PERSONNEL ALLOCATION FOR ENGINEERING PROJECTS

by

Louis Francois Theron

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Engineering at Stellenbosch University

Supervisor: Dr. G.C. van Rooyen

Faculty of Engineering

Department of Civil Engineering

December 2013

DECLARATION

Department of Civil Engineering

Stellenbosch University

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

December 2013

ABSTRACT

The logical allocation of tasks in engineering offices currently relies heavily on the experience and intuition of project managers. In large scale projects the complexity of the task allocation procedure exceeds the capacity of human intuition, and a systematic technique is required to aid project managers in assigning tasks to individuals. In this project such a systematic technique is modelled and implemented using the Java programming language. An equation was developed to calculate an individual's workload, and used in conjunction with other criteria to intelligently and systematically select an optimal individual to complete engineering tasks. The software solution is network-based, and also aims to aid project managers in various managerial duties.

OPSOMMING

Die logiese toekenning van ingenieurstake steun tans swaar op die ervaring en aanvoeling van projek bestuurders. In grootskaalse projekte is die kompleksiteit van die taak toekenningsproses veel groter as die kapasiteit van menslike intuïsie. Dus word ‘n sistematiese proses wat projek-bestuurders kan help met die toeken van take aan individue vereis. In hierdie projek is so 'n sistematiese tegniek ontwikkel en geïmplementeer met behulp van die Java-programmeringstaal. 'n Vergelyking is ontwikkel om 'n individu se werklading te bereken en is in samewerking met ander kriteria gebruik om take intelligent en sistematies toe te ken. Die sagteware is network en databasis-gebaseerd en kan ook gebruik word om projek-bestuurders te help met verskeie bestuurspligte.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF DATABASE TABLES

TABLE OF EQUATIONS

TABLE OF GRAPHS

TABLE OF SCREENSHOTS

TABLE OF SOURCE CODE INSERTS

TABLE OF RESULTS

DEFINITIONS

- `Class` – Any word written in this font and format, and starting with a Capital letter refers to Java class. If a name follows, the name references a specific instance of that class. E.g. `Workload wl` indicates that `wl` is an instance of class `Workload`.

- `variable` – Any word written in this font and format, not starting with a capital letter refers to a variable defined in a `Class`.

- `method()` – Any lowercase word written in this font and format, not starting with a capital drop and followed by parenthesis refers to a method defined in a Class.

- TABLENAME – Any word written in all capital letter like this refers to a table of the database.

- **column_head** – Any word or combination of words written in bold and starting without a capital letter refers to a database table column head.

- SQL – Structured Query Language

- ERD – Entity-Relationship Diagram

- _fk – Database tables are comprised of columns and rows. If a column name ends in this postfix, records within the column are foreign keys, and references a value existing in another database table. The referenced value must always be a primary key.

- _pk – If a column head ends in this postfix, records within the column are primary keys that are unique and can be referenced by other tables in the database. Referencing values are noted as foreign keys of this primary key.

- Null – Zero, empty or undefined

- ResultSet – the result of a database query

- GUI – Graphical User Interface. The part of an application displayed on the computer screen, allowing interaction with the computer and the application.
- Human Resource – Synonym for personnel. There are many types of resources that can be used in project management, human resources are just one of them.
- Risk – The probability of loss associated with a specific task. In the case of this document it is associated with the probability of a task not being completed on time.
- PDF - A file format that provides an electronic image of text or text and graphics that looks like a printed document and can be viewed with a PDF reader, e.g. Adobe Acrobat Reader.
- FLATLINE – A priority state a task can be declared as. It indicates that all assigned resources should be working 100% of their time to complete the task as soon as possible. The main reason for flat-lining a task would be if the task neared its latest end date – the date at which if it is not complete, the rest of the project would be influenced detrimentally. Also known as fast-tracking.
- Set – In both mathematics and the relational database model, a set is an unordered collection of unique, non-duplicated items.
- UML – Unified Modelling Language is a set of graphic notation techniques used to create visual models of object-oriented software-intensive systems.
- GRG – Generalised Reduced Gradient. A nonlinear optimisation method.
- CTC – Cost to Company. The rate which an individual is paid by the company to do work.

ACKNOWLEDGEMENTS

I would like to thank my friends and family for all their support and attempts to understand and be interested in this endeavour. I would especially like to thank Mareleen Smit, without whom my life would be meaningless.

Thank you to all the kind people who helped in testing the software and who I tested my theories on. Thank you to everyone whom I had long discussions with me regarding my work and who gave insightful advice.

I would like to give special thanks to soon to be Dr Johann Potgieter. His own work and insights were essential in shaping my project the last two years. He is a remarkably brilliant man, and someone I consider a friend.

Most importantly, I would like to thank Dr Gert van Rooyen, my study leader. His formidable knowledge in the area of informatics was invaluable for the successful implementation and completion of this thesis. The knowledge gathered in this documented thesis would not have been possible without his admirable help and guidance.

Thank you all, I appreciate everything you have done for me.

1. INTRODUCTION

1.1 INTRODUCTION AND BACKGROUND

Projects and project management are cornerstones of the engineering environment. Because of this, it is of the utmost importance to make sure that engineers are supported by the right project management software to make sure management runs smoothly and is as correct as humanly possible.

A project is defined as an organised undertaking designed to achieve an aim [1]. It can be further explained as an undertaking that involves a single definable purpose with well-defined deliverables or results, usually specified in terms of cost, performance requirements or schedule [2]. Every engineering project requires a slightly different approach, even if the same project has been attempted before, such as laying a few cables. Site-specific variables such as terrain, zoning laws, labour market issues, access and public services all combine to make it unique. An engineering project is most likely a once-off activity, never to be exactly repeated again.

Given that each engineering project is unique, it necessarily involves unfamiliarity and risk. It may involve new technology or techniques and possess significant elements of uncertainty for the organisation undertaking the project.

When an organisation wants to undertake a specific project it becomes important for them to ensure that the work is completed as planned. Failure to complete a project could jeopardise the organisation or its goals. Goals may usually be financial, but can possibly be reputation-based.

The incentive to succeed, combined with the inherent risk involved in projects, make planning any project very important. Planning a project correctly will greatly reduce the risk involved with completing it correctly, safely and on time. However, it is just as important for engineers to continuously apply innovative techniques to the way projects and the offices assigned to these projects are managed in order to ensure that the planned aims and milestones are actually achieved.

1.2 MOTIVATION FOR STUDY

One of the many challenges a project manager is faced with is assigning tasks to individuals within the workspace. The project manager should know when available personnel are overloaded so that he can outsource specific tasks in time. The crucial objective when assigning tasks is to maximise the probability that the tasks will be completed correctly and on-time. However, according to sources within the engineering industry, project managers have to rely on rudimentary heuristic methods to decide to whom tasks are assigned [3]. An example of this would be to numerically compare how many tasks individuals have assigned to them, assigning the next task to the one who has least. One manager jokingly said he would give the next task to the person that appears to be the least tired. Although such methods have been working surprisingly well, it is quite apparent that a number of factors are not brought into the calculation. As a result there is a pressing need in engineering design offices for an impartial method to determine who should be assigned more tasks and who should not.

The task assignment problem is not unique to engineering design offices. On construction sites, for example, there are even more resource constraints that have to be considered. Other types of projects may be of such a nature that no logical order of tasks can or want to be found. However, this study is focused on trying to solve the problem specifically in engineering offices. It is assumed that projects being executed by engineering design offices can be broken up into discrete tasks, which can be scheduled and optimised.

Consider, for example, programming projects. Complex programming projects cannot be perfectly partitioned into discrete tasks that can be worked on without communication between the workers and without establishing a set of complex relationships between tasks and the workers performing them [4]. Assigning additional programmers to such a project increases the communication overhead, which will consume an ever increasing quantity of time available for development, indicating that assigning more programmers to a project running behind schedule will make it even more behind schedule. Because these types of projects do not have discrete tasks, a detailed project schedule cannot be created. Such un-schedulable projects fall outside the scope of the study.

Researchers have been investigating the problem of allocating tasks [5]. Task juggling, where workers work on too many tasks at the same time and detrimentally affect performance, has been studied and documented [5]. A rudimentary standard workload formulation[a] is used to assign classes to teachers at the Algonquin College, but this technique is unusable for engineering purposes [6]. Much work has been done in trying to optimise task scheduling [7], and such an optimal schedule is assumed in the methods described here. Research has been done to investigate enhancing task-allocation algorithms for computer hardware [8].

Although good practices for personal workload and task management is advised, no sure way exists that addresses the task allocation problem completely. At most it has been proven that more care has to be taken when assigning tasks than using project evaluation and review techniques [9]. One research team believed that worker skill level in specific disciplines is all that is needed to optimally assign tasks [10]. Individual skill level is difficult to quantify, this and the complexity with regards to calculating and storing skill level for each task before it is even attempted, make this approach unusable for general use.

It is believed that although individual skill level is important in allocating tasks, the reality is that each worker will have to be scored for each expertise individually to ensure correctness and up-to-date relevance. This study instead chooses to focus on a broader spectrum of criteria for allocating tasks, making the assumption that workers employed by the organisation are in fact capable of completing tasks allocated to them.

The software and techniques developed here are intended to be used as tools to aid project managers in their tasks, and not merely to replace them.

[a] No official formulation of the equation could be located

1.2.1 EXISTING SOFTWARE SOLUTIONS

There are numerous existing software solutions that can be used to aid in project management. Microsoft Project [11] is a popular choice, with Bonita BPM [12], 24SevenOffice [13] and BrightWork [14] being some of the other choices. Some of these software packages include convenient web-based task management interfaces, promoting assembly line efficiency. The problem that most of these solutions share, is that they all try and cover as broad a spectrum of services and utilities as possible. This makes the software less relevant for a specific use. Software companies have already seen the need for specialised software, but the engineering industry is lagging behind. One specific software package, Clarity [15], provides automatic and semi-automatic task assignment and scheduling of jobs to field agents, specialising in the field of telecommunications. Although this is close to what is needed in engineering offices, this solution focuses on technicians doing field work, not engineers working on designs. In these solutions there are also one specific component missing, namely a quantifiable factor for individual workload.

