algorithm.

SR-68  
**div**: boolean  
Could have

Div stands for diversity. This is a variable which represents the difference between two subgraphs based on how many edges and nodes need to be changed to move from one subgraph to another. If set to true, the algorithm calculates the average diversity of the entire set of subgraphs in the result.

SR-69  
**status**: Status  
Must have

Describes the current status of the anomalous subgraph discovery phase.

SR-70  
**creationDate**: timestamp  
Could have

The creation date of the experiment phase.

SR-71  
**intermediateResults**: file[]  
Could have

An array of intermediate results, which replace the results of certain components that belong to the anomalous subgraph discovery phase.

SR-72  
**uploadIntermediateResults()**: void  
Could have

Uploads the provided intermediate results that replace certain components in the execution of this anomalous subgraph discovery phase.  
**Precondition**: The provided files all represent a result of a component of the anomalous subgraph discovery phase. The user initiating this operation is a project member of the project this anomalous subgraph discovery phase is part of. This user is logged in.  
**Postcondition**: The provided files are uploaded and can be used to replace certain components in the running of the anomalous subgraph discovery phase.

SR-73  
**run()**: void  
Must have

Runs the anomalous subgraph discovery phase using the parameters of this anomalous subgraph discovery phase, by running the appropriate components in order. Components for which an intermediate result was uploaded in the intermediateResults attribute will be skipped.  
**Precondition**: The user initiating this operation is a project member of the project this anomalous subgraph discovery phase is part of or is an administrator. This user is logged in.  
**Postcondition**: The phase is running.
SR-74

cancel(): void

Cancels running the anomalous subgraph discovery phase.

**Input:** -

**Precondition:** The anomalous subgraph discovery phase is running. The user initiating this operation is a project member of the project this anomalous subgraph discovery phase is part of. This user is logged in.

**Postcondition:** The anomalous subgraph discovery phase stopped running.

SR-75

delte(): void

Deletes this experiment phase.

**Input:** -

**Precondition:** The user initiating this operation is a project member of the project this anomalous subgraph discovery phase is part of or is an administrator. This user is logged in.

**Postcondition:** The experiment phase is deleted.

---

**PartialOrderDiscoveryPhase**

SR-76

**id:** integer

The unique identifier of the partial order discovery phase.

SR-77

**itemsetThreshold:** float

The minimum percentage of traces in the event log in which a set of subgraphs need to appear together for them to be considered a potential pattern. The order in which these subgraphs appear is not relevant for this parameter.

SR-78

**orderingRelationThreshold:** float

A pattern will only be shown as a result if all its ordering relations between two subgraphs in it occur in at least this percentage of the total set of traces.

SR-79

**status:** Status

Describes the current status of the partial order discovery phase.

SR-80

**creationDate:** timestamp

Creation date of the experiment phase.

SR-81

**intermediateResults:** file[]

An array of intermediate results, which replaces the results of certain components that belong to the partial order discovery phase.
**uploadIntermediateResults(): void**

Uploads the provided intermediate results that replace certain components in the execution of this partial order discovery phase.

**Input:** intermediateResults: file[]

**Precondition:** The provided files all represent a result of a component of the partial order discovery phase.

**Postcondition:** The provided files are uploaded and can be used to replace certain components in the running of the partial order discovery phase. The user initiating this operation is a project member of the project this partial order discovery phase is part of. This user is logged in.

---

**run(): void**

Runs the partial order discovery phase using the parameters of the partial order discovery phase, by running the appropriate components in order. Components for which an intermediate result was uploaded in the intermediateResults attribute will be skipped.

**Input:** -

**Precondition:** The user initiating this operation is a project member of the project this partial order discovery phase is part of or is an administrator. This user is logged in.

**Postcondition:** The partial order discovery experiment phase is started.

---

**cancel(): void**

Cancels running the partial order discovery phase.

**Input:** -

**Precondition:** The partial order discovery phase is running. The user initiating this operation is a project member of the project this partial order discovery phase is part of. This user is logged in.

**Postcondition:** The partial order discovery phase stopped running.

---

**delete(): void**

Deletes this experiment phase.

**Input:** -

**Precondition:** The user initiating this operation is a project member of the project this partial order discovery phase is part of or is an administrator. This user is logged in.

**Postcondition:** The partial order discovery experiment phase is deleted.

---

**Component**

**id: integer**

The unique identifier of the component.

**status: Status**

Describes the current status of the component.
SR-88
creationDate: timestamp
Creation date of the component.

SR-89
run(): file[]
Runs the component with the possible input of previously run components and returns the result files.

Input: -
Precondition: The results of components whose results are necessary for this component, are present. The user initiating this operation is a project member of the project this component is part of. This user is logged in.
Postcondition: The result files are returned.

SR-90
cancel(): void
Cancels running the component.

Input: -
Precondition: The component is running. The user initiating this operation is a project member of the project this component is part of. This user is logged in.
Postcondition: The component stopped running.

SR-91
delete(): void
Deletes this component.

Input: -
Precondition: The user initiating this operation is a project member of the project this component is part of or is an administrator. This user is logged in.
Postcondition: The component is deleted.

