Abstract

This document contains the software requirements for the QIS system. This program is part of the Software Engineering Project (2IP35) at Eindhoven University of Technology.

The requirements in this SRD (Software Requirements Document) satisfy the requirements in the URD[1]. The document complies with the Software Requirements Document (SRD) from the Software Engineering Standard, as set by the European Space Agency.
# Contents

1 Introduction .......................................................... 6
   1.1 Purpose .................................................................. 6
   1.2 Scope .................................................................. 6
   1.3 List of definitions .................................................. 6
   1.4 List of references ................................................... 6
   1.5 Overview ................................................................ 7

2 General description ...................................................... 8
   2.1 Relation to current projects ....................................... 8
   2.2 Relation to predecessor and successor projects .......... 8
   2.3 Function and purpose ............................................. 8
   2.4 Environment ....................................................... 8
   2.5 Relation to other systems ......................................... 9
      2.5.1 OWIS ............................................................. 9
      2.5.2 Syllabus+ ....................................................... 9
      2.5.3 NT authentication ........................................... 10
      2.5.4 Human resource department's ORCA ................ 10
   2.6 General constraints ................................................ 10
   2.7 Model description ................................................. 11
      2.7.1 Petri nets ........................................................ 11
      2.7.2 Login flow ...................................................... 11
      2.7.3 Account management ...................................... 11
      2.7.4 Provisional versions ........................................ 12
      2.7.5 Class diagram ................................................ 13
      2.7.6 Interactions .................................................... 13

3 Specific requirements .................................................. 21
   3.1 Helper functions .................................................. 21
      3.1.1 today ............................................................ 21
   3.2 Invariants ........................................................... 21
      3.2.1 Associations .................................................. 22
      3.2.2 Temporal ...................................................... 23
   3.3 Remarks on notation .............................................. 24
      3.3.1 Shorthand: attributes and associations of sets ...... 24
      3.3.2 Hire.ratio ...................................................... 24
      3.3.3 User .............................................................. 24
3.3.4 Types ........................................... 24
3.4 Functional requirements ........................................... 24
  3.4.1 CourseInstance ........................................... 24
  3.4.2 EducationTask ........................................... 26
  3.4.3 Employee ........................................... 27
  3.4.4 Course ........................................... 30
  3.4.5 EducationType ........................................... 31
  3.4.6 Department ........................................... 32
  3.4.7 Subdepartment ........................................... 33
  3.4.8 ExpertiseGroup ........................................... 36
  3.4.9 Task ........................................... 37
  3.4.10 Hire ........................................... 38
  3.4.11 Assignment ........................................... 40
  3.4.12 SystemYear ........................................... 40
  3.4.13 Period ........................................... 41
  3.4.14 SubPeriod ........................................... 42
  3.4.15 StudyProgram ........................................... 43
  3.4.16 TargetGroup ........................................... 43
  3.4.17 System ........................................... 44
  3.4.18 Notification ........................................... 45
  3.4.19 ExternalConnection ........................................... 45
  3.4.20 NT ........................................... 45
  3.4.21 ORCA ........................................... 45
  3.4.22 Syllabus+ ........................................... 46
  3.4.23 OWIS ........................................... 46
  3.4.24 Initial value requirements ........................................... 46
  3.4.25 Constraint requirements ........................................... 47
3.5 Non-functional requirements ........................................... 47
3.6 Rights table ........................................... 48
4 Reports list ........................................... 49
  4.1 Tasks related to own course instances ........................................... 49
  4.2 Tasks related to course instances of expertise group ........................................... 50
  4.3 Tasks related to course instances of subdepartment ........................................... 51
  4.4 Workload for a single employee ........................................... 51
  4.5 Workload of all employees of an expertise group ........................................... 52
  4.6 Workload of all employees of a subdepartment ........................................... 53
  4.7 Education workload for a course instance ........................................... 54
  4.8 Education workload for a target group ........................................... 55
  4.9 Education workload for a study program ........................................... 56
  4.10 Education workload for all study programs ........................................... 56
  4.11 Capacity of an expertise group ........................................... 57
  4.12 Capacity of a subdepartment ........................................... 58
  4.13 Budget for a subdepartment ........................................... 58
  4.14 Research hours for an expertise group ........................................... 60
  4.15 Research hours for a subdepartment ........................................... 60
  4.16 Management hours for an expertise group ........................................... 61
4.17 Management hours for a subdepartment ........................................... 61
4.18 Course instances and assigned employees for an education type .......... 62

5 Prototype
   5.1 Authentication ................................................................. 63
   5.2 Employees ........................................................................... 64
   5.3 Courses .............................................................................. 67
   5.4 Tasks .................................................................................. 71
   5.5 Administrative ..................................................................... 73
   5.6 Reports ................................................................................. 75

6 Requirements traceability matrix
   6.1 URD to SRD ........................................................................ 78
   6.2 SRD to URD ........................................................................ 79
Document Status Sheet

Document status overview

General

Document title: Software Requirements Document
Identification: SRD-1.0.1892
Authors: Michael van Duijkeren, Neal van den Eertwegh, Sander Leemans, Koen Kivits & Kevin van der Pol
Document status: Final

Document history

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Reason of change</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>5-10-2009</td>
<td>Sander</td>
<td>First hand-in version</td>
</tr>
</tbody>
</table>
Document Change Records since previous issue

General

Date: 08-09-2009
Document title: Software Requirements Document
Identification: SRD-1.0.1892

Changes

<table>
<thead>
<tr>
<th>Page</th>
<th>Paragraph</th>
<th>Reason to change</th>
</tr>
</thead>
</table>

5
Chapter 1

Introduction

1.1 Purpose

The SRD translates all of the specific requirements from the URD[1] into software requirements. These software requirements describe what the system must be able to do in order to satisfy the customer’s needs.

The software requirements form a logical model of the QIS application, giving a simplified view on what the application’s capabilities.

1.2 Scope

QIS is an application designed and developed by Group QIS for the Departement of Mathematics and Computer Science at the Eindhoven University of Technology. The purpose of the application is to support workload division within the department and to support various parties regarding the management of working hours and tasks, holidays, courses and employees.

1.3 List of definitions

| ESA | European Space Agency |
| SRD | Software Requirements Document |
| STU | Student Service Centre |
| URD | User Requirements Document |

1.4 List of references

1.5 Overview

The remainder of this document gives a general description of the application and its environment in chapter 2, software requirements in chapter 3 and a requirements traceability matrix, showing how each user requirement of the URD is linked to software requirements in the SRD in chapter 6.


Chapter 2

General description

2.1 Relation to current projects

There are no other projects related to Group QIS.

2.2 Relation to predecessor and successor projects

The predecessor of QIS is a custom written Access database. This database is outdated, nearly unadaptable to current changes, hard to extend and does not provide all information needed by those involved. Most of the work is done manually which makes the planning of workload time consuming and error-prone. QIS will solve most of the problems stated, it will keep a database that will be easy to maintain and most of the hardcoded design decisions in the current system will be adaptable.

At this moment, no plans for successors of QIS are known. Group QIS tries to make QIS as flexible as possible, to decrease the possibility that a successor is needed.

2.3 Function and purpose

QIS will be a tool for the workload managers of the department of Mathematics and Computer Science, which will provide a easy to use and efficient interface to manage the workload of all employees at the department of Mathematics and Computer Science. These employees will also use the system to view their personal workload.

2.4 Environment

QIS will be accessed through a web browser (QIS must have support for Microsoft Internet Explorer 7 and should have support for Mozilla Firefox 3.5). The backend of QIS will run on a web
The environment of QIS is specified in detail in section 2.5 of the URD[1].

The following users will use the system:

- Directors of education
- Human resource department
- Workload manager
- Scheduling coordinator
- Policy advisors
- Employees
- Bureau Computer Facilities

The characteristics of these users are specified in detail in section 2.4 of the URD[1].

### 2.5 Relation to other systems

QIS replaces two existing Access databases: one for mathematics, one for computer science. These systems are used for workload planning, but all data need to be entered manually. Every year the file containing the database is copied and a new workload planning is made.

QIS interfaces with several other systems used at the university. OWIS, Syllabus+, NT authentication and ORCA are existing systems QIS is planned to interact with or make use of.

A graphical representation of the environment of QIS is found in figure 2.1.

#### 2.5.1 OWIS

As said in the URD[1], OWIS is the educational information system at Eindhoven University of Technology. QIS has been in touch with the maintainers of the OWIS system, the Education and Student Service Center STU. A motivation of what QIS will need and provide, together with a description of security measures has to be handed in and approved before QIS will be able to access the OWIS system. If QIS gets accredited, it will have to connect the OWIS system via a VPN connection. QIS may then obtain and send data using web services.

#### 2.5.2 Syllabus+

Syllabus+ is the scheduling software that is used by scheduling coordinators at the university. This software can optimize timetabling around students’ module choices, staff preferences and the strategic goals of the institution. QIS should send information about longer absences to Syllabus+.
2.5.3 NT authentication

Employees and students of Eindhoven University of Technology are all assigned an NT account. This account is used to register with the network of the university. This account is also used to access a variety of services offered by the university, for example Studyweb. QIS should use this authentication protocol for authentication of its users that have an NT account. External lecturers (not employed by Eindhoven University of Technology) do not have an NT-account. They must be able to authenticate using a username and password provided by the administrator of QIS.

2.5.4 Human resource department’s ORCA

The ORCA system is used to support administrative processes of personnel and financial administration and is used by HR departments all over the university. If QIS would be able to connect to this system and obtain data about employees, capacity groups and expertise groups, then this data would not need to be entered manually. Group QIS has contacted the human resource department but before being provided with information about the data that may be obtained and sent, permission of several has to be obtained. If time allows, Group QIS will get those permissions and find out what data can be obtained. Data about employees, subdepartments and expertise groups can as well be obtained through the NT system and OWIS.

![Figure 2.1: Environment diagram](image)

2.6 General constraints

QIS should be easily maintainable. The methods used to program QIS should achieve this. This will not be tested but is assumed. First-time users should not have any difficulty getting familiar with QIS and further use should not cause any difficulties either.
2.7 Model description

2.7.1 Petri nets

In the following sections, the Petri net formalism is used. A formal definition of Petri nets is found in [2]. We will give a short, informal description of this formalism here. A Petri net consists of circular places and rectangular transitions. Places may contain tokens, represented by big dots. Each possible combination of tokens in places represents a state of the system.

A transition can account for a change of the state of the system whenever all places of which the transition has an incoming arc from have got a token. When fired, the transition will take one token from each place it has an incoming arc from and will place a token in all places to which it has an outgoing arc. Arcs are represented by arrows.

2.7.2 Login flow

Figure 2.2 shows the login flow of QIS for a single user. Users have to log in before they can perform any action. When a user does not have any more actions to perform, he or she can logout.

![Login flow diagram](image)

Figure 2.2: Login flow diagram

2.7.3 Account management

Figure 2.3 shows the account management flow of QIS for multiple users. A new account can be created, after which it can be given or denied rights in relation to a certain object (a subdepartment, expertise group, et cetera). An account can be granted or revoked an administrator status as well. An account can also be removed.
2.7.4 Provisional versions

Figure 2.4 shows the behavior of QIS for provisional versions. A provisional version and a definitive version of the workload planning always exist for a specific year and subdepartment. Only the provisional version can be edited. When the user thinks the provisional version is completed, it can be made definitive. Upon making a provisional version definitive, notifications are sent about the changes made to the definitive version. After this, it is still possible to change the provisional version and make the provisional version definitive, again triggering notifications to be sent.

When the user, manually, ends a year, nothing regarding that year can be modified anymore. When a year ends, the related provisional version will be made definitive.

Figure 2.4: Provisional versions diagram
2.7.5 Class diagram

An overview of the classes of QIS can be found in figure 2.5. A more detailed look of the System class can be found in figure 2.6, the Department class in figure 2.7, the Course class in figure 2.8, the Task class in figure 2.9 and the Employee class in figure 2.10.

Almost no information may be changed after a year has been closed. Due to this, QIS has a class SystemYear, to which almost everything is connected.

Nota bene: these models only generally describe the classes in QIS. Minor changes in implementation may occur.

2.7.6 Interactions

This section describes some high level interactions with and within QIS through message sequence charts (MSCs). A description of the semantics of MSCs can be found in a paper published as TU/e Computer Science report CSR00-12 [3].

The interaction with the system is depicted by messages sent by the environment to one of the objects in the system. The boundary of the charts represents the user interface of QIS, not the interaction of the user with the interface. Interactions within the system are described by message going back and forth between objects of the system. For information about the user interface, see [5]

Add course

The MSC in figure 2.11 shows the progress of adding a course, a course instance for that course and a education type for that course instance to a certain subdepartment. We assume a certain subdepartment and subperiod are already selected.

First, a course is added to a subdepartment, after which a new course instance is added to the newly created course. A list of the education types of the selected subdepartment is retrieved by the user. The user selects one of these types, and adds a new task of this type to the new course instance.

Add assignment to a course instance

Figure 2.12 describes the process of adding a task to a course instance and then assign an employee to that task. We assume that we have selected a certain subdepartment and subperiod, in order to get a list of course instances.

The user first requests a lists of course instances for the selected subperiod. The user then adds an education task to a certain course instance. When the education task is added, the user requests a list of all hires of the current subdepartment and selects one of those to assign it to the education task.
View course instance report

Figure 2.13 gives the MSC on how a Course instance report is generated. We assume the user is logged in as the instance of *Employee* in the MSC.

The user requests the report. The system will get a list of all hires connected to that user. The system then retrieves all assignments for each hire. For these assignments all education tasks are retrieved. For these education tasks all connected course instances are retrieved. It then retrieves all education tasks from each course instance. Then the system gets all assignments for these tasks. Then all hires connected to these hires are retrieved and all relevant course names are retrieved. Lastly, all employees connected to the retrieved hires are retrieved.

Then all data retrieved so far is used to generate a report. This report is then returned to the user.
Figure 2.5: Global class diagram
The page contains diagrams of class structures for System, SystemYear, StudyProgram, Subdepartment, Department, Task, EducationType, and ExpertiseGroup. The diagrams illustrate the relationships and methods associated with these classes.

**Figure 2.6: System class diagram**

**Figure 2.7: Department class diagram**
Figure 2.10: Employee class diagram
Figure 2.11: Add course

Figure 2.12: Add assignment to a course instance
Figure 2.13: View course instance report
Chapter 3

Specific requirements

The specific requirements discussed in this section are divided into logical subsections. Important to mention is that there is one subsection for each class. Methods and attributes are discussed in a subsection. When the word \textit{self} is used, it refers to an instance of the class involved.