1.3 DEFINITION OF THE RESEARCH PROBLEM

1.3.1 MAIN PROBLEM

With technology continuing to evolve, service industries continuing to grow, and domestic entities facing rapidly expanding global competition, the challenge of identifying, developing and deploying the right skill set has never been more important to the future of an organisation than it is now [16]. Nonetheless, even in the face of this realisation, companies are slow to adopt new technologies since irrational tradition typically hampers progress. In order to have any chance of acceptance, new techniques and software should be simple, easy to use, and result orientated. It is believed that companies are more willing to turn to a software package that solves the one specific problem they are struggling with, rather than to one which only lightly touches on some of their issues.

This study focuses on techniques and applied prototype software that can aid project managers in engineering offices. Chief amongst the managerial duties that have to be addressed, is a semi-automatic task allocation procedure. Stated in another way, a technique is required for project managers to easily assign incoming tasks to personnel in an intelligent manner. This has to be supported by a way to generate statistical work breakdown reports. These functionalities have to be available to all project managers connected to the organisation. For this implementation, a centralised relational SQL database was used to persist information across an organisation and ensure the required connectivity.

In summary, the following design goals were attempted:

1. Develop a technique with which project managers can allocate incoming tasks to personnel,
2. Implement techniques to generate graphical reports on tasks and personnel,
3. Provide ease of use and corporate availability via a client-side graphical user interface, or GUI,
4. Ensure integrity and persistence of information by implementing a suitable SQL database structure.

1.3.2 ADDITIONAL PROBLEMS

Task allocation cannot and should not be fully automated, as not all of the finer details of task allocation can be easily automated. The project managers must remain in control since they know the policies and personnel of the organisation. It was consequently deemed essential to develop a way for project managers to have full control over the task assignments. Filters which guide the personnel search, similarly to a sieve, were developed. The filters should be turned on or off individually, according to the policies of the organisation. Further customisation should also be possible in the form of sliders which determine the relative importance of specific filters to the search. After the search has run its course and a best candidate has been determined, it is imperative to still offer the project manager the choice of final allocation.

1.4 SOFTWARE USED

The following software was used in some way to aid in the completion of this thesis.

- PostgreSQL database and server
- Eclipse Java SDK, version 3.7.0
- Google’s WindowBuilder plugin for Eclipse
- JFreeChart libraries for Java
- iText libraries for Java
- Microsoft Access 2013
- Microsoft Office 2013
- Microsoft Excel 2013
- Google Chrome
- Microsoft Paint for Windows 7
- MATLAB R2012b
- www.surveymonkey.net

1.5 OUTLINE OF CHAPTERS

- *Introduction* describes the current state of affairs and why the project is necessary. A general overview of the project and the problems expected during its execution is also provided.
- *Software* describes the functionality of the finished software solution. Databases and data structures are also discussed.
- *Quantifying Workload* aims to describe and develop an equation to compute individual workload. Sensitivity analyses and a survey was utilised to test the validity of the equations developed to quantify workload.
- *Designing the Object Model* discusses modelling in general, and how the specific object model was created.
- *Database Modelling* discusses the mathematics behind data structures, along with key explanations of database terminology. Specific elements of the model are also discussed, with focus on how the model was translated to the SQL database.
- *Enhancing the Model* discusses how the model was enhanced with the addition of features and optimisation.
- *Test Project: Software in Action* runs through a simulated company and its task allocations, thus showing how the software provides results.
- *Project Resource Locator – Key Points of the Code* takes a cursory glance through the most important features of the software solution, with emphasis on how specific achievements were made using Java code.
- *Recommendations for further work* discusses the shortcomings of the software and how it could be improved in future development iterations.

2. SOFTWARE

A significant part of the work described in this document covers the development and prototype implementation of an object model and underlying database that supports project managers in the assignment of tasks. The techniques and technologies employed in the development are described in later chapters. In this chapter a very brief overview of the software is given. The intention is to give the reader a glimpse of what to expect since similar task allocation software, to the best of the author's knowledge, is not available on the market at the time of writing. The various functionalities that the prototype implementation provides come to the surface in the main application window, while the underlying database becomes visible in the form of the data displayed on the user interface.

2.1 FUNCTIONALITY

2.1.1 OVERVIEW

The software developed as part of this thesis aims to aid project managers in their managerial duties by adding a semi-automatic task allocation procedure to their arsenal. Furthermore the software adds additional functionality by allowing project managers to easily generate charts[b] that can be added to reports. These functions are managed by an underlying database model that can potentially be accessed from multiple outlets concurrently. User friendliness and compatibility was intended to be an important factor in the design of the software as it is estimated to speed adoption rates of the software.

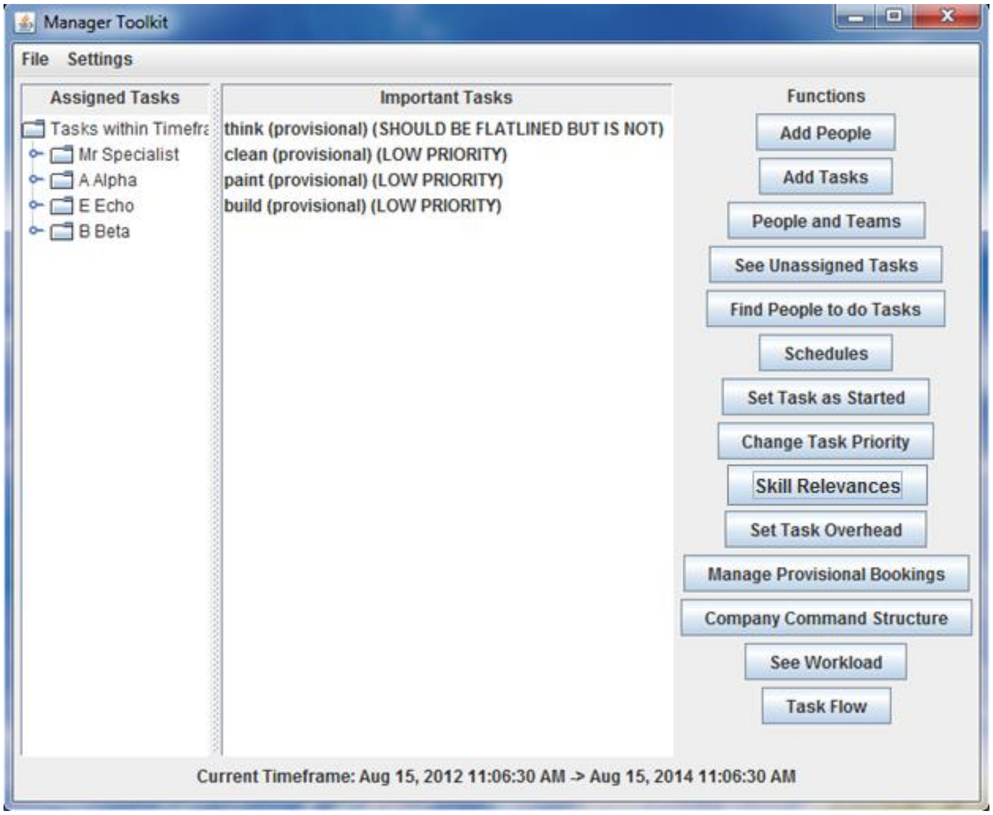
The user interface side of the software, managed by GUIs[c], is split into two main parts: namely the *Manager Toolkit* and the *Chart Drawer*.

[b] Or graphs

[c] Graphical user interfaces

2.1.2 MANAGER TOOLKIT

The *Manager Toolkit* is the hub from where a project manager can control all the different aspects of the software. To keep the interface from getting cluttered with information, only important notifications are shown for events that happen within a set timeframe. The GUI allows the user to manually set the timeframe scope, but ideally it should be automatically locked to an optimal timeframe. The GUI allows managers to identify which employees[d] are to complete which tasks at a glance. Tasks which are flagged as being of higher importance than normal, with FLATLINE being the highest importance, are also shown on the GUI. The *Manager Toolkit* can also be used to manage the information stored in the database by allowing the user to add people when new employees are hired, tasks when a new project comes in, or create new project teams. It is also possible to view the schedule of all the employees for the given timeframe in a Gantt chart format. The *Chart Drawer* window can be easily launched from the *Manager Toolkit*.



think (provisional) (SHOULD BE FLATLINED BUT IS NOT)
clean (provisional) (LOW PRIORITY)
paint (provisional) (LOW PRIORITY)
build (provisional) (LOW PRIORITY)

Current Timeframe: Aug 15, 2012 11:06:30 AM -> Aug 15, 2014 11:06:30 AM

Screenshot 1: Manager Toolkit

[d] Or Human Resources

The most important function available from the *Manager Toolkit* is the ability to intelligently find people to do tasks. Once the user launches the appropriate window, the user can select to either view all tasks, or only view the unassigned tasks. From this list the manager can select any number of tasks that he/she wants assigned, then toggle different search filters on or off, or change their relative importance[e]. The software then takes the manager's input, searches the database for the most appropriate candidates based on the search criteria, and displays the options on screen. The best choice according to the user's criteria for each task is automatically selected, but the manager can make individual allocation changes if he/she so desires. If the settings are accepted and the procedure started, the task allocations are stored as provisional bookings. Future improvements may enable users to take these provisional bookings and create cost estimates for how much a project is going to cost the organisation in terms of human resources. If the manager is satisfied with the allocations, it is possible to upgrade these provisional bookings to normal bookings. It is important to note that when a search is completed, it is entirely possible that no suitable candidates will be found. In such a case it is possible that the organisation simply does not have the manpower to complete the tasks and indicates that outsourcing the tasks is a must[f]. If the manager does not recommend outsourcing the task, it is still possible to forcibly allocate a task to an employee, it will however result in their workload jumping to an above acceptable level. This level of workload acceptance can be manually determined by the manager.