CustomASDComponent

SR-92
id: integer
The unique identifier of the custom subgraph discovery algorithm.

SR-93
status: Status
Describes the current status of the custom component.

SR-94
creationDate: timestamp
Creation date of the custom component.

SR-95
algorithm: file
The uploaded file containing the custom algorithm.
Could have
run(): file[]

Runs the component with the possible input of previously run components and returns the result files.

Input:
Precondition: The results of components whose results are necessary for this component, are present. The user initiating this operation is a project member of the project this component is part of. This user is logged in.
Postcondition: The result files are returned.

Could have
cancel(): void

Cancels running the component.

Input:
Precondition: The component is running. The user initiating this operation is a project member of the project this component is part of. This user is logged in.
Postcondition: The component stopped running.

Could have
delete(): void

Deletes this component.

Input:
Precondition: The component is idle. The user initiating this operation is a project member of the project this component is part of or is an administrator. This user is logged in.
Postcondition: The component is deleted.

AnomalousSubgraphDiscoveryResult

Must have
id: integer

The unique identifier of the experiment phase result.

Could have
averageSupport: float

The average support value of the subgraphs this anomalous subgraph discovery result consists of.

Could have
getSupportValue(): float

Returns the support value of the provided subgraph.

Input: subgraph : Subgraph
Precondition: The provided subgraph is part of the anomalous subgraph discovery result.
Postcondition: The support of the provided subgraph is returned.

Could have
getNumberOfOccurrences(): integer

Returns the number of occurrences of the provided subgraph.

Input: subgraph : Subgraph
Precondition: The provided subgraph is part of the anomalous subgraph discovery result.
Postcondition: The number of occurrences of the provided subgraph is returned.
SR-103

**expand():** Subgraph[]

*Must have*

Returns the fully expanded version of all provided subgraphs.

**Input:** subgraphs : Subgraph[]

**Precondition:** The provided subgraphs are part of the anomalous subgraph discovery result.

**Postcondition:** The fully expanded versions of the provided subgraphs are returned.

---

SR-104

**filter():** Subgraph[]

*Could have*

Filters the subgraphs of the result using the provided support value and filter type and returns these subgraphs.

**Input:** supportValue : float

```
   filterType : string
```

**Precondition:** The provided filter type is properly defined.

**Postcondition:** The filtered subgraphs are returned.

---

SR-105

**exportPNG():** file

*Could have*

Export the provided subgraphs to a PNG file.

**Input:** subgraphs : Subgraph[]

**Precondition:** The provided subgraphs are part of the anomalous subgraph discovery result.

**Postcondition:** The provided subgraphs are exported to a PNG file.

---

SR-106

**exportG():** file

*Could have*

Export the provided subgraphs to a .g file.

**Input:** subgraphs : Subgraph[]

**Precondition:** The provided subgraphs are part of the anomalous subgraph discovery result.

**Postcondition:** The subgraphs are exported to a .g file.

---

SR-107

**getParents():** Subgraph[]

*Must have*

Returns the intersection of parents of the provided subgraphs up to the provided depth level.

**Input:** depth : integer

```
   subgraphs : Subgraph[]
```

**Precondition:** The provided depth is properly defined. The provided subgraphs are part of the anomalous subgraph discovery result.

**Postcondition:** The parents, up to the provided depth level, of the provided subgraphs are returned.

---

SR-108

**getChildren():** Subgraph[]

*Must have*

Returns the intersection of children of the provided subgraphs up to the provided depth level.

**Input:** depth : integer

```
   subgraphs : Subgraph[]
```

**Precondition:** The provided depth is properly defined. The provided subgraphs are part of the anomalous subgraph discovery result.

**Postcondition:** The children, up to the provided depth level, of the provided subgraphs are displayed.
SR-109

\textit{delete}(): void

Deletes the experiment phase result.

\textbf{Input}: -

\textbf{Precondition}: The experiment phase is idle. The user initiating this operation is a project member of the project this anomalous subgraph discovery result is part of or is an administrator. This user is logged in.

\textbf{Postcondition}: The experiment phase is deleted.

\textit{PartialOrderDiscoveryResult}

\textbf{SR-110}

\textit{id}: integer

The unique identifier of the experiment phase result.

\textbf{SR-111}

\textit{numberOfPatterns}: integer

The number of patterns this partial order discovery result consists of.

\textbf{SR-112}

\textit{averageSupport}: float

The average support value of this partial order discovery result.

\textbf{SR-113}

\textit{expand}(): Pattern[]

Returns the fully expanded version of all the provided patterns.

\textbf{Input}: \textit{patterns}: Pattern[]

\textbf{Precondition}: The provided patterns are part of the partial order discovery result.

\textbf{Postcondition}: The fully expanded patterns are returned.

\textbf{SR-114}

\textit{delete}(): void

Deletes the experiment phase result.

\textbf{Input}: -

\textbf{Precondition}: The experiment phase is idle. The user initiating this operation is a project member of the project this partial order discovery result is part of or is an administrator. This user is logged in.