For prioritizing the specific requirements for QIS, we will adhere to the MoSCoW model. The capital letters in MoSCoW stand for:

- M \textit{Must have}; these requirements are essential for the product.
- S \textit{Should have}; these requirements are not critical for the product to work, but are nearly as important as the \textit{must haves}, meaning they must be implemented if at all possible.
- C \textit{Could have}; requirements which are not critical to the product’s success. If they can be implemented with little development costs, they can increase customer satisfaction.
- W \textit{Would have}; these requirements will not be implemented in this project. However, they would be nice to have in future versions of the product.

The priority for each requirement is listed with the respective requirement.

3.1 Helper functions

3.1.1 today

The function \textit{today()} returns the current date.

3.2 Invariants

Invariants are statements that may not be invalidated at all. They are assumed in every precondition and every postcondition.
3.2.1 Associations

The associations invariants state that the associations according to the UML diagram remain consistent. For example, if a year consists of several subperiods, then the subperiods referred to in a specific year should all refer back to that same year.

1. Inv-A-H: \((\forall a: Assignment, h: Hire | a \in h.assignments \iff h = a.hire)\)
2. Inv-A-T: \((\forall a: Assignment, t: Task | a \in t.assignments \iff t = a.task)\)
3. Inv-C-C: \((\forall c_1, c_2: Course | c_1 \in c_2.predecessors \iff c_2 \in c_1.successors)\)
4. Inv-C-SD: \((\forall c: Course, sd: Subdepartment | c \in sd.courses \iff sd = c.givenBy)\)
5. Inv-CI-E: \((\forall ci: CourseInstance, e: Employee | ci \in e.responsibleFor \iff e \in ci.responsibles)\)
6. Inv-CI-ET: \((\forall s: Subperiod, et: EducationTask | s \in et.subperiods \iff et \in s.educationTasks)\)
7. Inv-D-Y: \((\forall d: Department, y: SystemYear | d \in y.departments \iff y = d.systemYear)\)
8. Inv-E-H: \((\forall e: Employee, h: Hire | e = h.employe \iff h \in e.hires)\)
9. Inv-E-R: \((\forall e: Employee, r: Right | e = r.employe \iff r \in e.rights)\)
10. Inv-D-SD: \((\forall d: Department, sd: Subdepartment | d = sd.department \iff sd \in d.subdepartments)\)
11. Inv-EI-ET: \((\forall ei: EducationTask, et: EducationType | ei \in et.educationTasks \iff et = ei.educationType)\)
12. Inv-ET-SD: \((\forall et: EducationType, sd: Subdepartment | et \in sd.educationTypes \iff et = sd.subdepartment)\)
13. Inv-EG-H: \((\forall eg: ExpertiseGroup, h: Hire | eg \in h.expertiseGroup \iff h \in eg.hires)\)
14. Inv-EG-SD: \((\forall eg: ExpertiseGroup, sd: Subdepartment | eg \in sd.expertiseGroups \iff sd = eg.subdepartment)\)
15. Inv-P-SP: \((\forall p: Period, sp: Subperiod | p = sp.period \iff sp \in p.subperiods)\)
16. Inv-P-Y: \((\forall p: Period, y: SystemYear | p \in y.periods \iff y = p.systemYear)\)
17. Inv-SD-T: \((\forall sd: Subdepartment, t: Task | sd = t.subdepartment \iff t \in sd.tasks)\)
18. Inv-SP-ET: \((\forall s: Subperiod, et: EducationTask | s \in et.subperiods \iff et \in s.educationTasks)\)
22. Inv-S-TG: (∀s : StudyProgram, tg : TargetGroup |
   s = tg.studyProgram ⇔ tg ∈ s.targetGroups)
23. Inv-S-Y: (∀s : StudyProgram, y : SystemYear |
   s ∈ y.studyPrograms ⇔ y = s.systemYear)
24. Inv-S-SY: (∀sy : SystemYear, s : System |
   sy ∈ s.systemYear ⇔ s = sy.system)
25. Inv-S-E: (∀s : System, e : Employee |
   e ∈ s.employees ⇔ s = e.system)
26. Inv-S-N: (∀s : System, n : Notification |
   n ∈ s.notifications ⇔ s = n.system)
27. Inv-S: (∀s1, s2 : System |
   s1 = s2)
28. Inv-S-EC: (∀s : System, ec : ExternalConnection |
   ec ∈ s.externalConnections ⇔ s = ec.system)

3.2.2 Temporal

The temporal invariants state that all years, periods and subperiods remain separated. For example, if a course is given for a specific target group, then the year of the target group and the year of the course should be the same. It should not be possible to access different time periods by only taking a different path through the UML diagram.

1. Inv-CI1: (∀ci : CourseInstance |
   (∀tg ∈ ci.targetGroups, sp ∈ ci.subPeriods |
   tg.studyProgram.systemYear = sp.period.systemYear))
2. Inv-CI2: (∀ci : CourseInstance |
   (∀tg ∈ ci.targetGroups |
   tg.studyProgram.systemYear =
   ci.course.subdepartment.department.systemYear))
3. Inv-CI3: (∀ci : CourseInstance |
   (∀tg ∈ ci.targetGroups, et ∈ ci.educationTypes |
   tg.studyProgram.systemYear =
   et.subdepartment.department.systemYear))
4. Inv-CI4: (∀ci : CourseInstance |
   ∀tg ∈ ci.targetGroups, sp ∈ ci.subPeriods |
   tg.studyProgram.systemYear = sp.period.systemYear))
5. Inv-CI5: (∀ci : CourseInstance |
   (∀ei ∈ educationTasks |
   ei.subPeriods ∈ ci.subPeriods ⇔ ci.subPeriods ∈ ci.subPeriods))
6. Inv-CI6: (∀ei : EducationTask |
   (∀s ∈ ei.subperiods |
   ei.subdepartment.department.systemYear = s.period.systemYear))
3.3 Remarks on notation

3.3.1 Shorthand: attributes and associations of sets

Attributes or associations of sets are shorthand notation for union of these attributes or associations of all elements of that set. For example, for some employee \( e \), the set \( e.hires.tasks \) means \( \{ \bigcup h.tasks \mid h \in e.hires \} \). Likewise, the set \( e.hires.fte \) means \( \{ h.fte \mid h \in e.hires \} \).

3.3.2 Hire.ratio

For some hire \( h \), the ratio \( h.educationRatio \) may also be written \( h.ratio(education) \), and similarly for the other task types. This can be used when iterating over task types, making it possible to pass the task type as if it were an argument, to access the corresponding ratio. For task type “absence”, this ratio equals 0.

3.3.3 User

In the pre- and postcondition of the methods in the following sections \( user \) is used as a global variable. This signifies the employee who is logged in in the current session.

For example \( user \in self.responsibles \) means: the current user is a member of the set of responsibilities.

3.3.4 Types

Whenever the type of an attribute is omitted, it is assumed an implementation detail.

3.4 Functional requirements

Meanings of terms used and not explained can be found in the URD [1].

3.4.1 CourseInstance

A course instance denotes when and to who a course is given. To achieve this, it has connections to target groups and subperiods.

Attributes

<table>
<thead>
<tr>
<th>SCR1</th>
<th>must have</th>
</tr>
</thead>
</table>

Class CourseInstance has an attribute \( definitive \) of type boolean which denotes whether it is part of the definitive (true) or provisional (false) workload.
SCR2 must have
Class CourseInstance has an attribute *studyPhase* which represents the study phase this course instance is given to. The current study phases are bachelor and master.

SCR3 must have
Class CourseInstance has an attribute *givenToExternals* of type boolean, which denotes whether the course is given to an audience primarily consisting of students from other subdepartments.

SCR4 should have
Class CourseInstance has an attribute *publicRemark* of type string which contains all public remarks about the course.

SCR5 should have
Class CourseInstance has an attribute *privateRemark* of type string which contains all private remarks about the course.

SCR6 must have
Class CourseInstance has an attribute *course* of type Course which connects the course instance to its course.

SCR7 must have
Class CourseInstance has an attribute *subperiods* which is a set of Subperiod, which connects the course instance to its subperiods.

SCR8 must have
Class CourseInstance has an attribute *responsibles* which is a set of Employee, which connects the course instance to its responsible employees.

SCR9 must have
Class CourseInstance has an attribute *targetgroups* which is a set of TargetGroup, which connects the course instance to its targetgroups.

Methods

SCR10 must have
Class CourseInstance has a method *getAssignedHours*(): integer, which returns the amount of hours in assignments to education tasks connected to this course instance.

Pre condition: *self.course.givenBy.hasRight(view, workload)*
\[ \land \text{right} = \text{own} \Rightarrow (\]
\[ \exists \text{user} \in \text{self.responsibles}
\[ \lor \text{user} \in \{hs \in \bigcup eI \in \text{self.educationTask} | eI.getHires()\} | hs.employee\}
\]

Post condition: result = \{\sum eI \in \text{self.educationTasks}\ | \{\sum a \in eI.getAssignments() | a.getHours\}\}

SCR11 must have
Class CourseInstance has a method *addTargetGroup*(targetGroup: TargetGroup), which adds a target group to this course instance.

Pre condition: *self.course.givenBy.hasRight(modify, course information)*
\[ \land tG = self.targetGroups\]

Post condition: *self.targetGroups = tG \cup \{targetGroup\}*

SCR12 must have
Class CourseInstance has a method `removeTargetGroup(targetGroup : TargetGroup)`, which removes a target group from this course instance.

Pre condition: `self.course.givenBy.hasRight(modify, course information) ∧ tG = self.targetGroups`

Post condition: `self.targetGroups = tG\{targetGroup}`

**SCR13**

Class CourseInstance has a method `addResponsibleEmployee(employee : Employee)`, which adds a responsible employee to this course instance.

Pre condition: `self.course.givenBy.hasRight(modify, course information) ∧ rs = self.responsibles`

Post condition: `self.responsibles = rs ∪ {employee}`

**SCR14**

Class CourseInstance has a method `removeResponsibleEmployee(employee : Employee)`, which removes a responsible employee from this course instance.

Pre condition: `course.givenBy.hasRight(modify, course information) ∧ rs = self.responsibles`

Post condition: `self.responsibles = rs\{employee}`

**SCR15**

Class CourseInstance has a method `addEducationTask(educationType : EducationType, subPeriod : Subperiod)`, which adds an education task to this course instance.

Pre condition: `self.course.givenBy.hasRight(modify, course information) ∧ subPeriod ∈ self.subperiods ∧ eIs = self.educationTasks`

Post condition: A new `ei : EducationTask has been created`

- `ei.subperiod = subPeriod`
- `ei.courseInstance = self`
- `self.educationTasks = eIs ∪ {ei}`

**SCR16**

Class CourseInstance has a method `removeEducationTask(educationTask : EducationTask)`, which removes an education task from this course instance.

Pre condition: `self.course.givenBy.hasRight(modify, course information) ∧ educationTask ∈ self.educationTasks ∧ self.educationTasks = eIs`

Post condition: `self.educationTasks = eIs\{educationTask}`

**3.4.2 EducationTask**

An education task is a special kind of task. It has most of the properties of another task, but it can be connected to course instances as well. With this connection, it links employees and courses together.

**Attributes**

**SCR17**

must have
Class EducationTask has an attribute \textit{weeklyHours} of type integer that represents the amount of hours per week it is given.

SCR18 must have
Class EducationTask has an attribute \textit{numberOfEmployees} of type integer that represents how many employees are needed for it.

SCR19 must have
Class EducationTask has an attribute \textit{toBeRemoved} of type boolean and is used in the provisional workload version when the education task gets deleted. When the provisional version is made definitive, all education tasks with \( \textit{toBeRemoved} = \text{true} \) get deleted.

SCR20 must have
Class EducationTask has an attribute \textit{educationType} of type EducationType, which connects the education task to its education type.

SCR21 must have
Class EducationTask has an attribute \textit{courseInstance} of type CourseInstance, which connects the education task to its course instance.

\section*{Methods}

\subsection*{3.4.3 Employee}

The Employee class represents employees employed by the department. An Employee instance also contains the login information. To represent employment, links to expertise groups via class Hire are used.

\section*{Attributes}

SCR22 should have
Class Employee has an attribute \textit{publicRemark} of type string which contains all public remarks about the employee.

SCR23 should have
Class Employee has an attribute \textit{privateRemark} of type string which contains all private remarks about the employee.

SCR24 must have
Class Employee has an attribute \textit{userName} of type string, the user name of the employee.

SCR25 must have
Class Employee has an attribute \textit{receiveNotificationsEmployee} of type boolean, which signifies whether or not an employee will receive notifications about changes in his or her workload.

SCR26 must have
Class Employee has an attribute \textit{receiveNotificationsExpertiseGroup} of type boolean, which signifies whether or not he or she will receive notifications about changes in the workload of employees in his or her expertise group.

SCR27 must have
Class Employee has an attribute \textit{firstName} of type string, the first name of the employee.

SCR28 must have
Class Employee has an attribute \textit{lastName} of type string, the last name of the employee.
Class Employee has an attribute rights which is a set of Right, which connects the employee to its rights.

Class Employee has an attribute responsibleFor which is a set of CourseInstance, which connects the employee to its course instances.

Class Employee has an attribute hires which is a set of Hire, which connects the employee to its expertise groups.

Class Employee has a method viewTasksRelatedToOwnCourseInstances(): report, which generates a report of all education tasks related to any course instance of which there also exist an education task which is assigned to this employee.

Pre condition: self.hasRight(view, workload)

Post condition: result = report

Class Employee has a method grantRight(action: string, object: object, what: string), which grants a right to this employee.

Pre condition: user.isAdministrator() ∧
                rs = self.rights

Post condition: A new r : Right has been created
                ∧ self.rights = rs \ {ri}
                ∧ ri.action = action
                ∧ ri.object = object
                ∧ ri.what = what

Class Employee has a method revokeRight(r: Right), which revokes this employee a right.

Pre condition: user.isAdministrator() ∧
                ∧ rs = self.rights
                ∧ r ∈ rs

Post condition: self.rights = rs \ {r}

Class Employee has a method verifyPassword(password: string): boolean, which returns whether the password is valid for the employee.

Pre condition: true

Post condition: if NT.enabled,
               result = NT.verifyPassword(userName, password).
Otherwise,
               result = the pair (self.userName, password) are valid credentials.
Class Employee has a method `getResponsibleFor() : CourseInstances`, which returns the course instances the user is responsible for and is allowed to view.