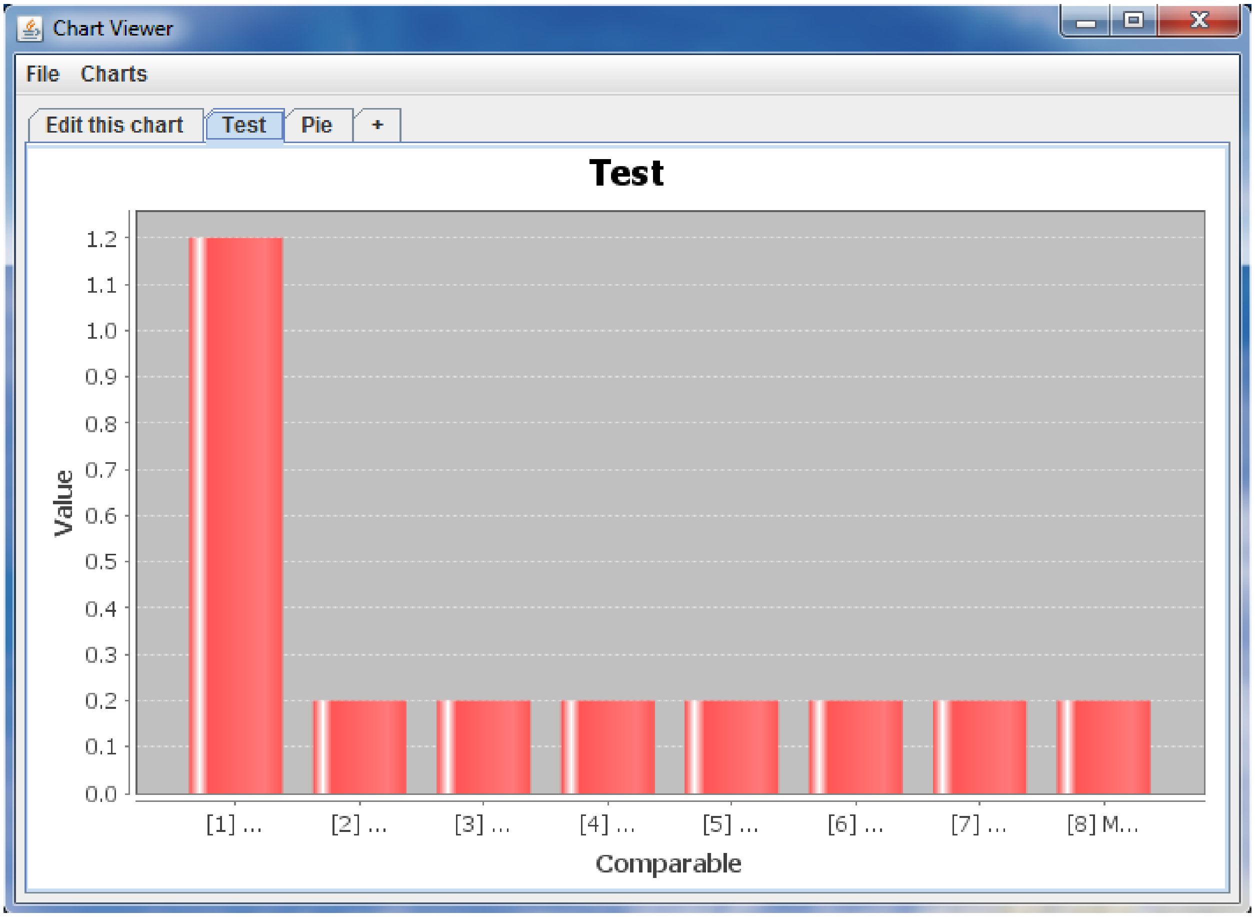
2.1.3 CHART DRAWER

The *Chart Drawer* is the part of the software that project managers use to draw up charts. Multiple charts can be created and stored for use in reports. While the user is still busy setting up the report, the charts can be saved to file, and restored later to continue working on them. When the user is done, all the charts can be exported to a single PDF for use in a report. When creating a chart, the user can select a graph from a short list of frequently used charts, e.g. work-breakdown pie chart, or he/she can create a unique chart. It is easy to select a chart type and the

[e] The search method will be discussed in more detail in chapter *6.2* Optimisation and Finding People

[f] Hiring an external resource

comparing data sets from handy drop down lists. After the comparable data sets have been selected, an automatic database query retrieves the relevant results from the persistent database. Different chart specific options can be toggled to customise the resulting graph. Once the chart is created, the user can zoom in to specific parts, change axes information or even change the colours used. These visual alterations will be remembered by software and can be exported to a PDF when so desired.



Screenshot 2: Chart Drawer

2.2 DATABASE

Traditional Human Resource databases track benefit information, data of newly hired personnel, performance reviews, exhaustive family information and perhaps salary and job history [16]. The database that is necessary for this software implementation does not need most of this information and is not meant to replace these already existing employee databases.

A new database was developed to ensure all the necessary data is present. Storing the same data multiple times is traditionally perceived as something to be avoided at all costs, however, the nature of creating a new isolated database means that data duplication across multiple databases is possible and even an unfortunate necessity.

The database was designed to be able to incorporate any future additions to the model and software. It not only stores basic employee data, but also tracks the projects and tasks completed by the organisation. This additional information can later be used to help serve as an engineering accounting platform by enabling the calculation of how much was spent on human resources. Functionality for doing a personnel allocation halfway, then stopping to be continued later was also implemented. Different task allocation scenarios can be created and quoted, to test the viability of projects.

SQL is a language that can easily communicate with databases. It is aimed to store, manipulate and query data stored in relational databases [17]

3. QUANTIFYING WORKLOAD

3.1 INTRODUCING WORKLOAD

An impartial method to determine who should be assigned more tasks and who should not, needs to be developed. Although current heuristic methods have been working surprisingly well, no task-assignment method could be found in the literature that considers individuals, as well as the relative difficulty of specific tasks, to more accurately decide with whom it is safe to leave a task. In this section such a method is developed. It is based on the workload, defined below, that individuals experience. It is very important to note that in order to make the method at all viable for use in industry, only data that is reasonably easy to capture should be used. If this were not the case and the method required a total restructuring of human resource management, combined with a multitude of additional costs, it will not be implemented and the industry will carry on business as usual.

The term used to describe how busy a person is[g], is workload. An individual's workload is represented as a percentage and is defined as the amount of work he/she has to do in the time available:

$$\text{workload} = \frac{\text{project hours per day}}{\text{hours in work day}} * 100\%,$$

Equation 1: Workload definition

Project hours are the sum of hours an individual is expected to work on all tasks assigned to him. Work hours are defined as the number of hours an individual is expected to work in a day. The higher an individual's workload is, the busier he/she will be, which increases the risk when a new task is assigned to him. The lower the workload, the less an individual has to do, and the safer it is to leave a task with him/her. Workload, as defined, is a very simple indicator that can be used in the allocation of tasks. However, there are a number of factors that influence the workload that an individual *experiences*. For example, if a person has too many tasks he feels overwhelmed, even though his maximum computed workload is limited to 100%. The components that make up

[g] As well as an indication whether said person can handle additional tasks

an individual’s *perceived* workload are considered in the following section.

3.2 COMPONENTS OF PERCEIVED WORKLOAD

The workload equation 1 provides a basis for task allocation, but a number of components make up the perceived workload, or PWL, that an individual experiences. These components have to be accounted for in the computation of workload in order to make it useful for task allocation. Since the components have to account for **perceptions** they are difficult to quantify. A novel approach was devised by the researcher: For each component a suitable equation was identified which *qualitatively* describes the influence that the component has on the workload that an individual experiences. The parameters of the equation were then adjusted, on the basis of investigations, regarding the magnitude that the component has on the actual workload.

PWL is thus defined as the sum of a few contributing components scaled to adjust their magnitude:

$$\text{PWL} = \sum(\text{workload components})$$

Equation 2: Perceived workload definition

The various components and the qualitative equations used to describe them are defined in the sub-sections below.

3.2.1 PERCEIVED WORKLOAD COMPONENT: WORKDAY LENGTH

The number of hours in a person's workday directly enters the workload equation 1. Since the calculation of workload specifically aims to facilitate project management in engineering design offices, it may be assumed that reasonably standard office hours are kept. However, depending on the amount of work to be done, office hours may be limited a little, or it may creep upwards when the office is under pressure. If the office does not have enough work, task allocation is not a problem and consequently that scenario is not investigated further. However, when work pressure is mounting, proper task allocation is important. In this situation individuals feel the pressure and the length of their work days start to increase. This has an influence on their perceived workloads, which has to be accounted for when actual workloads are computed.

An equation that measures the effect of work day length on workload should have the following properties:

- if a person is on leave or sick his work day has zero hours and the effect is zero.
- if a person only works a few hours a day, the change between not working at all and working one or two hours is dramatic, i.e. his PWL increases rapidly.
- if a person who works a normal 8-hour day stays on to work another hour or two, his PWL does not increase as rapidly as the case described above.

The function of equation 3 has the required properties:

$$\text{workday length, component of PWL} = \begin{cases} 0, & L = 0 \\ 0, & n = 0 \\ \log_a(L+1)\,, & n \neq 0 \end{cases}$$

Equation 3: Contribution of workday length on perceived workload

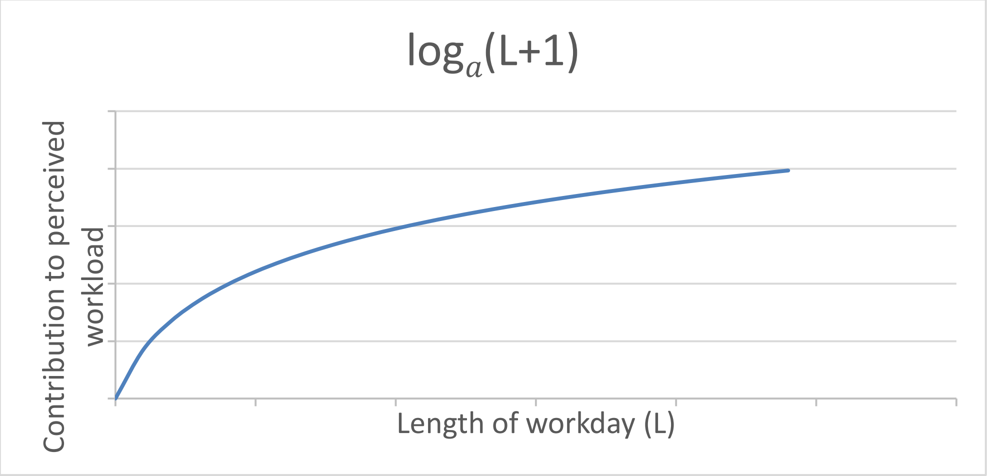
in which *L* = work day length,

a = parameter that adjusts the gradient of the function,

L+1 is taken to ensure the function is zero when *L* = 0,

n = number of tasks assigned to the person under consideration

Equation 3 is shown graphically in graph 1, indicating that the equation has all the properties described above.