\textbf{Postcondition}: The experiment phase is deleted.
## 4 Requirements traceability matrix

### 4.1 Software requirements to user requirements

<table>
<thead>
<tr>
<th>SRD</th>
<th>URD</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR-1</td>
<td>URF-1, URF-2, URF-3, URF-4, URF-5, URF-6, URF-7, URF-8, URF-9, URF-10, URF-11, URF-12, URF-13, URF-14, URF-15, URF-16, URF-17, URF-18, URF-19, URF-20, URF-21</td>
</tr>
<tr>
<td>SR-2</td>
<td>URF-1</td>
</tr>
<tr>
<td>SR-3</td>
<td>URF-4</td>
</tr>
<tr>
<td>SR-4</td>
<td>URF-12, URF-13, URC-2</td>
</tr>
<tr>
<td>SR-5</td>
<td>URF-6, URF-7, URF-8, URF-9, URF-10, URF-11</td>
</tr>
<tr>
<td>SR-6</td>
<td>URF-17, URC-2</td>
</tr>
<tr>
<td>SR-7</td>
<td>URF-18, URC-2</td>
</tr>
<tr>
<td>SR-8</td>
<td>URF-2, URC-2</td>
</tr>
<tr>
<td>SR-9</td>
<td>URF-3, URC-2</td>
</tr>
<tr>
<td>SR-10</td>
<td>URF-15, URC-2</td>
</tr>
<tr>
<td>SR-11</td>
<td>URF-16, URC-2</td>
</tr>
<tr>
<td>SR-12</td>
<td>URF-19, URC-2</td>
</tr>
<tr>
<td>SR-13</td>
<td>URF-43, URF-45, URF-46, URC-2</td>
</tr>
<tr>
<td>SR-14</td>
<td>URF-20, URC-2</td>
</tr>
<tr>
<td>SR-15</td>
<td>URF-23</td>
</tr>
<tr>
<td>SR-16</td>
<td>URF-25</td>
</tr>
<tr>
<td>SR-17</td>
<td>URF-34, URF-35, URF-36, URF-38</td>
</tr>
<tr>
<td>SR-18</td>
<td>URF-33</td>
</tr>
<tr>
<td>SR-20</td>
<td>URC-2</td>
</tr>
<tr>
<td>SR-21</td>
<td>URC-2</td>
</tr>
<tr>
<td>SR-22</td>
<td>URC-2</td>
</tr>
<tr>
<td>SR-23</td>
<td>URF-47, URF-48, URF-49, URF-50</td>
</tr>
<tr>
<td>SR-24</td>
<td>URF-52</td>
</tr>
<tr>
<td>SR-25</td>
<td>URF-53</td>
</tr>
<tr>
<td>SR-26</td>
<td>URF-58, URF-59</td>
</tr>
<tr>
<td>SR-27</td>
<td>URF-57, URF-60</td>
</tr>
<tr>
<td>SR-28</td>
<td>URF-61</td>
</tr>
<tr>
<td>SR-29</td>
<td>URF-62</td>
</tr>
<tr>
<td>SR-30</td>
<td>URF-2, URF-3, URF-4, URF-5, URF-6, URF-7, URF-8, URF-9, URF-10, URF-11, URF-12, URF-13, URF-14, URF-15, URF-16, URF-17, URF-18, URF-19, URF-20, URF-37</td>
</tr>
<tr>
<td>SR-31</td>
<td>URF-39</td>
</tr>
<tr>
<td>SR-32</td>
<td>URF-38</td>
</tr>
<tr>
<td>SR-33</td>
<td>URF-45</td>
</tr>
<tr>
<td>SR-34</td>
<td>URF-46</td>
</tr>
<tr>
<td>SR-35</td>
<td>URF-43</td>
</tr>
<tr>
<td>SR-36</td>
<td>URF-43</td>
</tr>
<tr>
<td>SR-37</td>
<td>URF-43</td>
</tr>
<tr>
<td>SR-38</td>
<td>URF-43</td>
</tr>
<tr>
<td>SR-39</td>
<td>URF-44, URC-2</td>
</tr>
<tr>
<td>SR-40</td>
<td>URC-2</td>
</tr>
<tr>
<td>SR-41</td>
<td>URF-47</td>
</tr>
<tr>
<td>SR-42</td>
<td>URF-47, URF-48</td>
</tr>
<tr>
<td>SR-43</td>
<td>URF-47, URF-50</td>
</tr>
<tr>
<td>SR-44</td>
<td>URF-47, URF-49</td>
</tr>
<tr>
<td>SR-45</td>
<td>URF-47</td>
</tr>
<tr>
<td>SR-46</td>
<td>URF-55</td>
</tr>
<tr>
<td>SR-47</td>
<td>URF-42, URF-56</td>
</tr>
<tr>
<td>SR-48</td>
<td>URF-73, URF-74, URF-75, URF-76, URF-77, URF-78, URF-79</td>
</tr>
<tr>
<td>SR-49</td>
<td>URF-110, URF-111, URF-112, URF-113</td>
</tr>
<tr>
<td>SRD</td>
<td>URD</td>
</tr>
<tr>
<td>-----</td>
<td>--------------------------</td>
</tr>
<tr>
<td>SR-50</td>
<td>URF-68</td>
</tr>
<tr>
<td>SR-51</td>
<td>URF-69</td>
</tr>
<tr>
<td>SR-52</td>
<td>URF-51</td>
</tr>
<tr>
<td>SR-53</td>
<td>URF-24, URF-41</td>
</tr>
<tr>
<td>SR-54</td>
<td>URF-24, URF-41</td>