Pre condition: `subdepartment ∈ {eG.subdepartment | eG ∈ self.expertiseGroups}`

Post condition: `result =
{x ∈ self.responsibleFor |
subdepartment.hasRight(modify, course information)
∧ (¬x.definitive ∨ subdepartment.department.systemYear.ends < today())
}
∪ {x ∈ self.responsibleFor |
¬subdepartment.hasRight(modify, course information)
∧ x.definitive}`

**SCR37**

Class Employee has a method `viewWorkload() : report`, which generates a report of the workload of this employee.

Pre condition: `self.hasRight(view, workload)`

Post condition: `result = report` 4.4

**SCR38**

Class Employee has a method `fillUpResearch()`, which, if the employee has only one assignment on a research project, fills up all available hours of an employee. These hours are added to or distracted from the only research project that the employee has an assignment on.

**SCR39**

Class Employee has a method `getNotifications() : Notifications`, which returns all notifications sent to this employee.

**SCR40**

Class Employee has a method `isAdministrator() : boolean` which results whether the employee is an administrator or not.

**SCR41**

Class Employee has a method `hasRight(action, what) : boolean`, which returns whether the user has that right for this employee.

Post condition: `result =
(action = modify) ⇒ false
∧ user is logged on with user
∧ (
  ∃{ri ∈ user.right |
  ri.action = action
  ∧ ri.object = self
  ∧ ri.what = what
  }
) ∨ (
  ∃{h ∈ self.hires |
  h.expertiseGoup.hasRight(action, what)
  }
)`
3.4.4 Course

Courses are the base of education. The Course class represents courses and holds information that is not dependent of when and to who a course is given. External courses are addable, so complete curriculum lists can be entered.

Attributes

<table>
<thead>
<tr>
<th>SCR</th>
<th>must have</th>
<th>Class Course has an attribute...</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCR42</td>
<td></td>
<td>code of type string which contains the code of the course.</td>
</tr>
<tr>
<td>SCR43</td>
<td></td>
<td>name of type string which contains the name of the course.</td>
</tr>
<tr>
<td>SCR44</td>
<td></td>
<td>description of type string which contains a description of the course.</td>
</tr>
<tr>
<td>SCR45</td>
<td></td>
<td>credits of type float which represents the amount of ects for the course.</td>
</tr>
<tr>
<td>SCR46</td>
<td></td>
<td>givenExternally of type boolean, which denotes whether the course is given by an external subdepartment.</td>
</tr>
<tr>
<td>SCR47</td>
<td>should have</td>
<td>successor which is a set of Course, which represents which courses succeed this course.</td>
</tr>
<tr>
<td>SCR48</td>
<td>should have</td>
<td>predecessor which is a set of Course, which represents which courses this course succeeds.</td>
</tr>
<tr>
<td>SCR49</td>
<td></td>
<td>givenBy of type Subdepartment which denotes by which subdepartment this course is given.</td>
</tr>
<tr>
<td>SCR50</td>
<td></td>
<td>courseInstances which is a set of CourseInstance, which connects the course instance to its course.</td>
</tr>
<tr>
<td>SCR51</td>
<td>could have</td>
<td>customAttributeValues which is a list, which items represent the values of custom course attributes.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>SCR</th>
<th>must have</th>
<th>Class Course has a method...</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCR52</td>
<td></td>
<td>addCourseInstance(), which adds a course instance to this course.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pre condition: self.givenBy.hasRight(modify, course information) ∧ cIs = self.courseInstances</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Post condition: A new ci : CourseInstance has been created ∧ self.courseInstances = cIs ∪ {ci}</td>
</tr>
</tbody>
</table>
Class Course has a method `removeCourseInstance(c i : CourseInstance)`, which removes a course instance from this course.

**Pre condition:** `self.givenBy.hasRight(modify, course information) ∧ cI s = self.courseInstances`

**Post condition:** `self.courseInstances = cI s \{ci\}`

---

**SCR54**

Class Course has a method `getCourseInstances() : CourseInstances`, which returns the course instances of that course and the user is allowed to view.

**Pre condition:** `self.givenBy.hasRight(view, course information)`

**Post condition:** `result = {x ∈ self.courseInstances ∧ self.givenBy.hasRight(modify, course information) ∧ ¬x.definitive ∧ self.subdepartment.systemYear.ends < today()} ∪ {x ∈ self.courseInstances ∧ ¬self.givenBy.hasRight(modify, course information) ∧ x.definitive | x}`

---

### 3.4.5 EducationType

An education type is a way in which a course is given. It can be anything from instructions, project work and internship coaching to lectures.

**Attributes**

- **SCR55** (must have)
  Class EducationType has an attribute `name` of type string, which contains the name of the education type.

- **SCR56** (must have)
  Class EducationType has an attribute `subdepartment` of type Subdepartment, which denotes to which subdepartment this education type belongs.

- **SCR57** (could have)
  Class EducationType has an attribute `customAttributeValues` which is a list, which items represent the values of custom education type attributes.

**Methods**

- **SCR58** (must have)
  Class EducationType has a method `viewCourseInstancesAndAssignedEmployeesReport() : report`, which generates a report of all education tasks and assigned employees, for the education tasks of this type.

  **Pre condition:** `self.hasRight(view, workload)`

  **Post condition:** `result = report 4.18`
3.4.6  Department

A department is the biggest organisatorial unit within the university.

Attributes

<table>
<thead>
<tr>
<th>SCR</th>
<th>must have</th>
<th>Class Department has an attribute name of type string which contains the name of this department.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCR60</td>
<td>must have</td>
<td>Class Department has an attribute subdepartments which is a set of Subdepartment, which holds all subdepartments within this department.</td>
</tr>
<tr>
<td>SCR61</td>
<td>must have</td>
<td>Class Department has an attribute systemYear of type SystemYear, which connects this department to its system year.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>SCR62</th>
<th>must have</th>
<th>Class Department has a method addSubdepartment(), which adds a subdepartment to this department.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Pre condition: user.isAdministrator() ∧ cSDs = self.subdepartments</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Post condition: A new sd : Subdepartment has been created ∧ self.subdepartments = cSDs ∪ {sd}</td>
</tr>
<tr>
<td>SCR63</td>
<td>must have</td>
<td>Class Department has a method removeSubdepartment(subdepartment : Subdepartment), which removes a subdepartment from this department.</td>
</tr>
<tr>
<td>-------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pre condition: user.isAdministrator() ∧ cSDs = self.subdepartments</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Post condition: self.subdepartments = cSDs{sd}</td>
</tr>
<tr>
<td>SCR64</td>
<td>should have</td>
<td>Class Department has a method getNotifications() : Notifications, which returns all notifications sent to an employee of this department.</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SCR65</td>
<td>must have</td>
<td>Class Department has a method hasRight(action, what) : boolean, which returns whether the user has that right for this department.</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Post condition: result = (action = modify) ⇒ ¬self.systemYear.closed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>∧ user is logged on with user</td>
</tr>
<tr>
<td></td>
<td></td>
<td>∧ ∃ri ∈ user.right</td>
</tr>
<tr>
<td></td>
<td></td>
<td>∧ ri.object = self</td>
</tr>
<tr>
<td></td>
<td></td>
<td>∧ ri.what = what</td>
</tr>
</tbody>
</table>
3.4.7 Subdepartment

A subdepartment is the organisatorial unit within a department. It is an important class, as the workload planning is done within subdepartmental offices.

Attributes

SCR66 must have
Class Subdepartment has an attribute name of type string, which contains the name of the subdepartment.

SCR67 could have
Class Subdepartment has an attribute formula of type string. For an explanation of these formulas, please refer to section 3.21 of the URD[1].

SCR68 must have
Class Subdepartment has an attribute department of type Department, which denotes to which department this subdepartment belongs.

SCR69 must have
Class Subdepartment has an attribute courses which is a set of Course, which holds all courses given by this subdepartment.

SCR70 must have
Class Subdepartment has an attribute educationTypes which is a set of EducationType, which holds all education types belonging to this subdepartment.

SCR71 must have
Class Subdepartment has an attribute expertiseGroups which is a set of ExpertiseGroup, which holds all expertise groups within this subdepartment.

SCR72 must have
Class Subdepartment has an attribute tasks which is a set of Task, which holds all tasks belonging to this subdepartment.

SCR73 could have
Class Subdepartment has an attribute customEducationTypeAttributes which is a list of string, which represents the names of the custom education type attributes.

SCR74 could have
Class Subdepartment has an attribute customCourseAttributes which is a list of string, which represents the names of the custom course attributes.

Methods

SCR75 must have
Class Subdepartment has a method addEducationType(), which adds an education type to that subdepartment.

SCR76 must have
Class Subdepartment has a method removeEducationType(), which removes an education type from that subdepartment.

SCR77 must have
Class Subdepartment has a method addCourse(), which adds a course to that subdepartment.

SCR78 must have
Class Subdepartment has a method `removeCourse(course : Course)`, which removes a course from that subdepartment.

**SCR79** must have

Class Subdepartment has a method `addExpertiseGroup()`, which adds an expertise group to that subdepartment.

**SCR80** must have

Class Subdepartment has a method `removeExpertiseGroup(expertiseGroup : ExpertiseGroup)`, which removes an expertise group from that subdepartment.

**SCR81** must have

Class Subdepartment has a method `getUnplannedCapacity() : integer`, which returns the available hours for planning, for all employees of that subdepartment.

**SCR82** must have

Class Subdepartment has a method `addTask()`, which adds a task to that subdepartment.

**SCR83** must have

Class Subdepartment has a method `removeTask()`, which removes a task from that subdepartment.

**SCR84** must have

Class Subdepartment has a method `makeAssignmentsDefinitive()`, which makes the provisional version of the assignments definitive, overwriting the previous definitive version. Every employee whose assignment, assigned task or responsible courses change, gets a notification.

Pre condition: \{t.assignments | t ∈ self.tasks\} = A

Post condition: \{t.assignments | t ∈ self.tasks\} = A’

\((∀a ∈ A’ | (∃₁a’ ∈ A’ | (a’.task, ..., a’.definitive) = (a.task, ..., ¬a.definitive))) \)

\((∀a ∈ A ∧ ¬a.definitive | (∃₁a’ ∈ A’ | (a’.task, ..., a’.definitive) = (a.task, ..., true))) \)

where “...” means all other attributes of a and a’ in the same order. Also, an e-mail has been sent to each employee whose assignment, assigned task or responsible courses has changed, notifying that employee something has changed.

**SCR85** must have

Class Subdepartment has a method `makeCourseInstancesDefinitive()`, which makes the provisional version of the course instances definitive, overwriting the previous definitive version. Every employee whose assignment, assigned task or responsible courses change, gets a notification.

Pre condition: \{c.courseInstances | c ∈ self.courses\} = A

Post condition: \{c.courseInstances | c ∈ self.courses\} = A’

\((∀a ∈ A’ | (∃₁a’ ∈ A’ | (a’.responsibles, ..., a’.definitive) = (a.responsibles, ..., ¬a.definitive))) \)

\((∀a ∈ A ∧ ¬a.definitive | (∃₁a’ ∈ A’ | (a’.responsibles, ..., a’.definitive) = (a.responsibles, ..., true))) \)

where “...” means all other attributes of a and a’ in the same order. Also, an e-mail has been sent to each employee whose assignment, assigned task or responsible courses has changed, notifying that employee something has changed.
Class Subdepartment has a method `copyWorkload(target : Subdepartment)`, which copies all data regarding the workload of this subdepartment in this year to a `target` subdepartment in another year. The target subdepartment’s workload will be emptied. The rights contained in this subdepartment will be copied and adapted to the `target` subdepartment.

**SCR87**  
Class Subdepartment has a method `copyCourseInformation(target : Subdepartment)`, which copies all data regarding the courses of this subdepartment in this year to a subdepartment in another year. The target subdepartment’s courses will be emptied.

**SCR88**  
Class Subdepartment has a method `viewSubdepartmentWorkloadReport() : Report`, which generates a report of the workload of all employees of that subdepartment, grouped by expertise group.

Pre condition: `self.hasRight(view, workload)`
Post condition: `result = Report 4.6`

**SCR89**  
Class Subdepartment has a method `getNotifications() : Notifications`, which returns all notifications sent to an employee of this subdepartment.

**SCR90**  
Class Subdepartment has a method `hasRight(action, what) : boolean`, which returns whether the user has that right for that subdepartment.

Post condition: 

\[ \text{result} = \begin{cases} \text{true} & \text{if action = modify} \Rightarrow \neg \text{self.department.systemYear.closed} \land \text{user is logged on with user} \\
\exists \{ \text{ri} \in \text{user.right} \mid \\
\text{ri.action} = \text{action} \\
\land \text{ri.object} = \text{self} \\
\land \text{ri.what} = \text{what} \} \\
\lor \{ \\
\text{self.department.hasRight(action, what)} \\
\} \end{cases} \]

**SCR91**  
Class Subdepartment has a method `viewTasksRelatedToSubdepartmentCourseInstances() : Report`, which generates a report of the course instances of that subdepartment and the employees assigned.

Pre condition: `self.hasRight(view, workload)`
Post condition: `result = Report 4.3`

**SCR92**  
Class Subdepartment has a method `viewBudgetReport() : Report`, which generates a report of the number of planned hours, broken down to employee position, task type, study phase and internal and external courses.

Pre condition: `self.hasRight(view, workload)`
Post condition: `result = Report 4.13`
SCR93 must have
Class Subdepartment has a method \textit{viewResearchSubdepartmentReport()} : \textit{Report}, which
generates a report of the number of planned research hours.
Pre condition: \textit{self.hasRight(view, workload)}
Post condition: \textit{result} = \textit{Report 4.15}

SCR94 must have
Class Subdepartment has a method \textit{viewManagementSubdepartmentReport()} : \textit{Report},
which generates a report of the number of planned management hours.
Pre condition: \textit{self.hasRight(view, workload)}
Post condition: \textit{result} = \textit{Report 4.17}

SCR95 must have
Class Subdepartment has a method \textit{viewCapacitySubdepartmentReport()} : \textit{Report}, which
generates a report of the capacity of all employees of that subdepartment, i.e. the number of
fte multiplied by the number of hours of an fte.
Pre condition: \textit{self.hasRight(view, workload)}
Post condition: \textit{result} = \textit{Report 4.12}

3.4.8 ExpertiseGroup

An expertise group is a unit within a subdepartment. Employees are employed within it.

Attributes

SCR96 must have
Class ExpertiseGroup has an attribute \textit{name} of type string which contains the name of the
expertise group.

SCR97 must have
Class ExpertiseGroup has an attribute \textit{subdepartment} of type Subdepartment which denotes
to which subdepartment this expertise group belongs.

SCR98 must have
Class ExpertiseGroup has an attribute \textit{hires} which is a set of Hire, which holds all hires within
this expertise group.