Graph 1: Contribution of workday length to perceived workload

3.2.2 PERCEIVED WORKLOAD COMPONENT: SLACK

Most engineering companies have lists of typical tasks and the average time each task takes from start to completion. As it is very subjective to attach a numerical value to any given task to indicate task difficulty, task duration can be used as an unbiased approximation for how difficult a task is supposed to be. These duration values are usually for the company as a whole and does not take the circumstances of individual branch offices into account. Take, for example, a task "Weekly hour-long meeting". As the name clearly states, this task should take an hour to complete. However, what if the office where the individual that is responsible for the task is on the opposite side of the city from where the meeting is held? The time it takes to drive to the meeting would be considered as a task overhead. Another example of overhead would be the additional time it takes between requesting old paper building plans locked up at head-office and physically receiving a scanned copy via email. A similar task overhead can be imagined for any type of task, whether it is time to actually get to where the task must be completed, or some other difficulty specific to an office.

The following expression formulates the relationship between total task duration and task overhead mathematically:

$$\text{total task duration} = \; t_i(1+o_i),$$

Equation 4: Total task duration

Where, t_i is the statistical average task duration of a specific task and o_i the overhead attached to that task.

The effect that task overhead has on individuals' PWL is that it impacts on their "free time" in the workday. We call this "free time" an individual's slack, defined as:

$$Slack = \; L - \sum_{i=1}^{n}(t_i(1+o_i)$$

Equation 5: Calculation of an individual's slack

In which t_i = duration of task i assigned to individual,

n = number of tasks assigned to a person,

o_i = overhead of task t_i for the office where the individual works,

L = length of work day.

An equation is required that measures the effect of an individual's slack on his PWL. The equation should have the following properties:

- If the slack is significant, a small change in slack has a negligible effect on PWL.
- As the slack becomes small, a change in slack starts to have an effect on PWL.
- If the slack becomes negative, i.e. the individual has to start working overtime, the effect on PWL increases rapidly.

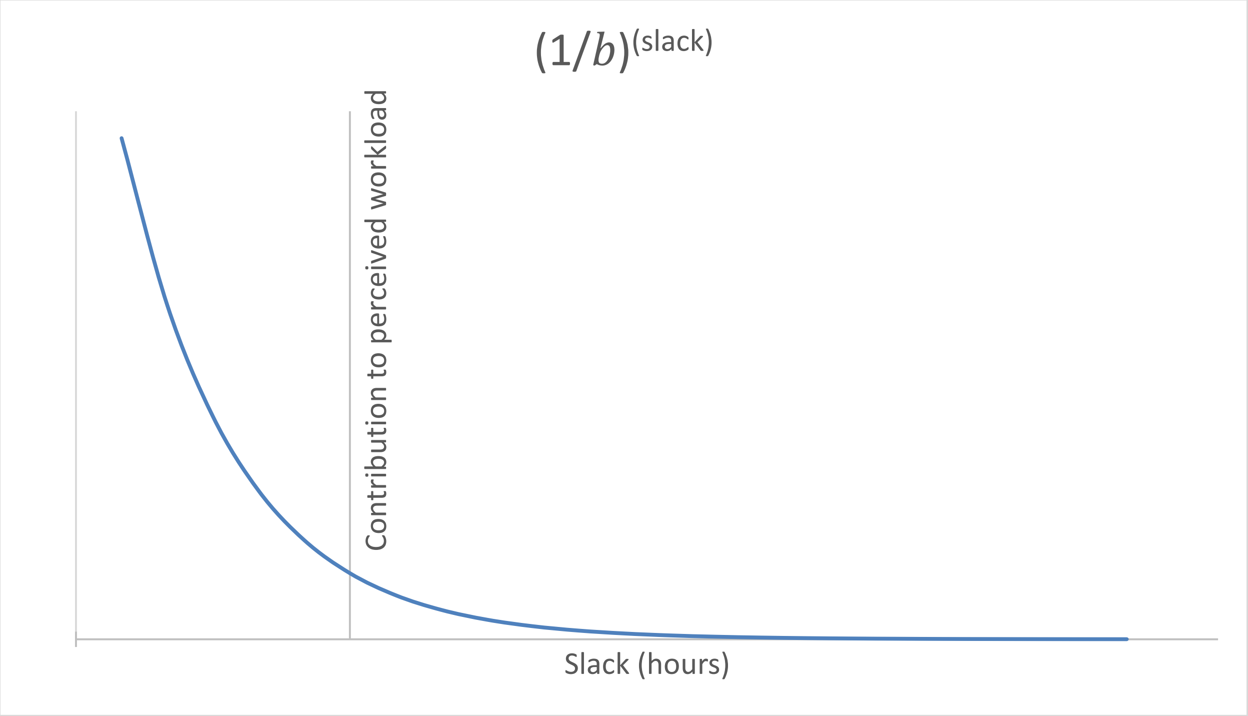
The exponential function of equation 5 has the required properties:

$$\text{slack, component of PWL} = \left(\frac{1}{b}\right)^{L-\sum_{i=1}^{n}(t_i(1+o_i)} = \left(\frac{1}{b}\right)^{slack},$$

Equation 6: Preliminary contribution of slack on perceived workload

In which b = parameter that adjusts the growth of the function.

Equation 6 is shown graphically in graph 2, indicating that the equation has the required properties.



Graph 2: Contribution of slack on perceived workload

Equation 6 has to deal with two special cases, namely when an individual's workday length is zero and when he has no tasks. In both cases the effect on workload should be zero. Equation 6 is consequently adapted as follows to form equation 7:

$$\text{slack, component of PWL} = \left(\left(\begin{cases} 0, & L = 0 \\ 0, & \sum_{i=1}^{n}(t_i(1+o_i) = 0 \\ \frac{1}{b}^{(T-\sum_{i=1}^{n}(t_i(1+o_i)}, & \sum_{i=1}^{n}(t_i(1+o_i) \neq 0 \end{cases}\right.\right)$$

Equation 7: Contribution of slack on perceived workload

3.2.3 PERCEIVED WORKLOAD COMPONENT: TASK STIFFNESS

In the previous section task overhead was introduced, which accounts for variability in task duration due to the work environment. This section introduces an overhead due to the nature of the task itself and is characterised by the so-called task stiffness. Stiffness may also be called “a lack of permeability” since it is a measure which indicates whether a task can be partially completed while the responsible individual also works on other tasks. As such it indicates whether a task can be broken up into segments, or if it has to be completed from start to finish without any interruptions. A stiffness of 1.0 indicates that a task has to be completed fully before moving on, while a stiffness of 0 indicates that the task can be left at any time to work on other tasks. This clearly has an influence on how much an individual can fit into a day and is thus an important influencing factor on workload. Task stiffness will, however, be a measure that will be more useful when trying to restructure task allocation and balance workloads. Task stiffness only has an effect on PWL when an individual is assigned more than one task at the same time. The more tasks, the greater the effect of stiff tasks will be. It is assumed that the stiffest task among all tasks will dictate the difficulty in completing other tasks.

An equation that measures the contribution of task stiffness to PWL should have the following properties:

- is only applicable when an individual is assigned more than one task.
- the effect should increase linearly with increasing task stiffness.
- the gradient of the equation is dependent on the number of assigned tasks, with the effect of task stiffness being greater for a larger number of tasks.
- only the highest stiffness among all assigned tasks should be significant.
- needs some constant to explicitly control the contribution of stiffness to PWL.

The following equation is proposed:

$$\text{task stiffness, component of PWL} = \begin{cases} 0, & n \leq 1 \\ e * s_{max} * n^{d}, & n > 1 \end{cases}$$

Equation 8: The effect of task stiffness on perceived workload

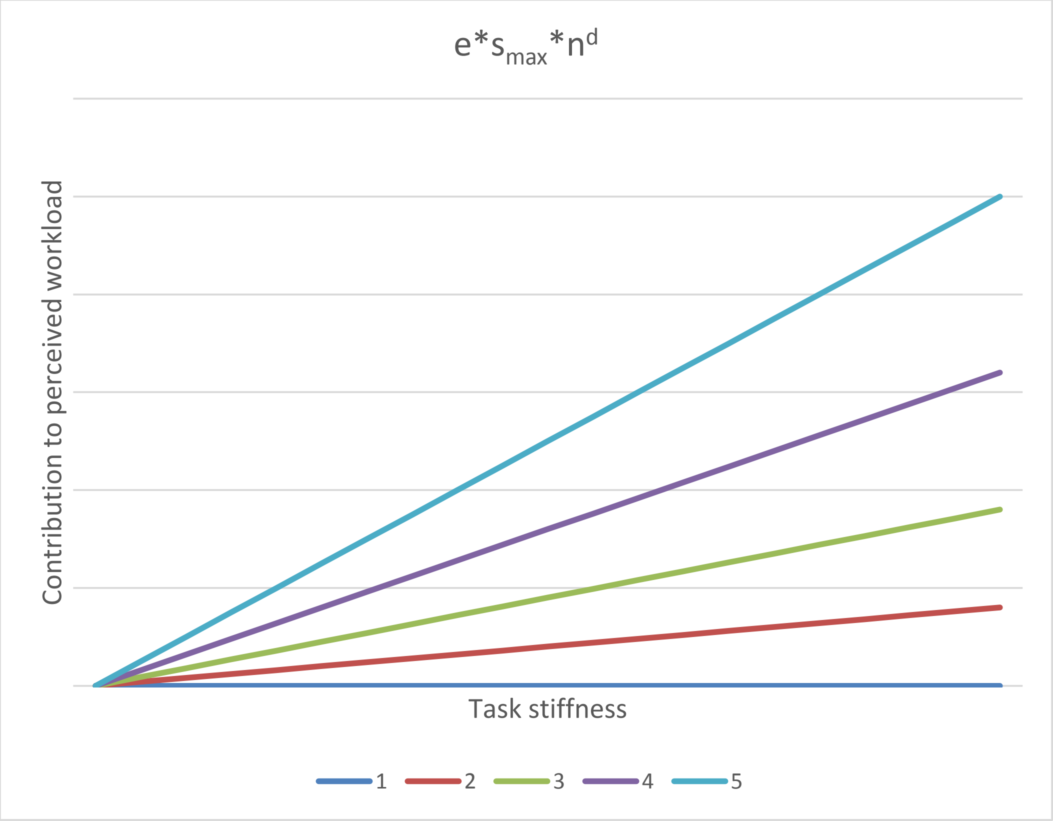
In which s_{max} = the highest task stiffness among all tasks assigned to the individual,

n = the number of assigned tasks,

d = parameter that adjusts the growth of the function,

e = parameter that adjusts the contribution of stiffness to PWL.