</tr>
<tr>
<td>SR-55</td>
<td>URF-24, URF-41, URF-63, URF-69, URF-80</td>
</tr>
<tr>
<td>SR-56</td>
<td>URF-24, URF-41</td>
</tr>
<tr>
<td>SR-57</td>
<td>URF-55, URF-65</td>
</tr>
<tr>
<td>SR-58</td>
<td>URF-42, URF-56</td>
</tr>
<tr>
<td>SR-59</td>
<td>URF-82</td>
</tr>
<tr>
<td>SR-60</td>
<td>URF-96</td>
</tr>
<tr>
<td>SR-61</td>
<td>URF-82</td>
</tr>
<tr>
<td>SR-63</td>
<td>URF-82, URF-83, URF-84, URF-85, URF-86, URF-87, URF-88, URF-89, URF-90, URF-91, URF-92, URF-93, URF-94, URF-95</td>
</tr>
<tr>
<td>SR-64</td>
<td>URF-24, URF-41</td>
</tr>
<tr>
<td>SR-65</td>
<td>URF-76</td>
</tr>
<tr>
<td>SR-66</td>
<td>URF-77</td>
</tr>
<tr>
<td>SR-67</td>
<td>URF-78</td>
</tr>
<tr>
<td>SR-68</td>
<td>URF-79</td>
</tr>
<tr>
<td>SR-69</td>
<td>URF-63, URF-69, URF-80</td>
</tr>
<tr>
<td>SR-70</td>
<td>URF-24, URF-41</td>
</tr>
<tr>
<td>SR-71</td>
<td>URF-70, URF-71, URF-72</td>
</tr>
<tr>
<td>SR-72</td>
<td>URF-70</td>
</tr>
<tr>
<td>SR-73</td>
<td>URF-24, URF-41, URF-71, URF-72, URF-76, URF-77, URF-78, URF-79</td>
</tr>
<tr>
<td>SR-74</td>
<td>URF-81</td>
</tr>
<tr>
<td>SR-75</td>
<td>URF-42, URF-56</td>
</tr>
<tr>
<td>SR-76</td>
<td>URF-24, URF-41</td>
</tr>
<tr>
<td>SR-77</td>
<td>URF-74</td>
</tr>
<tr>
<td>SR-78</td>
<td>URF-75</td>
</tr>
<tr>
<td>SR-79</td>
<td>URF-63, URF-69, URF-80</td>
</tr>
<tr>
<td>SR-80</td>
<td>URF-24, URF-41</td>
</tr>
<tr>
<td>SR-81</td>
<td>URF-70, URF-71, URF-72</td>
</tr>
<tr>
<td>SR-82</td>
<td>URF-70</td>
</tr>
<tr>
<td>SR-83</td>
<td>URF-24, URF-41, URF-71, URF-72, URF-74, URF-75</td>
</tr>
<tr>
<td>SR-84</td>
<td>URF-81</td>
</tr>
<tr>
<td>SR-85</td>
<td>URF-42, URF-56</td>
</tr>
<tr>
<td>SR-86</td>
<td>URF-71, URF-72</td>
</tr>
<tr>
<td>SR-87</td>
<td>URF-63, URF-69, URF-80</td>
</tr>
<tr>
<td>SR-88</td>
<td>URF-71, URF-72</td>
</tr>
<tr>
<td>SR-89</td>
<td>URF-71, URF-72</td>
</tr>
<tr>
<td>SR-90</td>
<td>URF-81</td>
</tr>
<tr>
<td>SR-91</td>
<td>URF-42, URF-56</td>
</tr>
<tr>
<td>SR-92</td>
<td>URF-115</td>
</tr>
<tr>
<td>SR-93</td>
<td>URF-63, URF-69, URF-80</td>
</tr>
<tr>
<td>SR-94</td>
<td>URF-114, URF-115</td>
</tr>
<tr>
<td>SR-95</td>
<td>URF-114, URF-115</td>
</tr>
<tr>
<td>SR-96</td>
<td>URF-71, URF-72, URF-115</td>
</tr>
<tr>
<td>SR-97</td>
<td>URF-81</td>
</tr>
<tr>
<td>SR-98</td>
<td>URF-42, URF-56</td>
</tr>
<tr>
<td>SR-99</td>
<td>URF-64</td>
</tr>
<tr>
<td>SR-100</td>
<td>URF-110, URF-113</td>
</tr>
<tr>
<td>SR-101</td>
<td>URF-99</td>
</tr>
<tr>
<td>SR-102</td>
<td>URF-100</td>
</tr>
<tr>
<td>SR-103</td>
<td>URF-97</td>
</tr>
<tr>
<td>SR-104</td>
<td>URF-107, URF-108, URF-109</td>
</tr>
<tr>
<td>SRD</td>
<td>URD</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>SR-105</td>
<td>URF-105</td>
</tr>
<tr>
<td>SR-106</td>
<td>URF-106</td>
</tr>
<tr>
<td>SR-107</td>
<td>URF-101, URF-103</td>
</tr>
<tr>
<td>SR-108</td>
<td>URF-102, URF-104</td>
</tr>
<tr>
<td>SR-109</td>
<td>URF-42, URF-56</td>
</tr>
<tr>
<td>SR-110</td>
<td>URF-66</td>
</tr>
<tr>
<td>SR-111</td>
<td>URF-112</td>
</tr>
<tr>
<td>SR-112</td>
<td>URF-110</td>
</tr>
<tr>
<td>SR-113</td>
<td>URF-98</td>
</tr>
<tr>
<td>SR-114</td>
<td>URF-42, URF-56</td>
</tr>
</tbody>
</table>