Methods

SCR99 should have
Class ExpertiseGroup has a method \textit{getNotifications()} : \textit{Notifications}, which returns all
notifications sent to an employee of this expertise group.

SCR100 must have
Class ExpertiseGroup has a method hasRight(action, what) : boolean, which returns whether the user has that right for that expertise group.

Post condition: result =

\[
\begin{align*}
&\ (action = \text{modify}) \Rightarrow \neg \text{self.subdepartment.department.systemYear.closed} \\
&\land \text{user is logged on with user} \\
&\land ( \\
&\ (\exists ri \in \text{user.right} \mid \\
&\ ri.action = action \\
&\land ri.object = \text{self} \\
&\land ri.what = \text{what}) \\
&\} \\
&\lor ( \\
&\ \text{self.subdepartment.hasRight(action, what)} \\
&\}
\end{align*}
\]

SCR101 must have

Class ExpertiseGroup has a method viewExpertiseGroupWorkloadReport() : Report, which generates a report of the workload of all employees of that expertise group.

Pre condition: self.hasRight(view, workload)

Post condition: result = Report 4.5

SCR102 must have

Class ExpertiseGroup has a method viewTasksRelatedToExpertiseGroupCourseInstances() : Report, which generates a report of the course instances of that expertise group and the employees assigned.

Pre condition: self.hasRight(view, workload)

Post condition: result = Report 4.2

3.4.9 Task

A task can be one of the following: a research project, a management task, a category of longer leaves, an education task or an miscellaneous task. To a task, an assignment can be assigned. This assignment contains the actual planned hours for a hire of an employee.

Attributes

SCR103 must have

Class Task has an attribute name of type string which contains the name of the task.

SCR104 must have

Class Task has an attribute description of type string which contains a description of the task.

SCR105 must have

Class Task has an attribute hours of type integer which specifies to number of hours needed for this task.

SCR106 must have

Class Task has an attribute type, which can be any of the following values: “education”, “management”, “leave”, “research” and “miscellaneous”. It denotes the type of this task.

SCR107 should have
Class Task has an attribute `publicRemark` of type string which contains all public remarks about the task.

**SCR108**  
Class Task has an attribute `privateRemark` of type string which contains all private remarks about the task.

**SCR109**  
Class Task has an attribute `subdepartment` of type Subdepartment which denotes to which subdepartment this task belongs.

### Methods

**SCR110**  
Class Task has a method `addAssignment(hire : Hire)`, which adds an assignment with a hire to this task instance.

**SCR111**  
Class Task has a method `removeAssignment(assignment : Assignment)`, which removes an assignment from this task instance.

**SCR112**  
Class Task has a method `getAssignments() : Assignments`, which returns the assignments the user is allowed to view.

Pre condition: `self.department.hasRight(view, workload)`

Post condition:  
\[
result = \{as \in self.assignments | \\
self.department.hasRight(modify, workload) \\
\land (\neg as.definitive \\
\lor \neg as.hire.expertiseGroup.subdepartment.department.systemYear.closed)\} \\
\cup \\
\{as \in self.assignments | \\
\neg self.department.hasRight(modify, workload) \\
\land as.definitive\}
\]

**SCR113**  
Class Task has a method `viewTasksRelatedToSubdepartmentCourseInstances() : Report`, which generates a report of the course instances of that subdepartment and the employees assigned.

Pre condition: `self.hasRight(view, workload)`

Post condition:  
\[
result = \text{Report 4.3}
\]

### 3.4.10 Hire

A hire is a relation between an employee and an expertise group. It means that the employee is employed by the expertise group.

**Attributes**

**SCR114**  

Class Hire has an attribute `fte` of type float, which is the amount of fte’s the employee is to spend on this hire.

Class Hire must have an attribute `starts` of type date, which signifies the start date of the hire for the expertise group for an employee.

Class Hire must have an attribute `ends` of type date, which signifies the end date of the hire for the expertise group for an employee.

Class Hire must have an attribute `position` of type date, which signifies the position or rank the employee has within the expertise group.

Class Hire must have an attribute `educationRatio` which denotes what ratio of available hours is to be spent on education tasks.

Class Hire must have an attribute `researchRatio` which denotes what ratio of available hours is to be spent on research tasks.

Class Hire must have an attribute `managementRatio` which denotes what ratio of available hours is to be spent on management tasks.

Class Hire must have an attribute `expertisegroup` of type ExpertiseGroup, which signifies the expertise group this Hire is connected to.

Class Hire must have an attribute `employee` of type Employee, which signifies the employee associated with this hire.

Class Hire must have an attribute `assignments` which is a set of Assignment, which is a list of all the assignments that belong to this Hire.

Methods

Class Hire must have a method `getAssignments() : Assignments`, which returns the assignments the user is allowed to view.

Pre condition: `self.department.hasRight(view, workload)`

Post condition: `result =
\{as ∈ self.assignments |
    self.department.hasRight(modify, workload)
    ∧ (¬as.definitive
        ∨ ¬as.hire.expertisegroup.subdepartment.department.systemYear.closed)
\} ∪
\{as ∈ self.assignments |
    ¬self.department.hasRight(modify, workload)
    ∧ as.definitive\}`
3.4.11 Assignment

An assignment is a relation between a task (project) and a hire. It means that the employee works on the task.

Attributes

**SCR125** must have
Class Assignment has an attribute *hours* of type integer, which specifies the amount of hours needed for this assignment.

**SCR126** must have
Class Assignment has an attribute *share* of type float, which represents the share of the employee in the task.

**SCR127** must have
Class Assignment has an attribute *definitive* of type boolean, which denotes whether it is part of the definitive (*true*) or provisional (*false*) workload.

**SCR128** must have
Class Assignment has an attribute *externallyFunded* of type boolean, which denotes whether the assignment is externally funded.

**SCR129** must have
Class Assignment has an attribute *task* of type Task, which denotes to which task this assignment belongs.

**SCR130** must have
Class Assignment has an attribute *hire* of type Hire, which connects the assignment to its hire.

Methods

**SCR131** could have
Class Assignment has a method *getHours() : integer*, which returns the number of hours in this assignment. When the task is a research project, this is added to the outcome of the formula as mentioned in **SCR68**.

3.4.12 SystemYear

A year is self-explanatory. It is used by QIS to discriminate between years, as data from previous years may not be altered.

Attributes

**SCR132** must have
Class SystemYear has an attribute *fteAmount* of type integer that represents the number of hours that an fte comprises.

**SCR133** must have
Class SystemYear has an attribute *begins* of type date which holds the date on which the year begins.

**SCR134** must have
Class SystemYear has an attribute *ends* of type date which holds the date on which the year ends.

**SCR135**

Class SystemYear has an attribute *closed* of type boolean which denotes whether the year is finished and closed.

**SCR136**

Class SystemYear has an attribute *studyPrograms* which is a set of StudyProgram, which holds all study programs within that year.

**SCR137**

Class SystemYear has an attribute *departments* which is a set of Department, which holds all departments within that year.

**SCR138**

Class SystemYear has an attribute *periods* which is a set of Period, which holds all periods given within that year.

**Methods**

**SCR139**

Class SystemYear has a method *addPeriod()*, which adds a period within that year.

**SCR140**

Class SystemYear has a method *removePeriod()*, which removes a period within that year.

**SCR141**

Class SystemYear has a method *addStudyProgram()*, which adds a study program to that year.

**SCR142**

Class SystemYear has a method *removeStudyProgram()*, which removes a study program from that year.

**SCR143**

Class SystemYear has a method *close()*, which makes everything of the subdepartment of the user definitive and closes the year.

### 3.4.13 Period

A period is the largest part of an academic year. At writing time, these were called semesters.

**Attributes**

**SCR144**

Class Period has an attribute *name* of type string.

**SCR145**

Class Period has an attribute *systemYear* of type SystemYear, which denotes to what year this period belongs.

**SCR146**

Class Period has an attribute *subperiods* which is a set of Subperiod, which denotes in what subperiods this period is divided.
Methods

SCR147  must have
Class Period has a method addSubPeriod(), which adds a subperiod within that period.

SCR148  must have
Class Period has a method removeSubPeriod(), which removes a subperiod within that period.

3.4.14 SubPeriod

A period is divided into smaller sections, called subperiods. At writing time, these were called kwartielen. All regular education related workload is connected to one or more subperiods.

Attributes

SCR149  must have
Class SubPeriod has an attribute name of type string which contains the name of the subperiod.

SCR150  must have
Class SubPeriod has an attribute weeks of type integer, which represents the number of weeks within that subperiod.

SCR151  must have
Class Subperiod has an attribute period of type Period which represents the period in which this subperiod is given.

SCR152  must have
Class Subperiod has an attribute courseInstances which is a set of CourseInstance, which contains all course instances given within that subperiod.

SCR153  must have
Class Subperiod has an attribute educationTasks which is a set of EducationTask, which contains all education tasks that are executed in that subperiod.

Methods

SCR154  must have
Class SubPeriod has a method getCourseInstances() : CourseInstances, which returns the course instances of that subperiod and the user is allowed to view.

Pre condition: true
Post condition: result = 
\{x \in self.courseInstances |
  x.course.givenBy.hasRight(modify, course information))
  \land (\neg x.definite
  \lor subdepartment.systemYear.ends < today())\}
\cup \{x \in self.courseInstances |
  \neg x.course.givenBy.hasRight(modify, course information) \land x.definite\}
3.4.15 StudyProgram

A study program is in example bachelor technische informatica or the master business information systems. It contains target groups.

Attributes

SCR155 must have
Class StudyProgram has an attribute name of type string, which contains the name of the study program.

SCR156 must have
Class StudyProgram has an attribute systemYear of type SystemYear, which denotes to which year this study program belongs.

SCR157 must have
Class StudyProgram has an attribute targetgroups which is a set of TargetGroup, which contains all target groups connected to the study program.

Methods

SCR158 must have
Class StudyProgram has a method addTargetGroup(), which adds a target group to that study program.

SCR159 must have
Class StudyProgram has a method removeTargetGroup(), which removes a target group from that study program.

3.4.16 TargetGroup

A target group is in example the first year, a second year or a homologation program.

Attributes

SCR160 must have
Class TargetGroup has an attribute name of type string which contains the name of the target group.

SCR161 must have
Class TargetGroup has an attribute studyprogram of type StudyProgram which denotes to which study program this target group belongs.

SCR162 must have
Class TargetGroup has an attribute courseInstances which is a list of CourseInstance, which contains all course instances followed by this targetgroup.

Methods

SCR163 must have
Class TargetGroup has a method `getCourseInstances() : CourseInstances`, which returns the course instances the target group takes and the user is allowed to view.

Pre condition: `true`

Post condition:

\[
\text{result} = \{ x \in \text{self.courseInstances} \mid \\
\quad x.course.givenBy.hasRight(\text{modify, course information}) \wedge \\
\quad \neg x.\text{definitive} \\
\quad \lor \text{subdepartment.department.systemYear.ends < today}() \}
\]

\[
\cup \{ x \in \text{self.courseInstances} \mid \\
\quad \neg x.course.givenBy.hasRight(\text{modify, course information}) \\
\quad \wedge x.\text{definitive} \mid x\}
\]

3.4.17 System

The system is the single instance of a super class. It is used by QIS to keep a few sets that cannot be kept elsewhere.

Attributes

**SCR164** should have
Class System has an attribute `notifications`. This attribute contains all notifications generated by QIS.

**SCR165** must have
Class System has an attribute `systemYears` which is a set of SystemYear. This attribute contains all years created within QIS.

**SCR166** must have
Class System has an attribute `employees` which is a set of Employee. This attribute contains all employees in QIS.

Methods

**SCR167** must have
Class System has a method `login(username, password : string)`.

Pre condition: `true`

Post condition: if (\(\exists e : \text{Employee} \mid e.\text{userName} = \text{username} \wedge e.\text{verifyPassword}(\text{password})\)), the user is logged on with e.

**SCR168** must have
Class System has a method `addEmployee()`, which adds an employee to QIS.

**SCR169** must have
Class System has a method `removeEmployee(employee : Employee)`, which removes an employee from QIS.
3.4.18 Notification

Notifications are stored by QIS for archiving purposes.

Attributes

<table>
<thead>
<tr>
<th>SCR170</th>
<th>must have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class Notification has an attribute date of type date, which represents the date the notification was sent.</td>
<td></td>
</tr>
</tbody>
</table>

3.4.19 ExternalConnection

An external connection provides QIS with information, or provides other systems with information from QIS.

Attributes

<table>
<thead>
<tr>
<th>SCR171</th>
<th>must have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class ExternalConnection has an attribute enabled of type boolean, which signals whether the connection is enabled or not.</td>
<td></td>
</tr>
</tbody>
</table>

3.4.20 NT

Methods

<table>
<thead>
<tr>
<th>SCR172</th>
<th>should have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class NT has a method verifyPassword(userName, password : string) : boolean, which returns whether the credentials are valid according to the NT database.</td>
<td></td>
</tr>
<tr>
<td>Pre condition: true</td>
<td></td>
</tr>
<tr>
<td>Post condition: result = the NT database accepts the pair (userName, password) as valid credentials.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SCR173</th>
<th>should have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class NT has a method retrieveEmployees(), which retrieves employee information from the NT system and processes it into QIS.</td>
<td></td>
</tr>
</tbody>
</table>

3.4.21 ORCA

Methods

<table>
<thead>
<tr>
<th>SCR174</th>
<th>could have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class ORCA has a method retrieveLeaves(), which synchronizes all tasks of type Leave to match the information in the ORCA database. At writing time of this document the specification of the data was still unknown.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SCR175</th>
<th>could have</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

45
Class ORCA has a method `retrieveEmployees()` which retrieves employee information from ORCA and processes it into QIS. At writing time of this document the specification of the data was still unknown.

### 3.4.22 Syllabus+

**Methods**

**SCR176**  
Class Syllabus+ has a method `sendLeaves()`, which sends all tasks of type leave to the Syllabus+ system. At the moment of delivering this document the specification of the data is still unknown.

### 3.4.23 OWIS

**Methods**

**SCR177**  
Class OWIS has a method `sendCourses()`, which sends all courses and course instances to the OWIS system.

**SCR178**  
Class OWIS has a method `retrievePeriods()`, which retrieves data about periods and subperiods from OWIS and processes it into QIS. At the moment of delivering this document the specification of the data is still unknown.

**SCR179**  
Class OWIS has a method `retrieveTargetGroups()`, which retrieves data about target groups and study programs from OWIS and processes it into QIS. At the moment of delivering this document the specification of the data is still unknown.