Equation 8 is shown graphically in graph 3, indicating that the equation has the properties described above.



Graph 3: The effect of task stiffness on perceived workload

3.2.4 PERCEIVED WORKLOAD COMPONENT: NUMBER OF ASSIGNED TASKS

The contribution that the number of tasks assigned to an individual has on his/her PWL is investigated in this section. It is believed that when two individuals are compared, the individual with more tasks assigned to him will be busier, especially when they work the same amount of time. This is important as it is very unlikely for individuals to ever only have one task to do in a day. It is a well-known time saving method to group tasks together in batches so that they can be focused on and completed together.

In a study done by Microsoft Research called "A Diary Study of Task Switching and Interruptions" [18], it is documented that information workers switch among tasks a significant number of times per day. The study shows that tasks which are returned to later were more complex, on average, than shorter-term tasks. Complex tasks which are significantly lengthier in duration were obviously interrupted more. Because the tasks were longer the tasks experienced more revisits by workers after attention switches. These complex tasks were also rated to be harder to return to than shorter term projects.

It is reasonable to assume that engineering workers will behave similarly, namely that the more tasks an individual have to do, the more attention switching will happen during a day. According to the definition of basic workload, the number of tasks do not have an effect, but due to attention switching an individual will feel busier. Therefore the number of tasks has an effect on an individual's PWL, as illustrated in the following example: Person A has one task to do and the duration of that task is just as long as his workday. Assume workday duration is 8 hours. According to our earlier definition of workload, Person A has 8 hours of work to do in 8 hours, which implies that he/she is 100% loaded. Person B has two tasks, both with duration of 4 hours each. Assuming the two people work in the same office, and have the same work hours, it is believed that Person B will feel slightly more loaded than Person A. This suggests that Person B could, for argument's sake, be 102% loaded. Just how much of a difference additional tasks make will be discussed later.

An equation that measures the effect of increasing the number of tasks assigned to an individual on PWL should have the following properties:

- if an individual has zero tasks assigned to him/her, the contribution to PWL should be zero.
- the contribution of increasing the number of tasks assigned should grow exponentially.
- a low number of tasks should result in less attention switching, resulting in a flat curve
- a high number of tasks should result in more attention switching, with each added task swelling the growth of PWL and increasing the curve gradient.

The exponential function of equation 9 has the required properties:

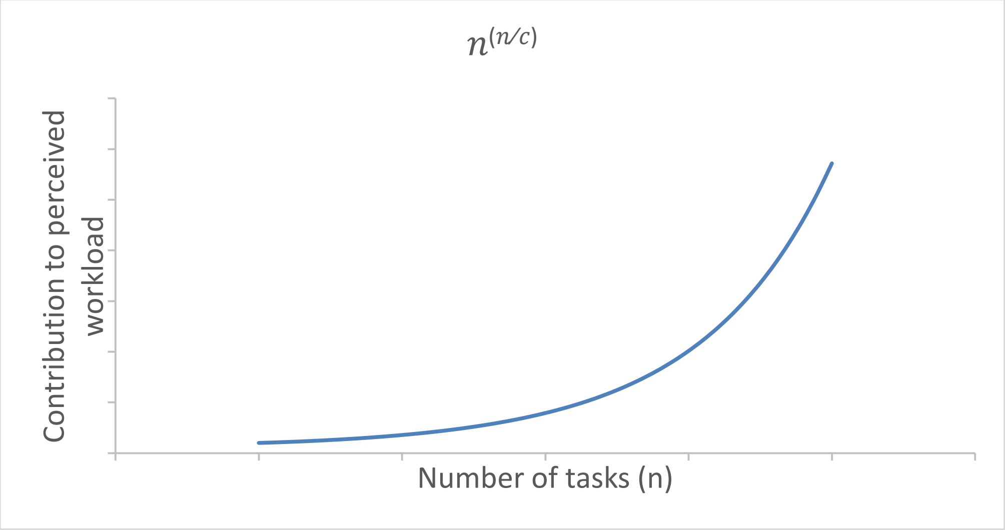
$$\text{number of tasks, component of PWL} = \begin{cases} 0, & n = 0 \\ n^{n/c}, & n \neq 0 \end{cases}$$

Equation 9: The contribution of increasing the number of assigned tasks on perceived workload

In which *n* = number of assigned tasks,

c = parameter that adjusts the growth of the function.

Equation 9 is shown graphically in graph 4, indicating that the equation has the properties described above.



Graph 4: The contribution of increasing the number of assigned tasks on perceived workload

3.2.5 IDLE FACTOR

The last component that needs to be added to the method of calculating PWL is an individual's idle factor. Idle factor is the percentage of a work day that an individual is expected to be idle. This can be when people take bathroom or coffee brakes, or time wasted browsing the internet. There are some documented statistics suggesting a typical idle factor value [19], but again it will be up to the project manager in question to personalise this value for the formula to be as effective as possible. The idle factor can also be used as an external manipulation tool to alter the values given by the formula. Project managers may want to do this if they want their employees to work a bit more, or a bit less. The idle factor will just be shown as IF in the equation. To preserve the definition of the equation, the following mathematical form shall be used:

$$\text{Idle factor} = \begin{cases} 0, & L = 0 \\ 0, & n = 0 \\ IF, & n \neq 0 \end{cases}$$

Equation 10: Contribution of individual idle factor to perceived workload

3.3 WORKER EFFICIENCY

The complexity of a task has an influence on workload, since it directly determines the duration of the task. The challenge with task complexity and duration is that it will be experienced differently by different individuals, raising the question of how difficult an individual will find a specific task. The answer to this dilemma lies in human psychology and experience. To really find out how difficult a task is for individuals, extensive psychological tests will have to be performed, trying to identify how a person performs in each test under different circumstances. Since the aim is to develop a method that does not extraneously modify the daily running of a company, trying to compute exact task difficulty for each individual should be avoided. Where task overhead provides a way to customise task duration and accounts for variability inherent to offices, worker efficiency caters for variability inherent to individuals. The variability measure that is easily available and can be reasonably estimated, is an individual's efficiency. All the unknown or unreachable psychological factors boil down to the idea that they influence how effective or efficient an individual is at his/her work. The less efficient an individual is at doing tasks, the longer these tasks will take to complete, resulting in him/her feeling overwhelmed and having an effect on his/her PWL. It is possible to approximate efficiency the same way in which task overhead affects the duration of a task. It can, however, be simplified to a single scale factor which is easier to incorporate into the PWL equation. It is believed that if an individual is infinitely efficient, he/she would be able to do any amount of work in the allotted time, resulting in a PWL of zero. With the same logic, if an individual is infinitely inefficient, no amount of time in the world would be enough to complete a task. This implies a directly linear relationship between worker efficiency and PWL, as the more efficient one becomes the more work one can fit into a day.

The term that indicates the effect of worker efficiency on PWL should have the following properties:

- the effect on PWL should result in a workload of zero if an individual has zero inefficiency.
- the effect on PWL should result in a maximum workload if an individual has zero efficiency, i.e. maximum inefficiency.

Worker efficiency, or inefficiency, is measured using the so-called time inefficiency factor. This factor is defined as the fraction of an hour an individual spends completing a statistical hour of work[h]. For example: if an individual works twice as fast as normal, the factor would be 0.5 and would be 2 if an individual worked twice as slow.

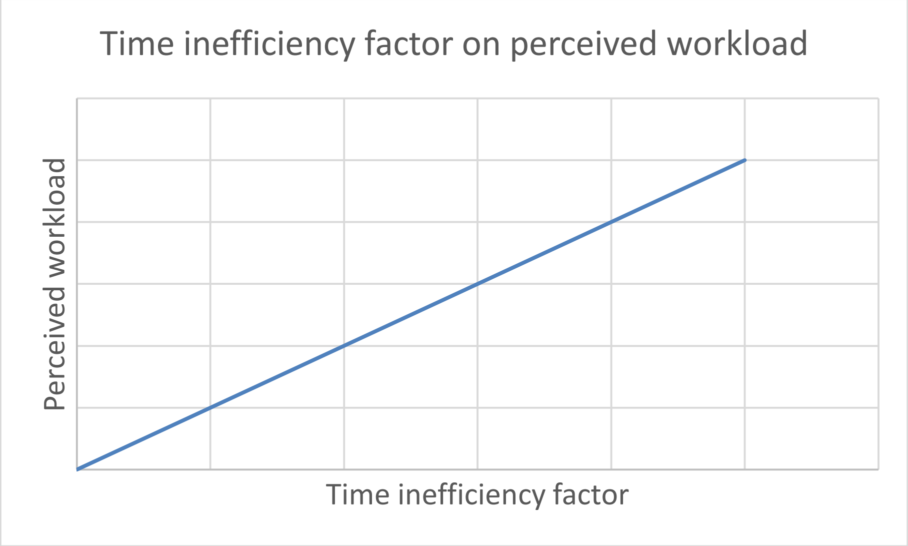
The following expression formulates the time inefficiency factor mathematically:

$$t_{\text{ineff}} = \frac{\text{hours required to complete one hour of work}}{1 \text{ hour}}$$

Equation 11: Time inefficiency factor

While the data used in the components described before reside in the projects data store of the company, a person's efficiency or inefficiency is a little harder to obtain. There are many ways to approximate work efficiency, for example having employees fill out time sheets or just approximating it in-situ[i]. Ultimately it is up to the project managers to find and implement a suitable method to gather this data.