4.2 User requirements to software requirements

<table>
<thead>
<tr>
<th>URD</th>
<th>SRD</th>
</tr>
</thead>
<tbody>
<tr>
<td>URF-1</td>
<td>SR-1, SR-2</td>
</tr>
<tr>
<td>URF-2</td>
<td>SR-1, SR-8, SR-30</td>
</tr>
<tr>
<td>URF-3</td>
<td>SR-1, SR-9, SR-30</td>
</tr>
<tr>
<td>URF-4</td>
<td>SR-1, SR-3, SR-30</td>
</tr>
<tr>
<td>URF-5</td>
<td>SR-1, SR-30</td>
</tr>
<tr>
<td>URF-6</td>
<td>SR-1, SR-5, SR-30</td>
</tr>
<tr>
<td>URF-7</td>
<td>SR-1, SR-5, SR-30</td>
</tr>
<tr>
<td>URF-8</td>
<td>SR-1, SR-5, SR-30</td>
</tr>
<tr>
<td>URF-9</td>
<td>SR-1, SR-5, SR-30</td>
</tr>
<tr>
<td>URF-10</td>
<td>SR-1, SR-5, SR-30</td>
</tr>
<tr>
<td>URF-11</td>
<td>SR-1, SR-5, SR-30</td>
</tr>
<tr>
<td>URF-12</td>
<td>SR-1, SR-4, SR-30</td>
</tr>
<tr>
<td>URF-13</td>
<td>SR-1, SR-4, SR-30</td>
</tr>
<tr>
<td>URF-14</td>
<td>SR-1, SR-30</td>
</tr>
<tr>
<td>URF-15</td>
<td>SR-1, SR-10, SR-30</td>
</tr>
<tr>
<td>URF-16</td>
<td>SR-1, SR-11, SR-30</td>
</tr>
<tr>
<td>URF-17</td>
<td>SR-1, SR-6, SR-30</td>
</tr>
<tr>
<td>URF-18</td>
<td>SR-1, SR-7, SR-30</td>
</tr>
<tr>
<td>URF-19</td>
<td>SR-1, SR-12, SR-30</td>
</tr>
<tr>
<td>URF-20</td>
<td>SR-1, SR-14, SR-30</td>
</tr>
<tr>
<td>URF-21</td>
<td>SR-1</td>
</tr>
<tr>
<td>URF-22</td>
<td></td>
</tr>
<tr>
<td>URF-23</td>
<td>SR-15</td>
</tr>
<tr>
<td>URF-25</td>
<td>SR-16</td>
</tr>
<tr>
<td>URF-26</td>
<td>SR-19</td>
</tr>
<tr>
<td>URF-27</td>
<td>SR-19</td>
</tr>
<tr>
<td>URF-28</td>
<td>SR-19</td>
</tr>
<tr>
<td>URF-29</td>
<td>SR-19</td>
</tr>
<tr>
<td>URF-30</td>
<td>SR-19</td>
</tr>
<tr>
<td>URF-31</td>
<td>SR-19</td>
</tr>
<tr>
<td>URF-32</td>
<td>SR-19</td>
</tr>
<tr>
<td>URF-33</td>
<td>SR-18</td>
</tr>
<tr>
<td>URF-34</td>
<td>SR-17</td>
</tr>
<tr>
<td>URF-35</td>
<td>SR-17</td>
</tr>
<tr>
<td>URF-36</td>
<td>SR-17</td>
</tr>
<tr>
<td>URF-37</td>
<td>SR-30</td>
</tr>
<tr>
<td>URF-38</td>
<td>SR-17, SR-32</td>
</tr>
<tr>
<td>URF-39</td>
<td>SR-31</td>
</tr>
<tr>
<td>URF-40</td>
<td></td>
</tr>
<tr>
<td>URD</td>
<td>SRD</td>
</tr>
<tr>
<td>------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>URF-44</td>
<td>SR-39</td>
</tr>
<tr>
<td>URF-46</td>
<td>SR-37</td>
</tr>
<tr>
<td>URF-48</td>
<td>SR-23, SR-42</td>
</tr>
<tr>
<td>URF-49</td>
<td>SR-23, SR-44</td>
</tr>
<tr>
<td>URF-50</td>
<td>SR-23, SR-43</td>
</tr>
<tr>
<td>URF-51</td>
<td>SR-24</td>
</tr>
<tr>
<td>URF-52</td>
<td>SR-25</td>
</tr>
<tr>
<td>URF-54</td>
<td>SR-29</td>
</tr>
<tr>
<td>URF-55</td>
<td>SR-46, SR-57</td>
</tr>
<tr>
<td>URF-57</td>
<td>SR-27</td>
</tr>
<tr>
<td>URF-58</td>
<td>SR-26</td>