### 3.4.24 Initial value requirements

**SCR180**  
QIS initially adheres to table 2.1 from the URD [1] for user rights.

**SCR181**  
QIS will send notifications via email.

**SCR182**  
Initially, all users receive notifications about their own workload.  
\[ account.receiveNotificationEmployee = true \]

**SCR183**  
Initially, all expertise group leaders receive no notifications about the workload in their expertise group.  
\[ account.receiveNotificationExpertiseGroup = false \]
3.4.25 Constraint requirements

**SCR184**
QIS allows users to access all the functions they are allowed to through HTTP\(^1\).  
*must have*

**SCR185**
QIS allows users to access all the functions they are allowed to via a user interface presented using a combination of HTML, CSS and Javascript.  
*must have*

**SCR186**
QIS allows users to access all the functions they are allowed to using Microsoft Internet Explorer 7.  
*must have*

**SCR187**
QIS allows users to access all the functions they are allowed to using Mozilla Firefox 3.5.  
*could have*

**SCR188**
An empty formula evaluates to 0.  
*must have*

**SCR189**
Calculated values are shown with their calculations.  
*could have*

3.5 Non-functional requirements

**SCR190**
QIS allows easy addition of connections to external systems through its code.  
*must have*

\(^1\) [http://www.w3.org/Protocols/rfc2616/rfc2616.html](http://www.w3.org/Protocols/rfc2616/rfc2616.html)
### 3.6 Rights table

<table>
<thead>
<tr>
<th>Req</th>
<th>Page</th>
<th>Action</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>25</td>
<td>CourseInstance.getAssignedHours</td>
<td>view subdepartment workload</td>
</tr>
<tr>
<td>11</td>
<td>25</td>
<td>CourseInstance.addTargetGroup</td>
<td>modify course information</td>
</tr>
<tr>
<td>12</td>
<td>25</td>
<td>CourseInstance.removeTargetGroup</td>
<td>modify course information</td>
</tr>
<tr>
<td>13</td>
<td>25</td>
<td>CourseInstance.addResponsibleEmployee</td>
<td>modify course information</td>
</tr>
<tr>
<td>14</td>
<td>25</td>
<td>CourseInstance.removeResponsibleEmployee</td>
<td>modify subdepartment course information</td>
</tr>
<tr>
<td>15</td>
<td>25</td>
<td>CourseInstance.addEducationTask</td>
<td>modify subdepartment course information</td>
</tr>
<tr>
<td>16</td>
<td>25</td>
<td>CourseInstance.removeEducationTask</td>
<td>modify subdepartment course information</td>
</tr>
<tr>
<td>32</td>
<td>28</td>
<td>Employee.viewTasksRelatedToOwnCourseInstances</td>
<td>view own workload</td>
</tr>
<tr>
<td>33</td>
<td>28</td>
<td>Employee.giveRight</td>
<td>administrator</td>
</tr>
<tr>
<td>34</td>
<td>28</td>
<td>Employee.revokeRight</td>
<td>administrator</td>
</tr>
<tr>
<td>36</td>
<td>28</td>
<td>Employee.getResponsibleFor (provisional)</td>
<td>modify course information</td>
</tr>
<tr>
<td>36</td>
<td>28</td>
<td>Employee.getResponsibleFor (definitive)</td>
<td>view course information</td>
</tr>
<tr>
<td>37</td>
<td>28</td>
<td>Employee.viewWorkload</td>
<td>view own workload</td>
</tr>
<tr>
<td>52</td>
<td>30</td>
<td>Course.addCourseInstance</td>
<td>modify course information</td>
</tr>
<tr>
<td>53</td>
<td>30</td>
<td>Course.removeCourseInstance</td>
<td>modify course information</td>
</tr>
<tr>
<td>54</td>
<td>30</td>
<td>Course.getCourseInstances (provisional)</td>
<td>modify course information</td>
</tr>
<tr>
<td>54</td>
<td>30</td>
<td>Course.getCourseInstances (definitive)</td>
<td>view course information</td>
</tr>
<tr>
<td>58</td>
<td>31</td>
<td>Employee.viewCourseInstancesAndAssignedEmployeesReport</td>
<td>view own workload</td>
</tr>
<tr>
<td>62</td>
<td>32</td>
<td>Department.addSubdepartment</td>
<td>administrator</td>
</tr>
<tr>
<td>63</td>
<td>32</td>
<td>Department.removeSubdepartment</td>
<td>administrator</td>
</tr>
<tr>
<td>75</td>
<td>33</td>
<td>Subdepartment.addEducationType</td>
<td>modify subdepartment workload</td>
</tr>
<tr>
<td>76</td>
<td>33</td>
<td>Subdepartment.removeEducationType</td>
<td>modify subdepartment workload</td>
</tr>
<tr>
<td>77</td>
<td>33</td>
<td>Subdepartment.addCourse</td>
<td>modify course information</td>
</tr>
<tr>
<td>78</td>
<td>33</td>
<td>Subdepartment.removeCourse</td>
<td>modify course information</td>
</tr>
<tr>
<td>79</td>
<td>33</td>
<td>Subdepartment.addExpertiseGroup</td>
<td>modify subdepartment workload</td>
</tr>
<tr>
<td>80</td>
<td>33</td>
<td>Subdepartment.removeExpertiseGroup</td>
<td>modify subdepartment workload</td>
</tr>
<tr>
<td>81</td>
<td>33</td>
<td>Subdepartment.getUnplannedCapacity</td>
<td>view subdepartment workload</td>
</tr>
<tr>
<td>82</td>
<td>33</td>
<td>Subdepartment.addTask</td>
<td>modify subdepartment workload</td>
</tr>
<tr>
<td>83</td>
<td>33</td>
<td>Subdepartment.removeTask</td>
<td>modify subdepartment workload</td>
</tr>
<tr>
<td>84</td>
<td>33</td>
<td>Subdepartment.makeAssignmentsDefinitive</td>
<td>modify subdepartment workload</td>
</tr>
<tr>
<td>85</td>
<td>34</td>
<td>Subdepartment.makeCourseInstancesDefinitive</td>
<td>modify course information</td>
</tr>
<tr>
<td>110</td>
<td>38</td>
<td>Task.addAssignment</td>
<td>modify subdepartment workload</td>
</tr>
<tr>
<td>111</td>
<td>38</td>
<td>Task.removeAssignment</td>
<td>modify subdepartment workload</td>
</tr>
<tr>
<td>112</td>
<td>38</td>
<td>Task.getAssignments (provisional)</td>
<td>modify subdepartment workload</td>
</tr>
<tr>
<td>112</td>
<td>38</td>
<td>Task.getAssignments (definitive)</td>
<td>view subdepartment workload</td>
</tr>
<tr>
<td>124</td>
<td>40</td>
<td>Hire.getAssignments (provisional)</td>
<td>modify subdepartment workload</td>
</tr>
<tr>
<td>124</td>
<td>40</td>
<td>Hire.getAssignments (definitive)</td>
<td>view subdepartment workload</td>
</tr>
<tr>
<td>139</td>
<td>42</td>
<td>SystemYear.addPeriod</td>
<td>administrator</td>
</tr>
<tr>
<td>140</td>
<td>42</td>
<td>SystemYear.removePeriod</td>
<td>administrator</td>
</tr>
<tr>
<td>141</td>
<td>42</td>
<td>SystemYear.addStudyProgram</td>
<td>modify subdepartment course information</td>
</tr>
<tr>
<td>142</td>
<td>42</td>
<td>SystemYear.removeStudyProgram</td>
<td>modify subdepartment course information</td>
</tr>
<tr>
<td>143</td>
<td>42</td>
<td>SystemYear.close</td>
<td>modify subdepartment course information</td>
</tr>
<tr>
<td>147</td>
<td>42</td>
<td>Period.addSubPeriod</td>
<td>modify subdepartment workload</td>
</tr>
<tr>
<td>148</td>
<td>42</td>
<td>Period.removeSubPeriod</td>
<td>modify subdepartment workload</td>
</tr>
<tr>
<td>154</td>
<td>43</td>
<td>SubPeriod.getCourseInstances (provisional)</td>
<td>modify course information</td>
</tr>
<tr>
<td>154</td>
<td>43</td>
<td>SubPeriod.getCourseInstances (definitive)</td>
<td>view course information</td>
</tr>
<tr>
<td>158</td>
<td>44</td>
<td>StudyProgram.addTargetGroup</td>
<td>modify subdepartment course information</td>
</tr>
<tr>
<td>159</td>
<td>44</td>
<td>StudyProgram.removeTargetGroup</td>
<td>modify subdepartment course information</td>
</tr>
<tr>
<td>163</td>
<td>44</td>
<td>TargetGroup.getCourseInstances (provisional)</td>
<td>modify course information</td>
</tr>
<tr>
<td>163</td>
<td>44</td>
<td>TargetGroup.getCourseInstances (definitive)</td>
<td>view course information</td>
</tr>
<tr>
<td>168</td>
<td>45</td>
<td>System.addEmployee</td>
<td>modify subdepartment workload</td>
</tr>
<tr>
<td>169</td>
<td>45</td>
<td>System.removeEmployee</td>
<td>modify subdepartment workload</td>
</tr>
<tr>
<td>171</td>
<td>46</td>
<td>ExternalConnection.set_enabled</td>
<td>administrator</td>
</tr>
</tbody>
</table>
Chapter 4

Reports list

This chapter lists all reports which will be generated by the system. The Moscow priorities can be found in section 3.

<table>
<thead>
<tr>
<th>SCR191</th>
<th>must have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users that can view a report, can print that report.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SCR192</th>
<th>should have</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users that can view a report, can print that report.</td>
<td></td>
</tr>
</tbody>
</table>

4.1 Tasks related to own course instances

This report shows the tasks related to a specific course instance, along with the employees assigned. It does not show the number of hours assigned to each employee, as in 4.7. This report is used in report 4.2 and 4.3.

<table>
<thead>
<tr>
<th>1</th>
<th>Year: systemYear.friendlyName</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Tasks related to courses related to employee.name</td>
</tr>
<tr>
<td>3</td>
<td>Course: ci.name (responsible:ci.responsibles.name)</td>
</tr>
<tr>
<td>4</td>
<td>Task: t.name, t.educationType.name, t.assignments.hire.employee.name</td>
</tr>
</tbody>
</table>

1 systemYear.friendlyName is a friendly name for the year for which the report is generated.

2 employee.name is the employee for which the report is generated.

3, 4 Lines 3 and 4 are repeated for each course instance for which that employee is responsible, to which that employee has hours assigned, or both.

Lines 3 and 4 are repeated for each ci ∈ employee.responsibleFor

∪

t.courseInstance |
t ∈ employee.hires.getAssignments().tasks
∧ ci ∈ t.courseInstances
∧ t.subPeriod.period.year = systemYear
∧ t : EducationTask

3 ci.responsibles.name is a list of the responsible employees of that course instance.

4 Line 4 is repeated for each task of that course instance.
Line 4 is repeated for each 

\[ t \in \{ t \mid t \in ci.educationTasks \land t.subPeriod.period.year = systemYear \} \].

4 t.assignments.hire.employee.name is a list of the employees assigned to that task.

4.2 Tasks related to course instances of expertise group

This report shows the tasks related to all course instances of an expertise group, along with the employees assigned.
This report is used in report 4.3.

<table>
<thead>
<tr>
<th>1</th>
<th>Year: systemYear.friendlyName</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Tasks related to courses related to expertiseGroup.name</td>
</tr>
<tr>
<td>3</td>
<td>Course: ci.name (responsible:ci.responsibles.name)</td>
</tr>
<tr>
<td>4</td>
<td>Task: t.name, t.educationType.name, t.assignments.hire.employee.name</td>
</tr>
</tbody>
</table>

1 systemYear.friendlyName is the year for which the report is generated.
2 expertiseGroup.name is the expertise group for which the report is generated.

3,4 Lines 3 and 4 are repeated for each course instance for which any employee of this expertise group is responsible, to which any employee of this expertise group has hours assigned, or both.
Lines 3 and 4 are repeated for each 

\[ ci \in \text{expertiseGroup.hires.employee.responsibleFor} \]
∪

\[ \{ t.courseInstance \mid t \in \text{expertiseGroup.hires.getAssignments().tasks} \land ci \in t.courseInstances \land t.subPeriod.period.year = systemYear \} \].

3 ci.responsibles.name is a list of the responsible employees of that course instance.

4 Line 4 is repeated for each task of that course instance.
Line 4 is repeated for each 

\[ t \in \{ t \mid t \in ci.educationTasks \land t.subPeriod.period.year = systemYear \} \].

4 t.assignments.hire.employee.name is a list of the employees assigned to that task.
4.3 Tasks related to course instances of subdepartment

This report shows the tasks related to all course instances of a subdepartment, along with the employees assigned.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Year: <code>systemYear.friendlyName</code></td>
</tr>
<tr>
<td>2</td>
<td>Tasks related to courses related to <code>self.name</code></td>
</tr>
<tr>
<td>3</td>
<td>Report 4.2 for <code>eg</code></td>
</tr>
</tbody>
</table>

1. `systemYear.friendlyName` is the year for which the report is generated.
2. `self.name` is the subdepartment for which the report is generated.
3. Line 3 is repeated for each expertise group of this subdepartment. Line 3 is repeated for each `eg ∈ self.expertiseGroups`.

4.4 Workload for a single employee

This report shows the workload of a single employee and provides totals for planned and unplanned hours, grouped by task type.

This report is used in report 4.5 and 4.6.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Year: <code>systemYear.friendlyName</code></td>
</tr>
<tr>
<td>2</td>
<td>Workload for <code>self.name</code></td>
</tr>
<tr>
<td>3</td>
<td>Task type: <code>tt</code></td>
</tr>
<tr>
<td>4</td>
<td>Course: <code>ci.name (responsible:ci.responsibles.name)</code></td>
</tr>
<tr>
<td>5</td>
<td>Task: <code>t.name, t.educationType.name</code></td>
</tr>
<tr>
<td>6</td>
<td>Assignment: <code>a.hours, a.externallyFunded</code></td>
</tr>
<tr>
<td>7</td>
<td>Planned hours for <code>tt</code>: <code>PlannedHours</code></td>
</tr>
<tr>
<td>8</td>
<td>Unplanned hours for <code>tt</code>: <code>UnplannedHours</code></td>
</tr>
<tr>
<td>9</td>
<td>Total planned hours: <code>TotalPlannedHours</code></td>
</tr>
<tr>
<td>10</td>
<td>Total unplanned hours: <code>TotalUnplannedHours</code></td>
</tr>
</tbody>
</table>

1. `systemYear.friendlyName` is the year for which the report is generated.
2. `self.name` is the employee for which the report is generated.
3-8 Lines 3 through 8 are repeated for each task type to which that employee has hours assigned.