The effect of worker inefficiency on PWL is shown graphically in graph 5:



Graph 5: Linear relationship between time inefficiency factor and perceived workload

[h] The average time it takes to complete a hour long task, according to statistical data
[i] Guessing efficiency on the fly without prior thought or calculations

3.4 FINAL FORM OF PERCEVIED WORKLOAD EQUATION

The components making up PWL have been described above, as well as worker efficiency. In this section they are entered into an equation for PWL. The nature and definition of each component determine its unit of contribution to PWL. The effect of worker efficiency and the idleness component have to be made special mention of. Due to the nature of worker efficiency, it influences all the components except idleness. If an individual is more efficient, a global effect will be felt across all components, reducing overall PWL. An individual cannot, however, be more efficient at being idle. An improved expression for PWL is:

$$
\begin{aligned}
\mathrm{PWL} &= t_{\mathrm{ineff}} * \sum(\mathrm{components}) + \mathrm{Idleness} \\
&= t_{\mathrm{ineff}} \\
&* \left(\left(\begin{cases} 0, & L = 0 \\ 0, & \sum_{i=1}^{n} (t_i(1 + o_i) = 0 \\ \frac{1^{(L - \sum_{i=1}^{n} (t_i(1+o_i)}}{b}, & \sum_{i=1}^{n} (t_i(1 + o_i) \neq 0 \end{cases} + \begin{cases} 0, & L = 0 \\ 0, & n = 0 \\ \log_a(L + 1), & n \neq 0 \end{cases} \right. \right. \\
&\left. + \left(\begin{cases} 0, & n \leq 1 \\ e * s_{max} * n^d, & n > 1 \end{cases} \right) + \begin{cases} 0, & n = 0 \\ n^{n/c}, & n \neq 0 \end{cases} \right) + \begin{cases} 0, & L = 0 \\ 0, & n = 0 \\ IF, & n \neq 0 \end{cases}
\end{aligned}
$$

Equation 12: Individual perceived workload equation framework

In equation 12 above it is assumed that all of the components have an equal contribution to the PWL, but this is not the case. Some components are more important than others in determining the PWL. What these divisions are exactly, are unknown as of yet and each component is assigned weight to alter its contribution. The weights have to be a value between 0 and 1.0, sum up 1.0, and ensure that the PWL is always equivalent to basic workload.

It was found, however, that if the component governing the effect of task stiffness is scaled alongside the other components, the equation returns illogical results when task stiffness does not play a role in the equation, i.e. for zero or one assigned tasks. The parameter *e*, that adjusts the contribution of stiffness to PWL, already scales the component with a fixed amount. Consequently it is unnecessary to scale it again with a weight. The final form the equation is:

$$
\begin{aligned}
\mathrm{PWL} = t_{\mathrm{ineff}} * \Bigg(& \overbrace{\left(\begin{cases} 0, & L = 0 \\ 0, & \sum_{i=1}^{n}(t_i(1+o_i) = 0 \\ \dfrac{1^{(L-\sum_{i=1}^{n}(t_i(1+o_i)}}{b}, & \sum_{i=1}^{n}(t_i(1+o_i) \neq 0 \end{cases}\right)}^{\text{slack}} (F_1) \\
& + \underbrace{(\begin{cases} 0, & L = 0 \\ 0, & n = 0 \\ \log_a(L+1), & n \neq 0 \end{cases})}_{\text{workday length}} (F_2) + \underbrace{\left(\begin{cases} 0, & n = 0 \\ n^{n/c}, & n \neq 0 \end{cases}\right)}_{\text{number of tasks}} (F_3) \\
& + \underbrace{\left(\begin{cases} 0, & n \leq 1 \\ e * s_{max} * n^{d}, & n > 1 \end{cases}\right)}_{\text{stiffness}} \Bigg) + \underbrace{\begin{cases} 0, & L = 0 \\ 0, & n = 0 \\ IF, & n \neq 0 \end{cases}}_{\text{idleness}}
\end{aligned}
$$

Equation 13: Individual workload equation combined – final form

3.5 CHOOSING CONSTANTS

The various components of the equation to calculate PWL have been described and formulated. The equation contains parameters and weights that must be assigned or determined in order to quantify the PWL. Parameters can be chosen intelligently or calculated according to tendencies inherent to the equation, as described later.

3.5.1 WEIGHTS

The weight factors of equation 13 will now be examined to determine their values. Within the slack-component, task duration and task overhead are combined to form the concept of slack. This is the only component that incorporates task duration. Since task duration is the only direct measurement of unbiased task difficulty, it is believed that the slack-component should receive precedence over any other component. This means that the weight factor governing the slack-component, F_1, should be the largest. The component of task stiffness, although important, is not assigned a weight factor, as described before.

It is important to note that as the nature of this equation is qualitative. The exact values do not matter overly much, as long as results are correct relative to each other. The weights will initially be chosen, then analysed in more detail in a sensitivity analysis that follows. For now[j], the relationship between F_1 and F_2 is chosen to be $F_1 = 1.923 * F_2$[k]. F_3 follows from:

$$1 = F_1 + F_2 + F_3$$

Equation 14: Balancing weight factors

Then, if F1 = 0.5 it follows that F2 = 0.26 and F3 = 0.24, in compliance with the argument above.

[j] This relationship may change, discussed later

[k] The ratio of F_1 to F_2 was chosen to result in a reasonable distribution of weights, starting with F_1 as 0.5.

3.5.2 COMPONENT PARAMETERS

Each of the components that contribute to the PWL has one or more parameters that must be qualified. The choice of values is described below.

Referring to the workday length component of equation 13, the constant *a* defines the shape of the logarithmic function and dictates the effect of workday duration on PWL. If all other components and factors are neglected, the value of *a* should yield a PWL of 1 for a given choice of standard workday length *T*. For *T* = 8, the PWL equals 1, i.e. it equals the basic workload, if *a* = 9.0.

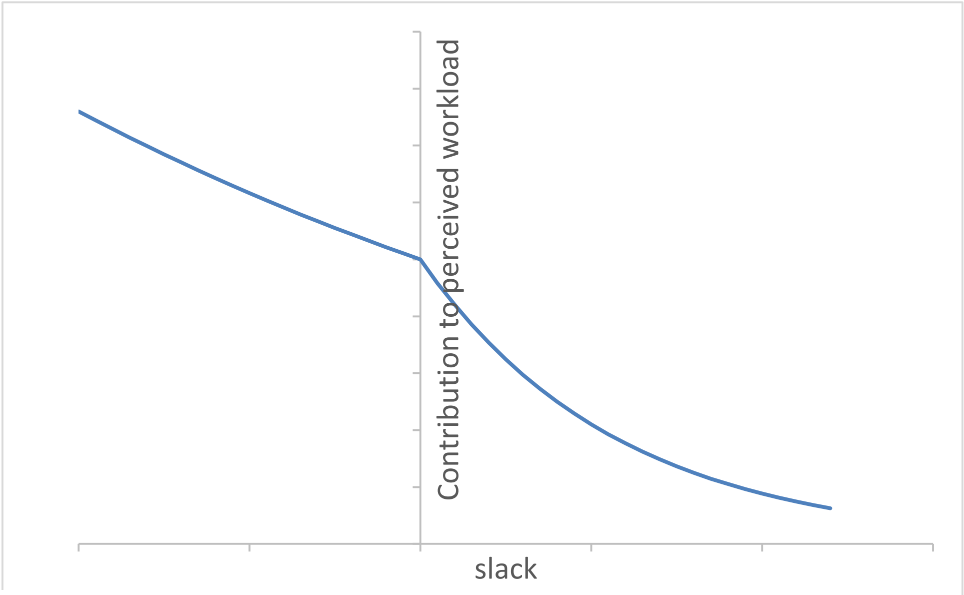
The slack-component is a function of the parameter *b*. Keeping the standard 8 hour workday in mind, a value for *b* must now be chosen. If an individual only spends half of his day on tasks, it seems intuitive that he/she should only be 50% loaded. Half a day, according to the pre-defined standard workday length, is 4 hours. If we choose *1/b* as 0.841, the function yields 50% at 4 hours free time and also approaches zero relatively quickly. This constant would be a perfect fit if not for the very steep incline for negative slack. The formula makes the effect of an individual having to work 1 hour overtime result in an 18% jump in PWL. Two hours of overtime results in a 40% increase in PWL. Although the function was chosen specifically to load overtime heavily, these values are too high. As a result the function for slack has to be split into two parts: the current part for positive values of slack, but a modified expression that will yield more realistic values for negative values of slack.

In a study performed by Revay and Associates called "Calculating Loss of Productivity Due to Overtime Using Published Charts – Fact or Fiction" [20] the effect of working overtime was investigated. Revay and Associates found that a 50 hours per week work schedule results in a loss of productivity of 10 percent on average. A 60 hour week results in a 17 percent loss, while a 70 hour week results in a 31 percent loss. These weekly work hours can be simplified as 2 hours overtime per day, 4 hours per day and 6 hours per day respectively. If this growth is modelled using an exponential function, the value of *1/b* that yields in values closest to those claimed by Revay and Associates is 0.959. The split expression for slack is therefore:

$$\text{Slack} = \left(\begin{cases} 0, & T = 0 \\ 0, & \sum_{i=1}^{n}(t_i(1+o_i) = 0 \\ 0.841^{(T-\sum_{i=1}^{n}(t_i(1+o_i)}, & \sum_{i=1}^{n}(t_i(1+o_i) \le T \\ 0.959^{(T-\sum_{i=1}^{n}(t_i(1+o_i)}, & \sum_{i=1}^{n}(t_i(1+o_i) > T \end{cases} \right)$$

Equation 15: Contribution of slack on perceived workload

Graph 6 visually represents a typical result from equation 15:



Graph 6: Contribution of slack on perceived workload

As equation 17 above is an empirical equation, it is not optimised or intended to be used for very high or very low values of slack. It does not take into account physical limitations, for example the fact that it is impossible for a person to have more than 24 hours free per day, or that workday plus overtime duration cannot exceed 24 hours.

The parameters of equation 13 that still have to be determined are *c,* used in the expression for the number of assigned tasks and *d* and *e* used in the expression for task stiffness. Although there are many studies and theories about human psychology[l] that support the assumption that PWL increases with an increasing number of tasks, no studies based on quantitative data exist which show the actual effect. Because of this, a value for *c* has to be found in another way. Since the parameters *c*, *d* and *e* determine the effect of added tasks on an individual's PWL, they have to be solved simultaneously. Since parameter *e* serves as the weight factor for task stiffness, its value is expected to be small, otherwise stiffness will overwhelm the equation.

Given equation 13, and a task stiffness of zero, the values $c = 33.96$, $d = 3.405$ and $e = 0.000472$ result in a PWL of 101.0% for two assigned tasks, and 109% for six assigned tasks. These results were obtained by solving for a minimum value of PWL for five assigned tasks using GRG Nonlinear optimisation[m]. When stiffness is fixed at 1.0, the maximum stiffness, two tasks render a PWL of 101.5%, while 6 tasks render a PWL of 130%.