</tr>
<tr>
<td>URF-59</td>
<td>SR-26</td>
</tr>
<tr>
<td>URF-60</td>
<td>SR-27</td>
</tr>
<tr>
<td>URF-61</td>
<td>SR-28</td>
</tr>
<tr>
<td>URF-62</td>
<td>SR-29</td>
</tr>
<tr>
<td>URF-64</td>
<td>SR-99</td>
</tr>
<tr>
<td>URF-65</td>
<td>SR-57</td>
</tr>
<tr>
<td>URF-66</td>
<td>SR-110</td>
</tr>
<tr>
<td>URF-67</td>
<td></td>
</tr>
<tr>
<td>URF-68</td>
<td>SR-50</td>
</tr>
<tr>
<td>URF-70</td>
<td>SR-71, SR-72, SR-81, SR-82</td>
</tr>
<tr>
<td>URF-73</td>
<td>SR-48</td>
</tr>
<tr>
<td>URF-74</td>
<td>SR-48, SR-77, SR-83</td>
</tr>
<tr>
<td>URF-75</td>
<td>SR-48, SR-78, SR-83</td>
</tr>
<tr>
<td>URF-76</td>
<td>SR-48, SR-65, SR-73</td>
</tr>
<tr>
<td>URF-77</td>
<td>SR-48, SR-66, SR-73</td>
</tr>
<tr>
<td>URF-78</td>
<td>SR-48, SR-67, SR-73</td>
</tr>
<tr>
<td>URF-79</td>
<td>SR-48, SR-68, SR-73</td>
</tr>
<tr>
<td>URF-81</td>
<td>SR-74, SR-84, SR-90, SR-97</td>
</tr>
<tr>
<td>URF-83</td>
<td>SR-62, SR-63</td>
</tr>
<tr>
<td>URF-84</td>
<td>SR-62, SR-63</td>
</tr>
<tr>
<td>URF-85</td>
<td>SR-62, SR-63</td>
</tr>
<tr>
<td>URF-86</td>
<td>SR-62, SR-63</td>
</tr>
<tr>
<td>URF-87</td>
<td>SR-63</td>
</tr>
<tr>
<td>URF-88</td>
<td>SR-62, SR-63</td>
</tr>
<tr>
<td>URF-89</td>
<td>SR-62, SR-63</td>
</tr>
<tr>
<td>URF-90</td>
<td>SR-63</td>
</tr>
<tr>
<td>URF-91</td>
<td>SR-62, SR-63</td>
</tr>
<tr>
<td>URF-92</td>
<td>SR-62, SR-63</td>
</tr>
<tr>
<td>URF-93</td>
<td>SR-63</td>
</tr>
<tr>
<td>URF-94</td>
<td>SR-62, SR-63</td>
</tr>
<tr>
<td>URF-95</td>
<td>SR-62, SR-63</td>
</tr>
<tr>
<td>URF-96</td>
<td>SR-60</td>
</tr>
<tr>
<td>URF-97</td>
<td>SR-103</td>
</tr>
<tr>
<td>URF-98</td>
<td>SR-113</td>
</tr>
<tr>
<td>URF-99</td>
<td>SR-101</td>
</tr>
<tr>
<td>URD</td>
<td>SRD</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>URF-100</td>
<td>SR-102</td>
</tr>
<tr>
<td>URF-101</td>
<td>SR-107</td>
</tr>
<tr>
<td>URF-102</td>
<td>SR-108</td>
</tr>
<tr>
<td>URF-103</td>
<td>SR-107</td>
</tr>
<tr>
<td>URF-104</td>
<td>SR-108</td>
</tr>
<tr>
<td>URF-105</td>
<td>SR-105</td>
</tr>
<tr>
<td>URF-106</td>
<td>SR-106</td>
</tr>
<tr>
<td>URF-107</td>
<td>SR-104</td>
</tr>
<tr>
<td>URF-108</td>
<td>SR-104</td>
</tr>
<tr>
<td>URF-109</td>
<td>SR-104</td>
</tr>
<tr>
<td>URF-110</td>
<td>SR-49, SR-100, SR-112</td>
</tr>
<tr>
<td>URF-111</td>
<td>SR-49</td>
</tr>
<tr>
<td>URF-112</td>
<td>SR-49, SR-111</td>
</tr>
<tr>
<td>URF-113</td>
<td>SR-49, SR-100</td>
</tr>
<tr>
<td>URF-114</td>
<td>SR-94, SR-95</td>
</tr>
</tbody>
</table>
A  User interface mock-ups

In this section, the design of the user interface is explained.