Lines 3 through 8 are repeated for each `tt ∈ {t.type | t ∈ self.hires.getAssignments().tasks ∧ t.subPeriod.period.year = systemYear}`.

3. `tt` is the task type.
4-6 If that task type is "education", lines 4 through 6 are repeated for each course instance to
which that employee has hours assigned. Otherwise, line 4 is omitted. If \( tt.type = "education" \), lines 4 through 6 are repeated for each \( ci \in \{ t.courseInstance | t \in self.hires.getAssignments().tasks ∧ t.type = education ∧ t.subPeriod.period.year = systemYear \} \). Otherwise, line 4 is omitted.

4. \( ci.responsibles.name \) is a list of the responsible employees of that course instance.

5,6 If that task type is "education", lines 5 and 6 are repeated for each task of that course instance. Otherwise, lines 5 and 6 are repeated for each task of that type assigned to this employee. If \( tt.type = "education" \), lines 5 and 6 are repeated for each \( t \in \{ t | t \in ci.educationTasks ∧ t.subPeriod.period.year = systemYear \} \). Otherwise, lines 5 and 6 are repeated for each \( t \in \{ t | t \in self.hires.getAssignments().tasks ∧ t.type = tt ∧ t.subPeriod.period.year = systemYear \} \).

5 If that task type is not "education", \( t.educationType.name \) is omitted. If \( tt.type \neq "education" \), \( t.educationType.name \) is omitted.

6 Line 6 is repeated for each assignment of that task. Line 6 is repeated for each \( a \in t.assignments \).

7 \( PlannedHours \) is the sum of all hours assigned to that employee, of that task type.
\[
PlannedHours = (\sum a.hours | a.task.type = tt ∧ a.task.subPeriod.period.year = systemYear ∧ a.hire.employee = self).
\]

8 \( UnplannedHours \) is the number of hours to be planned for that task type minus \( Workload \).
\[
UnplannedHours = (\sum h.fte * h.ratio(tt) | h \in self.hires) * currentYear.fteAmount - Workload.
\]

9 \( TotalPlannedHours \) is the sum of all hours assigned to that employee, excluding absences.
\[
TotalPlannedHours = (\sum a.hours | a.task.type \neq absence ∧ a.task.subPeriod.period.year = systemYear ∧ a.hire.employee = self).
\]

10 \( TotalUnplannedHours \) is the total of unplanned hours.
\[
TotalUnplannedHours = (\sum h.fte | h \in self.hires) * systemYear.fteAmount - TotalWorkload.
\]

### 4.5 Workload of all employees of an expertise group

This report shows the workload of all employees of an expertise group, and provides totals for planned and unplanned hours per task type and per employee. This report is used in report 4.6.

| Year: systemYear.friendlyName | Workload for all employees of expertiseGroup.name | e.viewSingleReport() |
4.6 Workload of all employees of a subdepartment

This report shows the workload of all employees of a subdepartment, and provides totals for planned and unplanned hours per task type, per expertise group and per employee.
Unplanned hours for \( tt \): \( UnplannedHours \)

Total planned hours: \( TotalPlannedHours \)

Total unplanned hours: \( TotalUnplannedHours \)

1. \( systemYear.friendlyName \) is the year for which the report is generated.
2. \( subdepartment.name \) is the subdepartment for which the report is generated.
3. Line 3 is repeated for all expertise groups of that subdepartment.

4-6 Lines 4 through 6 are repeated for each task type to which any employee of that subdepartment has hours assigned.

4. \( tt \) is the task type.

5. \( PlannedHours \) is the sum of all hours assigned to all employees, of that task type.

\[
PlannedHours = \left( \sum a.\text{hours} \mid \text{a.task.type} = tt \land \text{a.task.subPeriod.period.year} = systemYear \land \text{a.hire.expertiseGroup.subdepartment} = \text{subdepartment} \right).
\]

6. \( UnplannedHours \) is the number of hours to be planned for that task type minus \( Workload \).

\[
UnplannedHours = \left( \sum h.\text{fte} \ast h.\text{ratio}(tt) \mid \text{h} \in \text{subdepartment.hires} \right) \ast \text{currentYear.fteAmount} - \text{Workload}.
\]

7. \( TotalPlannedHours \) is the sum of all hours assigned to all employees, excluding absences.

\[
TotalPlannedHours = \left( \sum a.\text{hours} \mid \text{a.task.type} \neq \text{absence} \land \text{a.task.subPeriod.period.year} = systemYear \land \text{a.hire.expertiseGroup.subdepartment} = \text{subdepartment} \right).
\]

8. \( TotalUnplannedHours \) is the total of unplanned hours.

\[
TotalUnplannedHours = \left( \sum h.\text{fte} \mid \text{h} \in \text{subdepartment.expertiseGroups.hires} \right) \ast \text{systemYear.fteAmount} - \text{TotalWorkload}.
\]

### 4.7 Education workload for a course instance

This report shows the education workload for a specific course instance. It shows the number of hours of education tasks assigned to each employee.

This report is used in reports 4.8, 4.9, and 4.10.

This report needs both a course instance and a filter (subdepartment) as input. The course instance is called \( self \). The subdepartment is called \( sd \). How these values are passed onto the report is left as implementation detail.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Year: ( systemYear.friendlyName ), subdepartment: ( sd )</td>
</tr>
<tr>
<td>2</td>
<td>Education workload for ( self.name ) (external: ( self.course.givenExternally ))</td>
</tr>
<tr>
<td>3</td>
<td>Responsible: ( self.responsibles.name )</td>
</tr>
</tbody>
</table>
4 Subperiods: self.subperiods
5 Task: t.name
6 Assignment: a.hours, a.externallyFunded, a.hire.employee.name
7 Total education workload: TotalEducationWorkload

1 systemYear.friendlyName is the year for which the report is generated.
2 sd is the subdepartment for which the report is generated.
3 self.name is the course instance for which the report is generated.
4 self.course.givenExternally shows whether the course is given externally or not.\[1\]
5 self.responsibles.name is a list of all responsible employees for this course instance.
6 self.subperiods is a list of all subperiods this course instance is given.

5,6 Lines 5 and 6 are repeated for each task of that course instance.

Lines 5 and 6 are repeated for each \( t \in \{ t \mid t \in self.tasks \land sd \in t.assignments.hires.expertiseGroups.subdepartment \} \)

6 Line 6 is repeated for each assignment of that task.

Line 6 is repeated for each \( a \in t.assignments \).

7 TotalEducationWorkload is the sum of all hours assigned to this course instance.

\[
TotalEducationWorkload = (\sum a.hours \mid a.task.subPeriod.period.year = systemYear \land a.task.courseInstance = self).
\]

4.8 Education workload for a target group

This report shows the education workload for a target group. It shows the number of hours of education tasks assigned to each employee, grouped by course instance.

This report is used in reports [4.9] and [4.10]

This report needs both a target group and a filter (subdepartment) as input. The target group is called \( self \). The subdepartment is called \( sd \). How these values are passed onto the report is left as implementation detail.

1 Year: systemYear.friendlyName, subdepartment: sd
2 Education workload for self.name
3 Report 4.7 (with filter sd) for ci
4 Total education planned hours: TotalEducationPlannedHours

1 systemYear.friendlyName is the year for which the report is generated.

\[1\]Courses given externally are those that are given to an audience primarily consisting of students from other subdepartments.
1. *sd* is the subdepartment for which the report is generated.

2. *self.name* is the target group for which the report is generated.

3. Line 3 is repeated for each course instance of that target group. Line 3 is repeated for each *ci ∈ self.courseInstances*.

4. *TotalEducationPlannedHours* is the sum of all hours assigned to this target group. 
   
   \[
   \text{TotalEducationPlannedHours} = \left( \sum a.\text{hours} \mid a.\text{task}.\text{subPeriod}.\text{period}.\text{year} = \text{systemYear} \wedge \text{self} \in a.\text{task}.\text{courseInstance}.\text{targetGroups}.\text{studyProgram} \right).
   \]

### 4.9 Education workload for a study program

This report shows the education workload for a study program. It shows the number of hours of education tasks assigned to each employee, grouped by target group and course instance. This report is used in report [4.10].

This report needs both a study program and a filter (subdepartment) as input. The study program is called *self*. The subdepartment is called *sd*. How these values are passed onto the report is left as implementation detail.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Year: <em>systemYear.friendlyName</em>, subdepartment: <em>sd</em></td>
</tr>
<tr>
<td>2</td>
<td>Education workload for <em>self.name</em></td>
</tr>
<tr>
<td>3</td>
<td>Report <a href="#">4.8</a> (with filter <em>sd</em>) for <em>tg</em></td>
</tr>
<tr>
<td>4</td>
<td>Total education planned hours: <em>TotalEducationPlannedHours</em></td>
</tr>
</tbody>
</table>

1. *systemYear.friendlyName* is the year for which the report is generated.

2. *sd* is the subdepartment for which the report is generated.

3. *self.name* is the study program for which the report is generated.

4. *TotalEducationPlannedHours* is the sum of all hours assigned to this study program. 
   
   \[
   \text{TotalEducationPlannedHours} = \left( \sum a.\text{hours} \mid a.\text{task}.\text{subPeriod}.\text{period}.\text{year} = \text{systemYear} \wedge \text{self} \in a.\text{task}.\text{courseInstance}.\text{targetGroups}.\text{studyProgram} \right).
   \]

### 4.10 Education workload for all study programs

This report shows the education workload for all study programs. It shows the number of hours of education tasks assigned to each employee, grouped by target group, course instance and study program.

This report needs both a year and a filter (subdepartment) as input. The year is called *self*. The
subdepartment is called \( sd \). How these values are passed onto the report is left as implementation detail.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Education workload for ( \text{systemYear.friendlyName} ), subdepartment: ( sd )</td>
</tr>
<tr>
<td>2</td>
<td>Report 4.9 (with filter ( sd )) for ( sp )</td>
</tr>
<tr>
<td>3</td>
<td>Total education planned hours: ( \text{TotalEducationPlannedHours} )</td>
</tr>
</tbody>
</table>

1 \( \text{systemYear.friendlyName} \) is the year for which the report is generated.

1 \( sd \) is the subdepartment for which the report is generated.

2 Line 2 is repeated for each study program of that year.

Line 3 is repeated for each \( sp \in \text{self.studyPrograms} \).

3 \( \text{TotalEducationPlannedHours} \) is the sum of all hours assigned to all study programs.

\[
\text{TotalEducationPlannedHours} = \left( \sum \text{a.hours} \mid
\begin{align*}
\text{a.task.subPeriod.period.year} &= \text{systemYear} \\
\text{self} &\in \text{a.task.type} = \text{“education”} \\
\text{a.task.subperiod.period.systemYear} &= \text{self}
\end{align*}
\right).
\]

### 4.11 Capacity of an expertise group

This report shows the number of hours available for planning for the employees of an expertise group. It shows the number of fte multiplied by the number of hours in one fte, grouped by the position of that employee and broken down into the different task types. This report is used in report 4.12.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Year: ( \text{systemYear.friendlyName} )</td>
</tr>
<tr>
<td>2</td>
<td>Hours for ( \text{self.name} )</td>
</tr>
<tr>
<td>3</td>
<td>Position: ( \text{position} )</td>
</tr>
<tr>
<td>4</td>
<td>Capacity for ( \text{tt} ): ( \text{capacity} )</td>
</tr>
<tr>
<td>5</td>
<td>Capacity for ( \text{position} ): ( \text{capacityPosition} )</td>
</tr>
<tr>
<td>6</td>
<td>Total capacity: ( \text{totalCapacity} )</td>
</tr>
</tbody>
</table>

1 \( \text{systemYear.friendlyName} \) is the year for which the report is generated.

2 \( \text{self.name} \) is the expertise group for which the report is generated.

3-5 Lines 3 through 5 are repeated for each position of that expertise group.

Line 3 through 5 are repeated for each \( \text{position} \in \{h.position \mid h \in \text{self.hires}\} \).

3 \( \text{position} \) is the position.

4 Line 4 is repeated for each task type any employee of this position of this expertise group has a task assigned to.
Line 4 is repeated for each \( tt \in \{ a.task.type \mid a \in self.hires.assignments \land a.hire.position = position \} \).

4 capacity is the capacity of all employees in that position for that task type.
\[
\text{capacity} = \left( \prod h.ratio(tt) \mid h \in self.hires \land h.position = position \right) \ast \text{systemYear.fteAmount}
\]

5 totalCapacity is the capacity of all employees of that expertise group.
\[
\text{totalCapacity} = \left( \prod h.fte \mid h \in self.hires \right) \ast \text{systemYear.fteAmount}.
\]

4.12 Capacity of a subdepartment

This report shows the number of hours available for planning for the employees of a subdepartment. It shows the number of fte multiplied by the number of hours in one fte, grouped by subdepartment and the position of that employee, and broken down into the different task types.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Year: systemYear.friendlyName</td>
</tr>
<tr>
<td>2</td>
<td>Hours for self.name</td>
</tr>
<tr>
<td>3</td>
<td>Report 4.11 for eg</td>
</tr>
<tr>
<td>4</td>
<td>Total capacity: totalCapacity</td>
</tr>
</tbody>
</table>

1 systemYear.friendlyName is the year for which the report is generated.
2 self.name is the subdepartment for which the report is generated.
3 Line 3 is repeated for each expertise group of that subdepartment.
   Line 3 is repeated for each egself.expertiseGroups.
4 TotalCapacity is the sum of the capacity of all employees of that subdepartment.
\[
\text{TotalCapacity} = \left( \prod h.fte \mid h \in self.expertiseGroups.hires \right) \ast \text{systemYear.fteAmount}
\]

4.13 Budget for a subdepartment

This report shows the number of planned hours, grouped by the position of the employee assigned to the tasks. For education tasks, the numbers are further broken down into study phase and internal and external education.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Year: systemYear.friendlyName</td>
</tr>
<tr>
<td>2</td>
<td>Budget for self.name</td>
</tr>
<tr>
<td>3</td>
<td>Position: position</td>
</tr>
<tr>
<td>4</td>
<td>Task type: tt</td>
</tr>
<tr>
<td>5</td>
<td>External: external</td>
</tr>
<tr>
<td>6</td>
<td>Study phase: studyPhase</td>
</tr>
<tr>
<td>7</td>
<td>Planned hours: plannedHours</td>
</tr>
</tbody>
</table>
Planned hours for \( tt \): \( \text{plannedHours}_{tt} \)

Planned hours for position: \( \text{plannedHours}_{\text{Position}} \)

Total planned hours: \( \text{totalPlannedHours} \)

1. \( \text{systemYear.friendlyName} \) is the year for which the report is generated.