A realistic value for stiffness is 0.5. When this is the case, two tasks results in a PWL of 101.25%, and six tasks 119.5%. These values seem reasonable, and were tested against opinions of managers, as described in 3.8.

The GRG method allows nonlinear constraints and arbitrary bounds on an equation's variables. The idea of the method is to choose the independent variables to be the reduced gradient. A step size between the boundaries is then chosen and for each step a correction procedure applied to optimize the answer. The calculated gradient is used in the correction.

[l] See "Stroop effect" [32]

[m] Generalized reduced gradient method

3.6 SENSITIVITY ANALYSIS

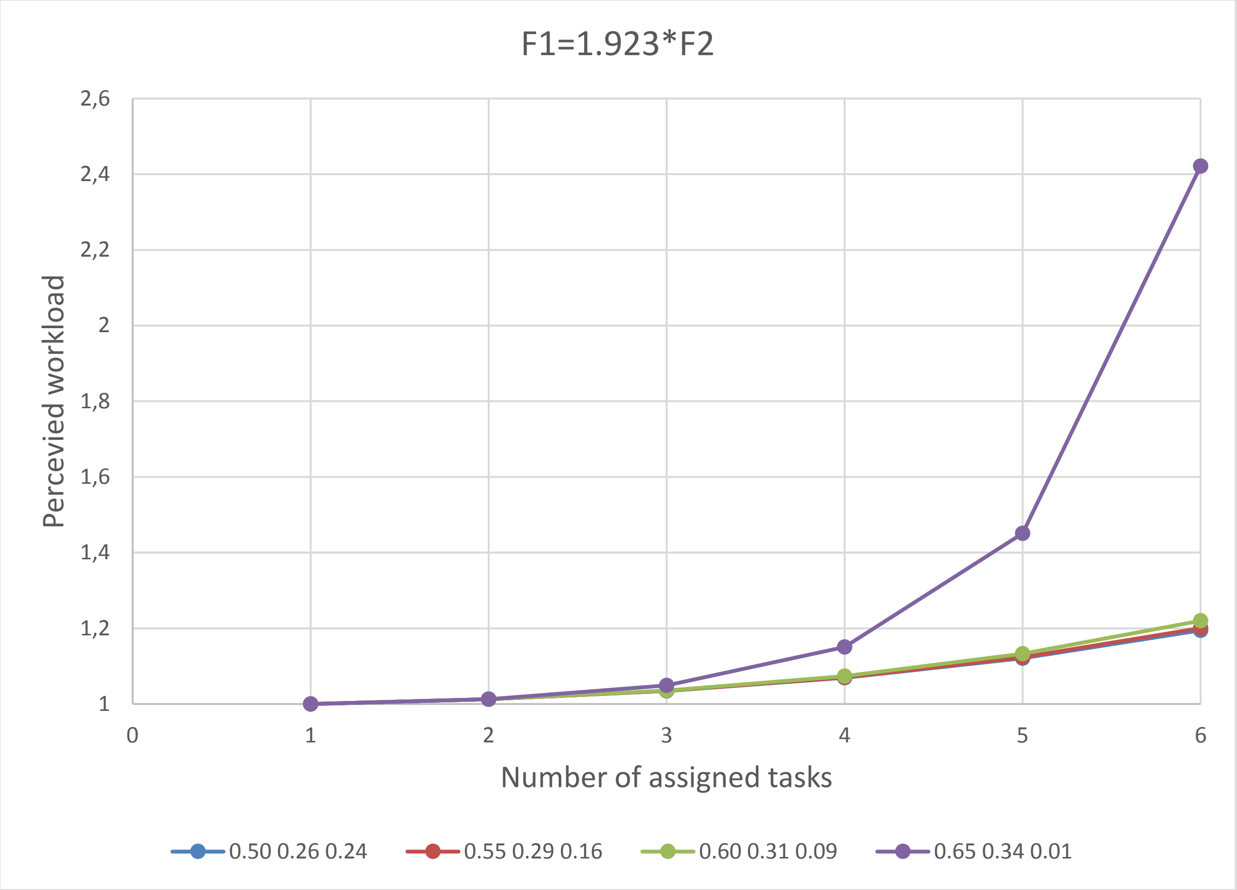
The validity of the weight factors and component parameters of the PWL equation 13 have to be evaluated. Keeping in mind that exact quantification of the PWL is not crucial, it is appropriate to determine how sensitive the expression is to the various weight factors and parameters. If the sensitivity to a certain factor or parameter is low, its exact value is less important. However, if a small change is the value of a weight factor or parameter causes a significant change in the value of the *PWL*, the value of that factor or parameter has to be considered with more care.

All sensitivity analyses are performed by taking the PWL equation, varying one specific variable, or set of variables, while keeping other variables constant and plotting how the change influences PWL. PWL is always plotted against the number of assigned tasks, since task assignment is the core issue. Other variables PWL could be plotted against include workday length, stiffness or slack. However these variables cannot be controlled by the project manager in the same way that the number of assigned tasks can be controlled. Consequently their effect is the same for all members of a project team. It should be remembered that they do have an effect, and as such are considered in the calculation. However, no sensitivity analyses were performed for these variables.

3.6.1 SENSITIVITY TO THE WEIGHT FACTORS

In this analysis the workload growth between 1 and 2 tasks was kept at a constant 1.25% for a fixed stiffness of 0.5. To accomplish this, the constant *c*, which parameterises the number of tasks effect, was recalculated for various combinations of the weight factors. This means that a workload of 101.25%, when two tasks were assigned, was always guaranteed for this analysis. If this was not the case no real comparison could be made to how the weight factors influence the growth of PWL. Workday length was set to 8 hours, the total number of hours worked was always adjusted to sum to 8 and all other constants were chosen as defined in section *Component Parameters*.

A graphical representation of valid weight distributions for the following relationship: $F_1 = 1.923 * F_2$ is shown in graph 7. Valid combinations are obtained when weights add up to 1.0, are not negative and the following relationship: $F_1 > F_2 > F_3$ applies. This results in the following valid tuples of (F_1, F_2, F_3): (*0.5, 0.26, 0.24*), (*0.55, 0.29, 0.16*), (*0.6, 0.31, 0.24*) and (*0.65, 0.34, 0.01*).

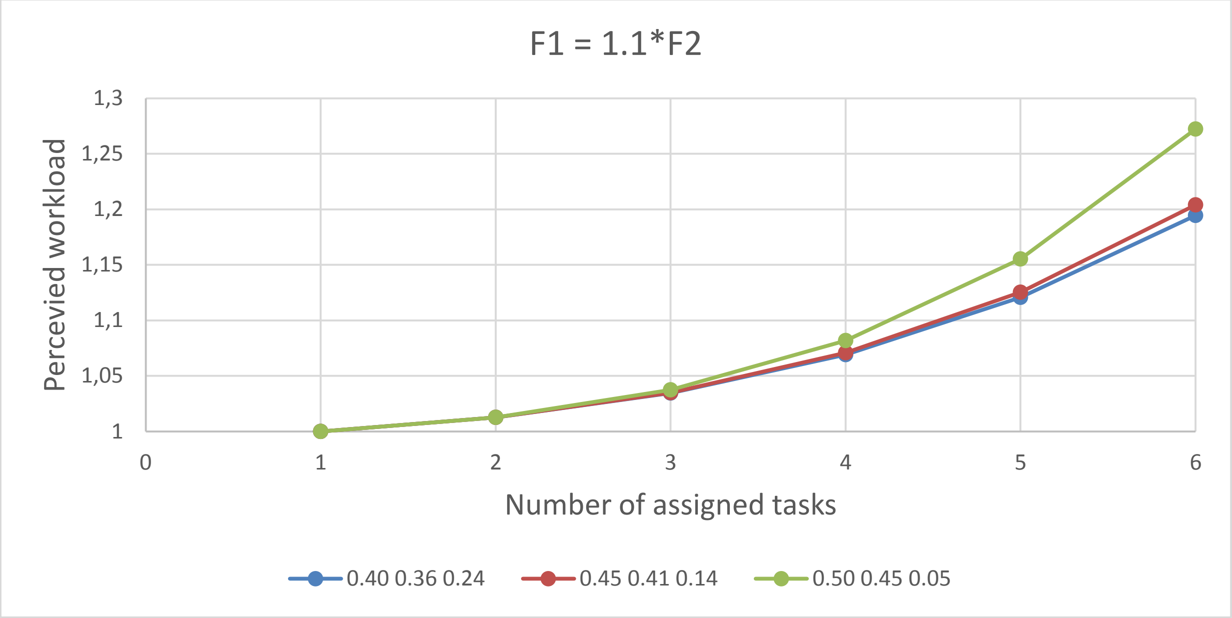


Graph 7: Perceived workload vs number of allocated tasks – F_1=1.923*F_2

Graph 7 indicates the PWL is not sensitive to the weight factor values when the number of tasks are varied, except when F_3 becomes very small. Since F_3 weighs the number of tasks component, small values of F_3 effectively removes this component from PWL and the change in PWL is then the result of the ratio of $F_1 : F_2$.

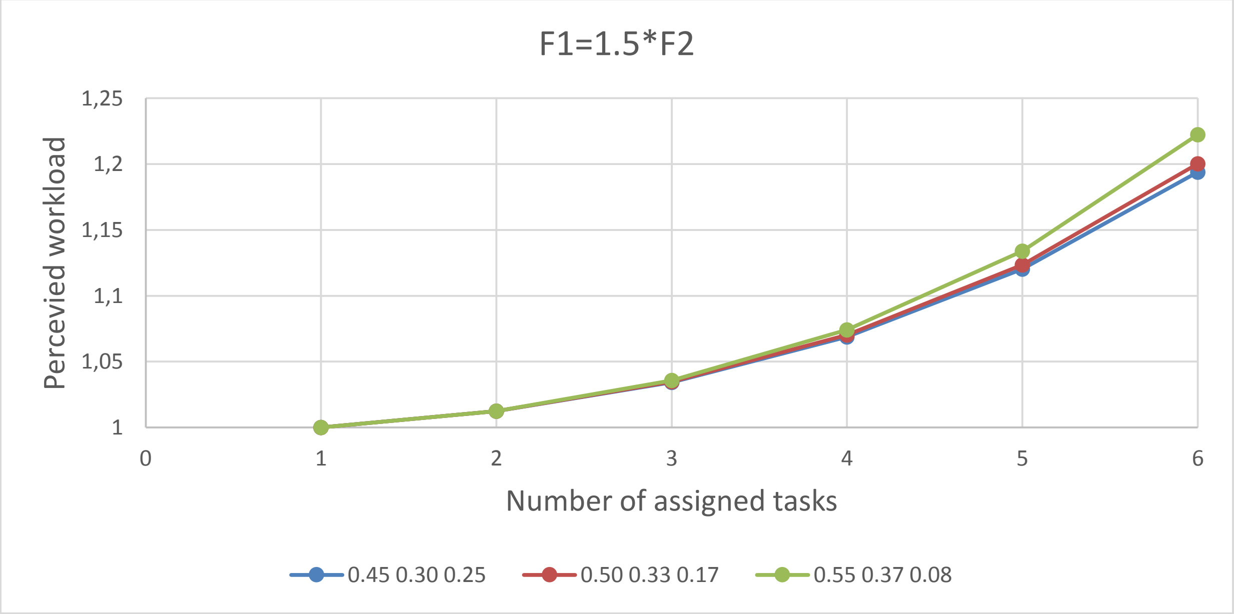
The effect of changing the ratio $F_1 : F_2$ is shown in Graphs 8 and 9. It indicates that PWL is not sensitive to this ratio.