A.1  Login page

When the user opens the tool, they will arrive at the login page displayed in Figure 48. The user has to enter their username and password correctly in order to log in. The home page will then be displayed. Both input fields must be filled in to be able to click the Login button. If the user focuses the username input field and focuses out without filling in anything, the message: “Please enter your username” will be shown below the username field. The password input field works similarly. If the username and password do not match, an error message will be displayed below the password input field, as depicted in Figure 49. It is also possible to go to the register page and to the forgot password page. Lastly, the user can continue as guest. The user can only view the demo project in that case.

![Login page](image1)

![Login page with wrong password](image2)

On the forgot password screen, displayed in Figure 50, the user can enter their username or email address. If it exists, an email will be sent to the corresponding email address. Otherwise, this will be indicated below the input field. By clicking the question mark, the user can read information about what will happen after the password is reset. The user can also go back to the login page here.

![Forgot password page](image3)

A.2  Registration page

Figure 51 displays the page where a user can register for an account. The user must fill in the username, email address, password and password confirmation fields. Furthermore, the user must
accept the privacy regulations. The user can also fill in the optional input fields. If the user focuses an input field and then focuses out, a message will be displayed below the input field if it is not filled in properly. It is only possible to press the register button when all required input fields are filled in and there are no error messages displayed. The administrators will be notified by mail when a new user registers. The user can also go back to the login page or continue as a guest.

A.3 Home page

When a user is logged in, the user arrives at the home page displayed in Figure 52. This home screen displays a list of all projects that the user can access. The name of the project, the creation date, and the project owner are displayed for each project. Each project owned by the logged in user also has a Share button, allowing users to share the project with other users. Owned projects can also be deleted using the Delete button, which is the red trash can icon. All projects have a Files button to view, download or delete the files of the project. By clicking one of the projects, the project page of the project will open.

On the right, the user can filter through the list of projects. The user can search for a specific project name — or part of a name — or they can choose to only display projects in a specific interval. On the left, the user can navigate to other pages using the side navigation menu. This menu will also be visible on other pages and can be collapsed by clicking the icon on the top left. Users can go back to the home page, view user information and log out using this menu. Administrators will also be able to access the admin tools page, this option is not displayed for normal users. Similarly, the
Create project button is not displayed when an unregistered user views the page as a guest. This button will open the modal displayed in Figure 53.

In Figure 53, the modal is displayed which is used to create a new project. The user needs to upload a .xes and a .pnml file. Also a project name must be added. If the name already exists, a message will be displayed below the input field. It is only possible to create the project if all input fields are filled in and there are no error messages.

While uploading the project files a loading icon will be displayed as in Figure 54.

In Figure 55, the modal used to share a project with other users is shown. The user can change and remove rights for users with whom the project is already shared. The user can also add a user to share the project with by searching for the user first.
A.4 Project page

After selecting a project, the project page depicted in Figure 56 will be displayed. This page contains a list of experiments with a creation date and a status. It is also possible to filter here, based on name, creation date or status, just like on the home page except for experiments. An experiment has three possible statuses: ‘Error’ if a phase has an error, ‘Running’ if there are no errors and exactly one phase is running or complete, and ‘Finished’ only when both phases have been run successfully. A new experiment can be created with the Create experiment button. This will open a modal, displayed in Figure 57, where only the name can be specified. The user can also compare results of different experiments with the Compare experiments button. The user will then be able to select experiments to show on the compare experiments page.

When an experiment is selected, it will expand, as displayed in Figure 56 in the row Delta Test Experiment Expanded. Here, the user can see what the status of each phase is. To stop a phase, the user can click the Stop button. With the Start a new phase button, new phases can be run.
Figure 57: Modal for experiment creation

Clicking the button opens the modal displayed in Figure 58. With the Logs button, the user can choose to download the log file or delete it. The Files button allows users to access all the files of an experiment and, depending on their privileges for the project, view and/or delete them. Lastly, the user can view the results with the View results button. This will open the results of the completed phases in the results page.

In Figure 58, the modal for adding a new experiment phase is displayed. The user can select one of the two phases, or add both phases simultaneously. Under select components, it is possible to deselect components in case the user has their own files already for that component. It is also possible to use a custom component, uploaded by an administrator, instead of a normal one. It will not be possible to run the phase(s) directly if the user deselects components, but the user will be able to use the Run from file button in that case. The user can upload the required files to run the
phase(s) there. Lastly, all required parameters must be entered. By clicking on the plus sign, the user can specify a range instead of a single value. New experiments will be created in that case.

In Figure 59, the compare experiments page is shown. This page shows the average support values and the number of patterns for each experiment. The experiments to compare are selected by the user on the project page. With the Change selection button, the user can show other experiments in the table. The results of an experiment can be opened directly with the View results button.
A.5 Results page

On the left, the side navigation menu is still displayed and can be collapsed. The page is divided in two parts. The top part displays the results of the anomalous subgraph discovery phase and the bottom part displays the results of the partial order discovery phase. Double-clicking a subgraph or pattern will expand that subgraph or pattern. In the expanded subgraphs in the anomalous subgraph discovery part, the support value and number of occurrences is displayed. In the patterns in the partial order discovery part, the support value is displayed. If the phase is not completed yet, the tool will display that no results have been found for that phase. Each phase has its own toolbar above the results.