2. \( \text{self.name} \) is the subdepartment for which the report is generated.

3-9 Lines 3 through 9 are repeated for each position of any employee of that subdepartment. Lines 3 through 9 are repeated for each \( \text{position} \in \{ \text{h.position} \mid \text{h} \in \text{self.expertiseGroups.hires} \} \).

3. \( \text{position} \) is the position.

4-8 Lines 4 through 8 are repeated for each task type assigned to any employee of that subdepartment of that position, excluding absences. Lines 4 through 8 are repeated for each \( \text{tt} \in \{ \text{a.task.type} \mid \text{a} \in \text{h.assignments} \land \text{h} \in \text{self.expertiseGroups.hires} \land \text{h.position} = \text{pos} \}\)\{“absence”\}.

4. \( \text{tt} \) is the task type.

5-7 If that task type is “education”, lines 5 through 7 are repeated for internal and external courses. Otherwise, line 5 is omitted. If \( \text{tt} = \) “education”, lines 5 through 7 are repeated for \( \text{external} = \text{true} \) and \( \text{external} = \text{false} \).

6,7 If that task type is “education”, lines 6 through 7 are repeated for each study phase of courses. Otherwise, line 6 is omitted. If that task type is “education”, lines 6 through 7 are repeated for each \( \text{studyPhase} \in \{ \text{ci.studyPhase} \mid \text{ci} \in \text{self.expertiseGroups.hires.assignments.tasks.courseInstances} \land \text{ci.external} = \text{external} \} \). Otherwise, line 6 is omitted.

7. If that task type is “education”, \( \text{plannedHours} \) is the number of hours planned for all courses of that that external type and study phase. Otherwise, line 7 is omitted. If that task type is “education”, \( \text{plannedHours} = (\sum \text{a.hours} \mid \text{ci} \in \text{self.expertiseGroups.hires.assignments.tasks.courseInstances} \land \text{ci.external} = \text{external} \land \text{ci.studyPhase} = \text{studyPhase} \land \text{a} \in \text{ci.educationTasks.assignments} \land \text{a.hire.position} = \text{position}) \). Otherwise, line 7 is omitted.

8. \( \text{plannedHour}_{tt} \) is the number of hours planned for that task type and that position. \( \text{plannedHour}_{tt} = (\sum \text{a.hours} \mid \text{a} \in \text{self.expertiseGroups.hires.assignments} \land \text{a.type} = \text{tt} \land \text{a.hire.position} = \text{position}) \)

9. \( \text{plannedHours}_{\text{Position}} \) is the number of hours planned for that position. \( \text{plannedHours}_{\text{Position}} = (\sum \text{a.hours} \mid \text{a} \in \text{self.expertiseGroups.hires.assignments} \land \text{a.hire.position} = \text{position}) \)

10. \( \text{totalPlannedHours} \) is the number of hours planned for this subdepartment. \( \text{totalPlannedHours} = (\sum \text{a.hours} \mid \text{a} \in \text{self.expertiseGroups.hires.assignments}) \)
### 4.14 Research hours for an expertise group

This report is used in report 4.15.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Year: <code>systemYear.friendlyName</code></td>
</tr>
<tr>
<td>2</td>
<td>Research report for <code>self.name</code></td>
</tr>
<tr>
<td>3</td>
<td>Task: <code>t.name</code></td>
</tr>
<tr>
<td>4</td>
<td>Assignment: <code>a.hours</code>, <code>a.externallyFunded</code>, <code>a.hire.employee.name</code></td>
</tr>
<tr>
<td>5</td>
<td>Research hours for <code>t.name</code>: <code>TaskResearchHours</code></td>
</tr>
<tr>
<td>6</td>
<td>Total research hours: <code>TotalResearchHours</code></td>
</tr>
</tbody>
</table>

1. `systemYear.friendlyName` is the year for which the report is generated.
2. `self.name` is the expertise group for which the report is generated.
3-5 Lines 3 through 5 are repeated for each research task to which any employee of this expertise group has hours assigned.

Lines 3 through 5 are repeated for each `t ∈ {t | t ∈ self.hires.assignments.task ∧ t.type = "research" ∧ t.subperiod.period.year = systemYear}`

4. Line 4 is repeated for each assignment of that task.
   Line 4 is repeated for each `a ∈ t.assignments`.
5. `TaskResearchHours` is the number of research hours assigned to that task.
   \[
   TaskResearchHours = \left( \sum a.hours \mid a ∈ t.assignments \right)
   \]
6. `TotalResearchHours` is the number of research hours assigned to this expertise group.
   \[
   TotalResearchHours = \left( \sum a.hours \mid a ∈ self.hires.assignments ∧ a.task.type = "research" ∧ a.task.subperiod.period.year = systemYear \right)
   \]

### 4.15 Research hours for a subdepartment

This report shows the planned hours for research tasks. It shows the hours and employees assigned to each task.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Year: <code>systemYear.friendlyName</code></td>
</tr>
<tr>
<td>2</td>
<td>Research report for <code>self.name</code></td>
</tr>
<tr>
<td>3</td>
<td>Report 4.14 for <code>eg</code></td>
</tr>
<tr>
<td>4</td>
<td>Total research hours: <code>TotalResearchHours</code></td>
</tr>
</tbody>
</table>

1. `systemYear.friendlyName` is the year for which the report is generated.
2. `self.name` is the subdepartment for which the report is generated.
3. Line 3 is repeated for each expertise group of that subdepartment.
Line 3 is repeated for each \( eg \in self.expertiseGroups \).

4 TotalResearchHours is the total number of research hours assigned to this subdepartment. 
TotalResearchHours = \( (\sum a.hours \mid a \in self.expertiseGroups.hires.assignments \land a.task.subPeriod.period.year = systemYear \land a.task.type = \text{"research"}) \).

### 4.16 Management hours for an expertise group

This report is used in report 4.17.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Year: systemYear.friendlyName</td>
</tr>
<tr>
<td>2</td>
<td>Management report for self.name</td>
</tr>
<tr>
<td>3</td>
<td>Task: t.name</td>
</tr>
<tr>
<td>4</td>
<td>Assignment: a.hours, a.externallyFunded, a.hire.employee.name</td>
</tr>
<tr>
<td>5</td>
<td>Management hours for t.name: TaskManagementHours</td>
</tr>
<tr>
<td>6</td>
<td>Total management hours: TotalManagementHours</td>
</tr>
</tbody>
</table>

1 systemYear.friendlyName is the year for which the report is generated.
2 self.name is the expertise group for which the report is generated.

3-5 Lines 3 through 5 are repeated for each management task to which any employee of this expertise group has hours assigned.

Lines 3 through 5 are repeated for each \( t \in \{t \mid t \in self.hires.assignments.task \land t.type = \text{"management"} \land t.subperiod.period.year = systemYear\} \)

4 Line 4 is repeated for each assignment of that task.
Line 4 is repeated for each \( a \in t.assignments \).

5 TaskManagementHours is the number of management hours assigned to that task.
TaskManagementHours = \( (\sum a.hours \mid a \in t.assignments) \)

6 TotalManagementHours is the number of management hours assigned to this expertise group.
TotalManagementHours = \( (\sum a.hours \mid a \in self.hires.assignments \land a.task.type = \text{"management"} \land a.task.subperiod.period.year = systemYear) \)

### 4.17 Management hours for a subdepartment

This report shows the hours planned in management tasks. It shows the hours and employee assigned to each task.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Year: systemYear.friendlyName</td>
</tr>
<tr>
<td>2</td>
<td>Management report for self.name</td>
</tr>
</tbody>
</table>
Report 4.16 for eg

Total management hours: TotalManagementHours

1 systemYear.friendlyName is the year for which the report is generated.
2 self.name is the subdepartment for which the report is generated.
3 Line 3 is repeated for each expertise group of that subdepartment.
   Line 3 is repeated for each eg ∈ self.expertiseGroups.
4 TotalManagementHours is the total number of management hours assigned to this sub-department.
   TotalManagementHours = (∑ a.hours | a ∈ self.expertiseGroups.hires.assignments∧
   a.task.subPeriod.period.year = systemYear ∧ a.task.type = "management").

4.18 Course instances and assigned employees for an education type

This report shows the course instances and assigned employees for those education tasks of a specific type. For example, a list of all lectures and assigned employees.

1 Year: systemYear.friendlyName
2 Courses and employees for courses with self.name
3 Course: ci.name (responsible: ci.responsibles.name)
4 Task: t.name, t.assignments.hire.employee.name

1 systemYear.friendlyName is the year for which the report is generated.
2 self.name is the education type for which the report is generated.
3,4 Lines 3 and 4 are repeated for each course instance for which there exists an education task of this type.
   Lines 3 and 4 are repeated for each ci ∈ self.educationTasks.courseInstances.
3 ci.responsibles.name is a list of responsible employees of that course instance.
4 Line 4 is repeated for each task of that type of that course instance.
   Line 4 is repeated for each t ∈ {t | t ∈ ci.tasks ∧ ci.educationType = self}.
Chapter 5

Prototype

This chapter describes in broad terms what the graphical user interface of QIS will resemble when it is completed. Screens that are left out are simple forms for adding or editing information, which are quite trivial. The user interface to match the prior software requirements is explained here. The screenshots presented here are meant to provide the user with a general impression of the interface. They are indicative of functionality only, details are subject to change.

5.1 Authentication

Login screen

Figure 5.1: Login screen

Figure 5.1 shows the log-in screen to which users authenticate themselves. Afterwards, they will be greeted by the screen from Figure 5.2.
5.2 Employees

Employee workload

Figure 5.2: Single employee information screen, workload shown

Figure 5.2 shows the detailed workload of an employee, in addition to other employee information which is relevant to them. The edit details button allows the user to change some preferences. With sufficient permissions, other attributes of that employee may be changed.

This screen also illustrates the overall structure of the user interface through the top menu. These screens correspond to different sections of this explanation. A regular employee, one that is only authorized to view their own workload, will not be shown any navigation.

Employee list

Figure 5.3 shows the employees screen. This is the main overview of all employees of a subdepartment. This list can be filtered and sorted by first and last name. On the top we find buttons related to global actions of this screen. To the left of the list we find several buttons for each employee which allow the user to perform an operation on that object quickly. These buttons are added for the most common operations.
Figure 5.3: Employee list

Buttons

**Expand employees** Expands all employees, as if pressing the “expand employee” action button for every employee in the list. If any or all are already expanded, this button changes into a collapse employees button, collapsing all expanded employees.

**Expand hires** Expands the hires of all expanded employees. See the information on expanded employees for figure 5.4. If any or all are already expanded, this button changes into a collapse hires button, collapsing all expanded hires.

**Add employee** Directs the user to a screen where a new employee can be added.

**Action button 1: expand/collapse employee** Expands this employee, showing the associated hires, as in figure 5.4. If it is already expanded, this button collapses it again.

**Action button 2: view employee workload** Directs the user to the workload screen for this employee, as in figure 5.2.

**Action button 3: edit employee** Directs the user to a screen where the employee information can be edited, or it can be removed.

Employee expanded

Figure 5.4 is the same screen as in figure 5.3 but with the hires expanded. All information about that hire is displayed in that row, in particular the hours that employee still has available. These
hours are shown in total and for each category: Eduction tasks, Management projects, Research projects and Other tasks. Additional buttons appear. Note that some common actions such as filling up the research hours to match the hours that remain can become action buttons as well.

**Buttons**

**Collapse employees** Undoes the operation of expanding employees.

**Employee action button 4: add hire** Directs the user to a screen where a new hire can be added.

**Hire action button 1: expand/collapse hire** Expands this hire, showing all associated assignments, as in figure 5.5. If it is already expanded, this button collapses it again.

**Hire action button 2: edit hire** Directs the user to a screen where that hire information can be edited.

**Hire action button 3: remove hire** Directs the user to a deletion confirmation screen.

**Hire expanded**

Figure 5.5 is the same screen as in Figure 5.4 but with a single employee’s hire expanded. This shows all assignments grouped by category and the buttons to add, modify and remove them. It also displays information related to the calculation of the hours.
Figure 5.5: Employees list, tasks shown

Buttons

Assign (task type) Directs the user to a screen where new tasks can be assigned. The tasks can be selected from a list and additional information can be entered, such as the number of hours of the assignment.

Assignment action button 1: edit task Directs the user to a screen where the task information can be edited.

Assignment action button 2: remove task Directs the user to a deletion confirmation screen.

5.3 Courses

Course list

Figure 5.6 shows the courses screen. This is the main overview of the courses of a subdepartment. The list can be filtered and sorted by code and name.
Figure 5.6: Course list

Buttons

Expand courses Expand courses, as if pressing the “expand course” button for every course in the list. If any or all are already expanded, this button changes into a collapse courses button, collapsing all expanded courses.

Expand instances Expand course instances of all expanded courses. See the information on expanded courses for figure 5.7. If any or all are already expanded, this button changes into a collapse course instances button, collapsing all expanded course instances.

Expand tasks Expand all tasks of all expanded course instances. See the information on expanded course instances for figure 5.8. If any or all are already expanded, this button changes into a collapse tasks button, collapsing all expanded tasks.

Add course Directs the user to a screen where a new course can be added.

Via target group Directs the user to the list of target groups, from where they can add and edit courses as per Figure 5.15. This assists users who intend to set courses per target group, instead of the other way round.

Action button 1: expand course Expands the course, showing all course instances, as in figure 5.7. If it is already expanded, this button collapses it again.

Action button 2: edit course Directs the user to a screen where the course can be edited.

Action button 3: remove course Directs the user to a confirmation screen.

Course expanded

Figure 5.7 is the same screen as in Figure 5.6, but with a single course expanded. This shows all course instances and the buttons to add, modify and remove them.

Buttons

Add instance Directs the user to a screen where a course instance can be added.
Figure 5.7: Course list, course instances shown

**Instance action button 1: expand course instance** Expands the course, showing all tasks, as in figure 5.8. If it is already expanded, this button collapses it again.

**Instance action button 2: edit course instance** Directs the user to a screen where the course instance can be edited.

**Instance action button 3: remove course instance** Directs the user to a confirmation screen.

**Course instance expanded**

Figure 5.8 is the same screen as in Figure 5.7 but with a single course instance expanded. This shows all tasks and the buttons to add, modify and remove them.

**Buttons**

**Add task** Directs the user to a screen where a task can be added.

**Instance action button 1: expand task** Expands the task, showing all assignments, as in figure 5.9. If it is already expanded, this button collapses it again.