For the anomalous subgraph discovery phase, the following options can be used from left to right:

- **Select all.** This selects all subgraphs.
- **Deselect all.** This deselects all subgraphs.
- **Expand.** This expands all selected subgraphs.
- **Compress.** This compresses all selected subgraphs.
- **Open in new tab.** This opens all selected subgraphs in a new tab.
- **Filter.** This allows the user to filter the subgraphs by support value, or select another patterns kind (minimal or maximal).
- **Download.** The user can select a file to download here. This could be the .svg or the .dot file, but also one of the occurrence matrices or the .g or .subs file. The user can also download the selected subgraph(s) as a .png or .g file.
- **Search.** The user can search for a subgraph that contains the searched string here.
- **Search results.** This displays a list of subgraphs that contain the searched string. It also displays which of the search results are expanded subgraphs and which are compressed.
- **Parent/child levels.** The user can specify here how many levels of parents and children should be displayed in the parents/children results.
- **Parent/child results.** This displays a list of subgraphs that are parents or children of all selected subgraphs. Also parents at a deeper level or children at a deeper level will be indicated if the user specified this in the previous field.
For the partial order discovery phase, the functionality is very similar. The differences are that the download button will now only include files related to this phase, it is not possible to open partial orders in a new tab, and there are no parents/children results.

A.6 User information page

![User information page](image)

Figure 61: User information page

By using the side navigation, the user can navigate to the user information page displayed in Figure 61. On this page, all information is displayed that was provided during registration. The user can modify this information by pressing the **Edit information** button. It then becomes possible to edit the text fields and select a different home country. The user can also change the current password using the **Change password** button. The user has to specify the old password and the new password in that case. Lastly, the user can request deletion of the user log with the **Delete user log** button and view the user log with the **View user log** button.

A.7 Admin tools

Only administrators can access the admin tools page displayed in Figure 62. On this page, a list of all users is displayed. Projects are not shown here, because administrators can see all projects on the home page. With the **Create user** button, the administrator can manually create new accounts. A modal will then be displayed and the administrator can register a new user, just like on the registration page. To the right of the list of users, the administrator has an overview of all custom components. With the **Upload** button, he can upload a new custom component.

By clicking a user, it will expand, as shown in Figure 63. The administrator can only see this information: only the user can edit it. The administrator can also open the user activity log or delete it if a user requested that. Tracking a user can be enabled and disabled and the entire user account can also be deleted. Lastly, the administrator can reset the password of the user. The user will receive an email then, similar to what happens at the forgot password page.
Figure 62: Overview of admin page

Figure 63: Admin page with expanded user information
B Transitions

In appendix A the different pages in the user interface are described. In this section, the transitions between these interfaces are clarified using a transition diagram. The transition diagram is modelled as a Petri-net and is shown in Figure 64. Every page is modelled as a node (a circle) and every user action as a transition (a square). The model is not complete, global actions are omitted to keep the model clear. It is always possible to use a transition in opposite direction to go to the previous page, unless one of the pages is the login page. In the figure the complete model is displayed. Certain actions are restricted based on user role. This is indicated using colors within the nodes and squares. Only a user with the indicated role can perform that action or access that page. The bars on the right-top serve as legend.

When the user accesses the tool, they start on the login page. They have the options of creating an account, attempting to login, resetting their password or continuing as a guest. Choosing to proceed as guest will bring the user to the demo home view. As a guest, the user is able to view the demo project and related results. If they choose to login, incorrect credentials will result in 'invalid login' firing and the user simply remains in the login page. Correct credentials allow the user to access the home page.

From the home page it is possible to access side navigation menu. Using the side navigation menu, the user is able to log out, open the admin page, open the user information page and go back to the home page. The admin page is only accessible to administrators and is used for multiple purposes: viewing or deleting the user activity log, creating or deleting users, resetting the password of a user and uploading a custom component. On the user information page, the user can view and edit their account information. The user can also change their password or delete the account there. The latter will also bring the user back to the login page. From both the admin and user information page, the user can go back to the home page using the side navigation menu.

On the home page, the user can perform the following transitions: 'Create project', 'Share project', 'Delete project' and 'Open project'. Firing of 'Create project' will show a model on 'Project creation'. After successfully creating a new project the user will return to the home page. The 'Share project' and 'Delete project' transitions will be enabled when the user is a project owner. Firing the first transition will bring the user to 'Share project' where they can assign rights to users or update access rights of other users. 'Delete project' will delete the project and 'Open project' leads to the 'Project page'.

On the project page, the user can see all experiments of the corresponding project. If the user is a project observer, they can only fire the transitions 'Show results' and 'Compare experiment results'. The first is used to view the results of a complete experiment phase, the second one to compare multiple experiments. In case the user is a project member, they can also 'Create experiment', 'Delete experiment' and 'Run experiment phase'. When the user wants to create a new experiment, they can do so in the 'Experiment setup'. From the project page, a user can delete an experiment. In case the user wants to run an experiment phase, the 'Experiment phase setup' will be shown and needs to be filled in.
Figure 64: Transition diagram of the Delta extension