**Instance action button 2: edit task** Directs the user to a screen where the task can be edited.

**Instance action button 3: remove task** Directs the user to a confirmation screen.
Figure 5.8: Course list, tasks shown

Task expanded

Figure 5.9 is the same screen as in Figure 5.8 but with a single task expanded. This shows all assignments and the buttons to add, modify and remove them.

Buttons

Add task Directs the user to a screen where an assignment can be added. The hire can be selected from a list and additional information can be entered, such as the number of hours of the assignment.

Instance action button 1: edit assignment Directs the user to a screen where the assignment can be edited.

Instance action button 2: remove assignment Directs the user to a confirmation screen.
5.4 Tasks

Task list

Figure 5.9: Course list, assignments shown

Figure 5.10: Task list

Figure 5.10 shows the tasks screen. This is the main overview of all tasks of a subdepartment. This list can be filtered and sorted by task name and type.
Buttons

Expand tasks  Expand tasks, as if pressing the “expand task” button for every task in the list. If any or all are already expanded, this button changes into a collapse tasks button, collapsing all expanded tasks.

Add task  Directs the user to a screen where a new task can be added.

Action button 1: expand task  Expands the task, showing all assignments, as in figure 5.11. If it is already expanded, this button collapses it again.

Action button 2: edit task  Directs the user to a screen where the task can be edited.

Action button 3: remove task  Directs the user to a confirmation screen.

Task expanded

![Task list, assignments shown](image)

Figure 5.11: Task list, assignments shown

Figure 5.11 is the same screen as in Figure 5.10, but with a single task expanded. This shows all assignments and the buttons to add, modify and remove them.

Buttons

Add assignment  Directs the user to a screen where a new assignment can be added. The hire can be selected from a list and additional information can be entered, such as the number of hours of the assignment.

Assignment action button 1: edit assignment  Directs the user to a screen where the assignment can be edited.

Assignment action button 2: remove assignment  Directs the user to a confirmation screen.
5.5 Administrative

Figure 5.12 shows the administrative screen. This is where general administrative work of the system can be done. For each administrative object, it is possible to

- View a list of these objects by clicking the object name (as in figure 5.13)
- Add a new object (as in figure 5.14)
- Modify an object (as in figure 5.15)

Figure 5.13 shows a typical administrative object list view. Objects can be mass-selected for deletion, be filtered and sorted by their most common attributes, and a button directing the user
to add a new object is present.

Figure 5.14: Administrative - Add system year

Figure 5.14 shows a typical example of an object add screen. It is possible to set specific attributes and also allow for easy creation of other object which relate to it. In this example, the Add system year form allows for an optional copy from a previous year, and it allows for adding periods within the same operation for ease of use. Add forms of objects throughout the interface have similar functionality.

Submit button 1: Save and add another will allow the user to keep adding administrative objects of this kind by returning to the add form immediately after.

Submit button 2: Save and continue editing will create the object in the database while allowing the user to continue to modify the attributes. This is particularly useful if the user desires to add multiple of the related items. For example, for the add year page the user would press this button after having filled in all 3 periods fields. The periods given at that time would then be stored and additional periods can be added.

Submit button 3: Save will add the new object to the database and return to the list of objects.

Figure 5.15 is a typical example of an edit form. It is very similar to an add form’s functionality, except that it also allows for deletion of the object. Edit forms of objects throughout the interface have similar functionality.

This particular form allows assigning courses directly to target groups. After the courses have been added to the topmost course list and saving, the user could edit the associated hours belonging to the education instance of that course directly. The Save and continue editing button will assist such modification.
5.6 Reports

Figure 5.16 shows the reports list. Clicking on a report directs the user to a printable version of that report. For example, the report “Single employee” is shown in Figure 5.17. For categories of reports of which there are many, a dropdown box allows for selecting which of the specific reports is to be displayed.
**Figure 5.16: Reports list**
**Employee report: Sidorova, Natalia**

### Workload overview

<table>
<thead>
<tr>
<th>Type</th>
<th>Ratio</th>
<th>Available</th>
<th>Assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Education hours</td>
<td>0.2</td>
<td>152</td>
<td>332</td>
</tr>
<tr>
<td>Management hours</td>
<td>0.3</td>
<td>198</td>
<td>0</td>
</tr>
<tr>
<td>Research hours</td>
<td>0.0</td>
<td>250</td>
<td>0</td>
</tr>
<tr>
<td>Other hours</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total hours</td>
<td>1.0</td>
<td>560</td>
<td>332</td>
</tr>
</tbody>
</table>

### Education

- Lecture 23965: 120 hours
- Instruction 21935: 12 hours
- Total: 132 hours

### Leaves

- Pregnancy leave: 500 hours
- Maternity: 500 hours
- Total: 1000 hours

### Vacation (9.6 Fte as Docent)

<table>
<thead>
<tr>
<th>Type</th>
<th>Ratio</th>
<th>Available</th>
<th>Assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Education hours</td>
<td>0.2</td>
<td>132</td>
<td>0</td>
</tr>
<tr>
<td>Management hours</td>
<td>0.3</td>
<td>199</td>
<td>0</td>
</tr>
<tr>
<td>Research hours</td>
<td>0.0</td>
<td>332</td>
<td>0</td>
</tr>
<tr>
<td>Other hours</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total hours</td>
<td>1.0</td>
<td>664</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**Figure 5.17: Employee workload report**
# Requirements traceability matrix

## 6.1 URD to SRD

<table>
<thead>
<tr>
<th>User</th>
<th>Software</th>
<th>User</th>
<th>Software</th>
<th>User</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>41, 65, 90, 100</td>
<td>60</td>
<td>71, 80, 97</td>
<td>119</td>
<td>173</td>
</tr>
<tr>
<td>2</td>
<td>166, 168, 169</td>
<td>61</td>
<td>96</td>
<td>120</td>
<td>84, 85, 170</td>
</tr>
<tr>
<td>3</td>
<td>24, 35, 167</td>
<td>62</td>
<td>37</td>
<td>121</td>
<td>84, 85, 181</td>
</tr>
<tr>
<td>4</td>
<td>41, 65, 90, 100</td>
<td>63</td>
<td>37, 101</td>
<td>122</td>
<td>170</td>
</tr>
<tr>
<td>5</td>
<td>29, 40, 41, 65, 90, 100</td>
<td>64</td>
<td>37, 101</td>
<td>123</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>41, 65, 90, 100, 133, 134, 135</td>
<td>65</td>
<td>37, 101</td>
<td>124</td>
<td>26</td>
</tr>
<tr>
<td>7</td>
<td>135, 143</td>
<td>66</td>
<td>102</td>
<td>125</td>
<td>182</td>
</tr>
<tr>
<td>8</td>
<td>180</td>
<td>67</td>
<td>60, 62, 68</td>
<td>126</td>
<td>183</td>
</tr>
<tr>
<td>9</td>
<td>40, 166, 168</td>
<td>68</td>
<td>60, 63, 66, 68</td>
<td>127</td>
<td>164</td>
</tr>
<tr>
<td>10</td>
<td>40, 166, 169</td>
<td>69</td>
<td>60, 68, 113</td>
<td>128</td>
<td>39, 64, 89, 99</td>
</tr>
<tr>
<td>11</td>
<td>33, 40</td>
<td>70</td>
<td>37, 88, 101</td>
<td>129</td>
<td>23</td>
</tr>
<tr>
<td>12</td>
<td>34, 40</td>
<td>71</td>
<td>91, 102</td>
<td>130</td>
<td>23</td>
</tr>
<tr>
<td>13</td>
<td>40, 132</td>
<td>72</td>
<td>81</td>
<td>131</td>
<td>22</td>
</tr>
<tr>
<td>14</td>
<td>49, 69, 77</td>
<td>73</td>
<td>37, 88, 101</td>
<td>132</td>
<td>22</td>
</tr>
<tr>
<td>15</td>
<td>49, 69, 78</td>
<td>74</td>
<td>101</td>
<td>133</td>
<td>5</td>
</tr>
<tr>
<td>16</td>
<td>42, 43, 44, 45, 46, 47, 48</td>
<td>75</td>
<td>37, 88</td>
<td>134</td>
<td>5</td>
</tr>
<tr>
<td>17</td>
<td>10</td>
<td>76</td>
<td>40, 61, 137</td>
<td>135</td>
<td>4</td>
</tr>
<tr>
<td>18</td>
<td>9, 11, 12, 162</td>
<td>77</td>
<td>40, 61, 137</td>
<td>136</td>
<td>4</td>
</tr>
<tr>
<td>19</td>
<td>8, 13, 14, 30</td>
<td>78</td>
<td>40, 59</td>
<td>137</td>
<td>108</td>
</tr>
<tr>
<td>20</td>
<td>32</td>
<td>79</td>
<td>72, 82, 109</td>
<td>138</td>
<td>108</td>
</tr>
<tr>
<td>21</td>
<td>6, 50, 52, 152</td>
<td>80</td>
<td>72, 83, 109</td>
<td>139</td>
<td>107</td>
</tr>
<tr>
<td>22</td>
<td>6, 50, 53, 152</td>
<td>81</td>
<td>103, 104, 105, 106</td>
<td>140</td>
<td>107</td>
</tr>
<tr>
<td>23</td>
<td>7, 15, 20, 21, 72, 82, 106, 109, 153</td>
<td>82</td>
<td>110, 123, 129, 130</td>
<td>141</td>
<td>22</td>
</tr>
<tr>
<td>24</td>
<td>7, 16, 20, 21, 72, 83, 109, 153</td>
<td>83</td>
<td>111, 125, 126</td>
<td>142</td>
<td>22</td>
</tr>
<tr>
<td>25</td>
<td>17, 18, 20, 21, 103, 104, 105</td>
<td>84</td>
<td>128</td>
<td>143</td>
<td>4</td>
</tr>
<tr>
<td>26</td>
<td>17</td>
<td>85</td>
<td>38</td>
<td>144</td>
<td>4</td>
</tr>
<tr>
<td>27</td>
<td>17, 110</td>
<td>86</td>
<td>72, 82, 106, 109</td>
<td>145</td>
<td>107</td>
</tr>
<tr>
<td>28</td>
<td>118</td>
<td>87</td>
<td>72, 83, 109</td>
<td>146</td>
<td>107</td>
</tr>
<tr>
<td>29</td>
<td>18</td>
<td>88</td>
<td>103, 104, 105</td>
<td>147</td>
<td>57, 73</td>
</tr>
<tr>
<td>30</td>
<td>56, 70, 75</td>
<td>89</td>
<td>110, 111, 123, 129, 130</td>
<td>148</td>
<td>57, 73</td>
</tr>
</tbody>
</table>
6.2 SRD to URD

<table>
<thead>
<tr>
<th>Software</th>
<th>User</th>
<th>Software</th>
<th>User</th>
<th>Software</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>43, 44, 45, 46, 47</td>
<td>65</td>
<td>1, 4, 5, 6</td>
<td>129</td>
<td>82, 89, 94</td>
</tr>
<tr>
<td>2</td>
<td>172</td>
<td>66</td>
<td>68</td>
<td>130</td>
<td>82, 89, 94</td>
</tr>
<tr>
<td>3</td>
<td>172</td>
<td>67</td>
<td>153, 154, 155, 156, 157</td>
<td>131</td>
<td>153, 155, 156, 157</td>
</tr>
<tr>
<td>4</td>
<td>135, 136, 143, 144</td>
<td>68</td>
<td>67, 68, 69</td>
<td>132</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>133, 134</td>
<td>69</td>
<td>14, 15</td>
<td>133</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>21, 22</td>
<td>70</td>
<td>30, 31</td>
<td>134</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>23, 24</td>
<td>71</td>
<td>59, 60</td>
<td>135</td>
<td>6, 7</td>
</tr>
<tr>
<td>8</td>
<td>19</td>
<td>72</td>
<td>23, 24, 79, 80, 86, 87, 91, 92, 102, 103</td>
<td>136</td>
<td>96, 97</td>
</tr>
<tr>
<td>9</td>
<td>18</td>
<td>73</td>
<td>147, 148, 156</td>
<td>137</td>
<td>76, 77</td>
</tr>
<tr>
<td>10</td>
<td>17</td>
<td>74</td>
<td>150, 151, 157</td>
<td>138</td>
<td>33</td>
</tr>
<tr>
<td>11</td>
<td>18</td>
<td>75</td>
<td>30</td>
<td>139</td>
<td>33</td>
</tr>
<tr>
<td>12</td>
<td>18</td>
<td>76</td>
<td>31</td>
<td>140</td>
<td>33</td>
</tr>
<tr>
<td>13</td>
<td>19</td>
<td>77</td>
<td>14</td>
<td>141</td>
<td>96</td>
</tr>
<tr>
<td>14</td>
<td>19</td>
<td>78</td>
<td>15</td>
<td>142</td>
<td>97</td>
</tr>
<tr>
<td>15</td>
<td>23</td>
<td>79</td>
<td>59</td>
<td>143</td>
<td>7</td>
</tr>
<tr>
<td>16</td>
<td>24</td>
<td>80</td>
<td>60</td>
<td>144</td>
<td>34</td>
</tr>
</tbody>
</table>

79
| 17 | 25, 26, 27 |
| 18 | 25, 29 |
| 19 | 38, 39, 42 |
| 20 | 23, 24, 25 |
| 21 | 23, 24, 25 |
| 22 | 131, 132, 141, 142 |
| 23 | 129, 130 |
| 24 | 3, 49 |
| 25 | 123 |
| 26 | 124 |
| 27 | 49 |
| 28 | 49 |
| 29 | 5 |
| 30 | 19 |
| 31 | 52 |
| 32 | 20 |
| 33 | 11, 58 |
| 34 | 12 |
| 35 | 3, 105, 118 |
| 36 | 47 |
| 37 | 51, 56, 57, 62, 63, 64, 65, 70, 73, 75, 171 |
| 38 | 85 |
| 39 | 128 |
| 40 | 5, 9, 10, 11, 12, 13, 76, 77, 78 |
| 41 | 1, 4, 5, 6 |
| 42 | 16 |
| 43 | 16 |
| 44 | 16 |
| 45 | 16 |
| 46 | 16 |
| 47 | 16 |
| 48 | 16 |
| 49 | 14, 15 |
| 50 | 21, 22 |
| 51 | 150, 151, 152 |
| 52 | 21 |
| 53 | 22 |
| 54 | 47 |
| 55 | 32 |
| 56 | 30, 31 |
| 57 | 147, 148, 149 |
| 58 | 175 |
| 59 | 78 |
| 60 | 67, 68, 69 |
| 61 | 76, 77 |
| 62 | 67 |
| 63 | 68 |
| 64 | 128 |