Valid tuples of (F_1, F_2, F_3) for the ratio $F_1 = 1.1*F_2$ are: (*0.4, 0.36, 0.24*), (*0.45, 0.41, 0.14*) and (*0.5, 0.45, 0.5*).



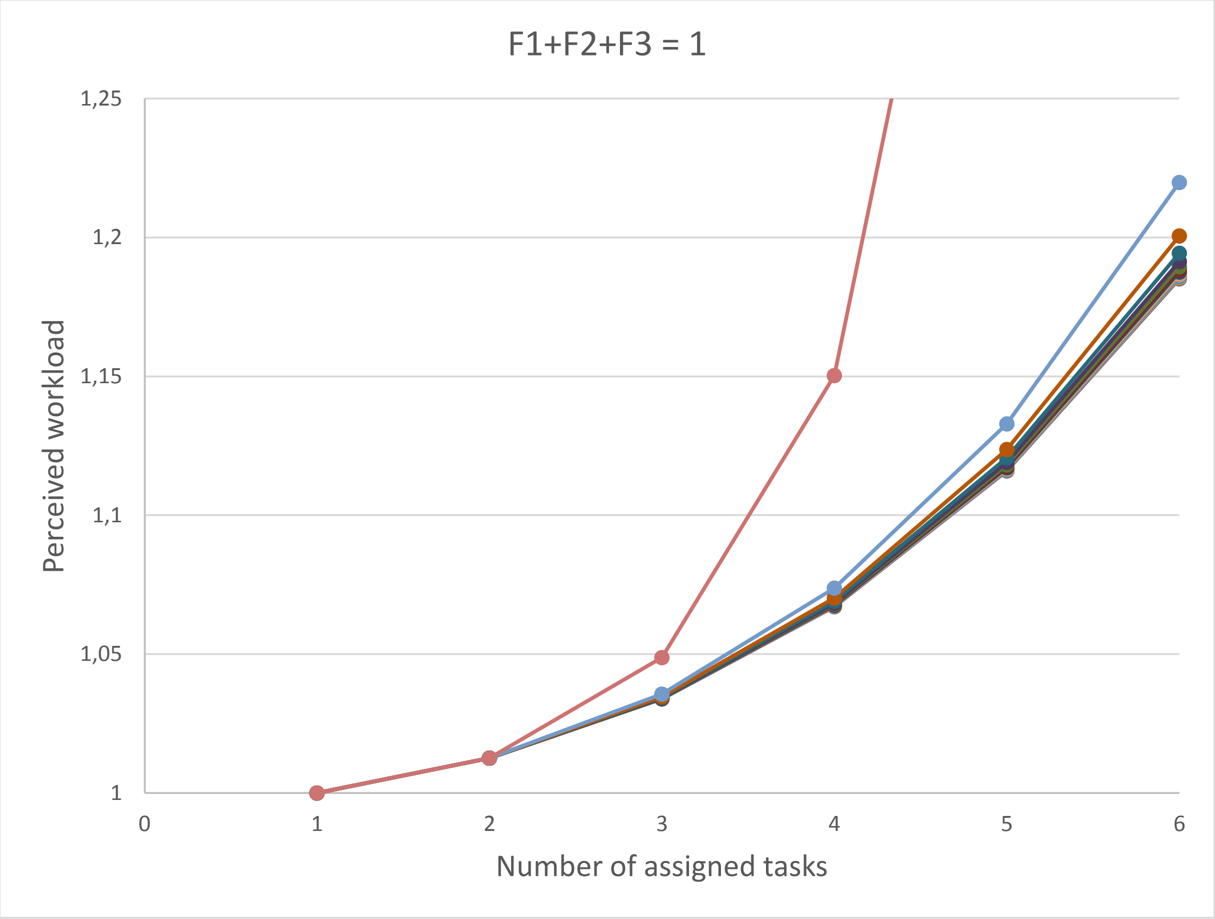
Graph 8: Perceived workload vs number of allocated tasks – F_1=1.1*F_2

Valid tuples of (F_1, F_2, F_3) for the ratio $F_1 = 1.5*F_2$ are: (*0.45, 0.3, 0.25*), (*0.5, 0.33, 0.14*) and (*0.55, 0.37, 0.08*).



Graph 9: Perceived workload vs number of allocated tasks – F_1=1.5*F_2

When the restriction: $F_1 > F_2 > F_3$ is waved, results for the relationship $F_1 = 1.923 * F_2$ are as follows:

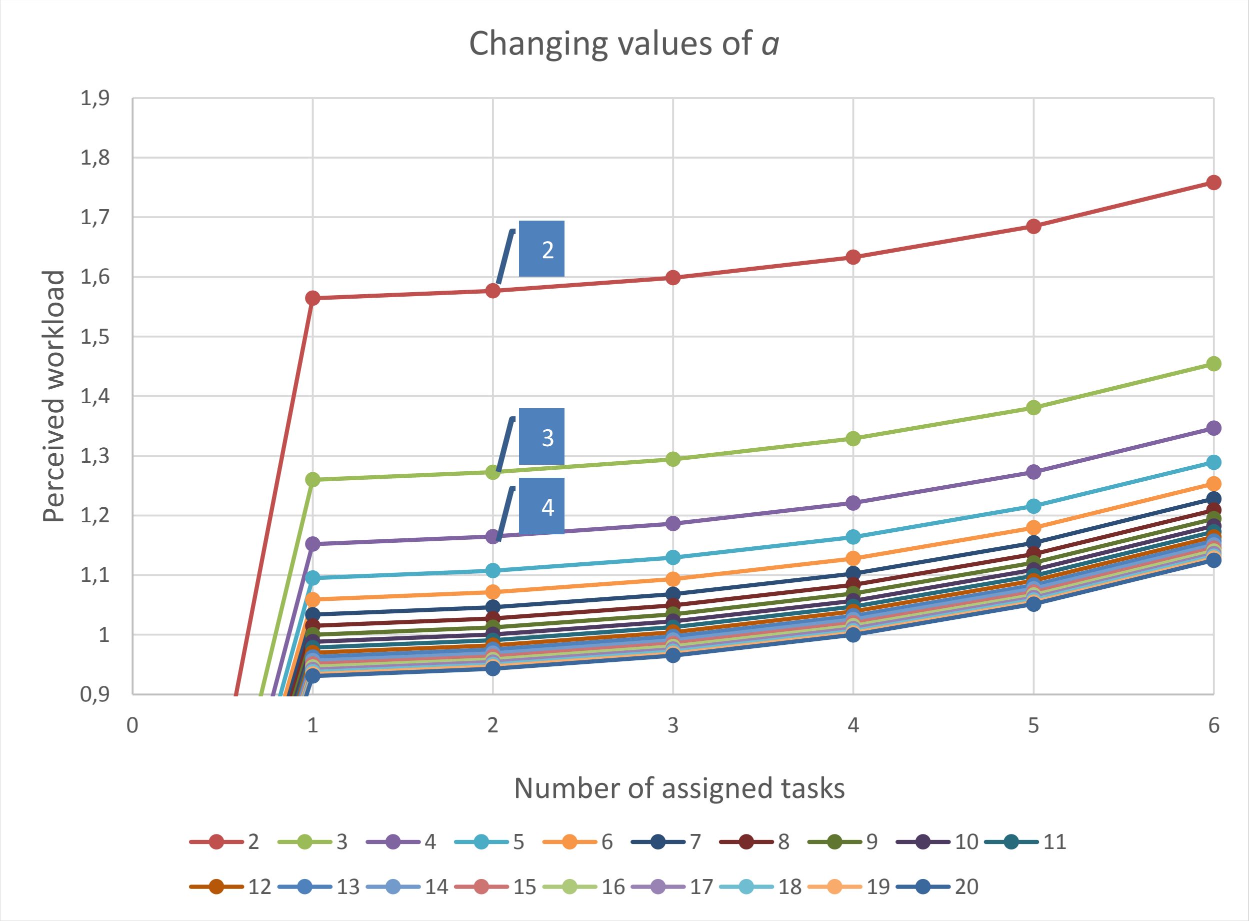


Graph 10: Perceived workload vs number of allocated tasks – no relationship between F_1, F_2 and F_3

Due to the number of results the legend in graph 10 is omitted. The graph shows the same grouping as the previous graphs, indicating that the restriction of $F_1 > F_2 > F_3$ is not critical. This means that the choice of the weight parameters as $F_1 = 0.5$, $F_2 = 0.26$ and $F_3 = 0.24$ were acceptable, since changing these values will only have a small effect on the workload. It was found that whatever the changes in parameter values, the calculated PWL always change in relation to each other. As was previously stated, as long as this is the case, the values remain relevant.

3.6.2 SENSITIVITY TO WORKDAY DURATION LOGARITHM BASE *a*

To test the sensitivity of the logarithm base number controlling the effect of workday duration on PWL, the base number *a* was varied and PWL plotted as a function of the number of assigned tasks. Stiffness of all tasks was fixed at 0.5, workday length was set to 8 hours, the total amount of hours worked was always adjusted to sum to 8 and all other constants were chosen as defined in section *3.5.2 Component Parameters*. Graph 11 shows the PWL for different values of *a*:

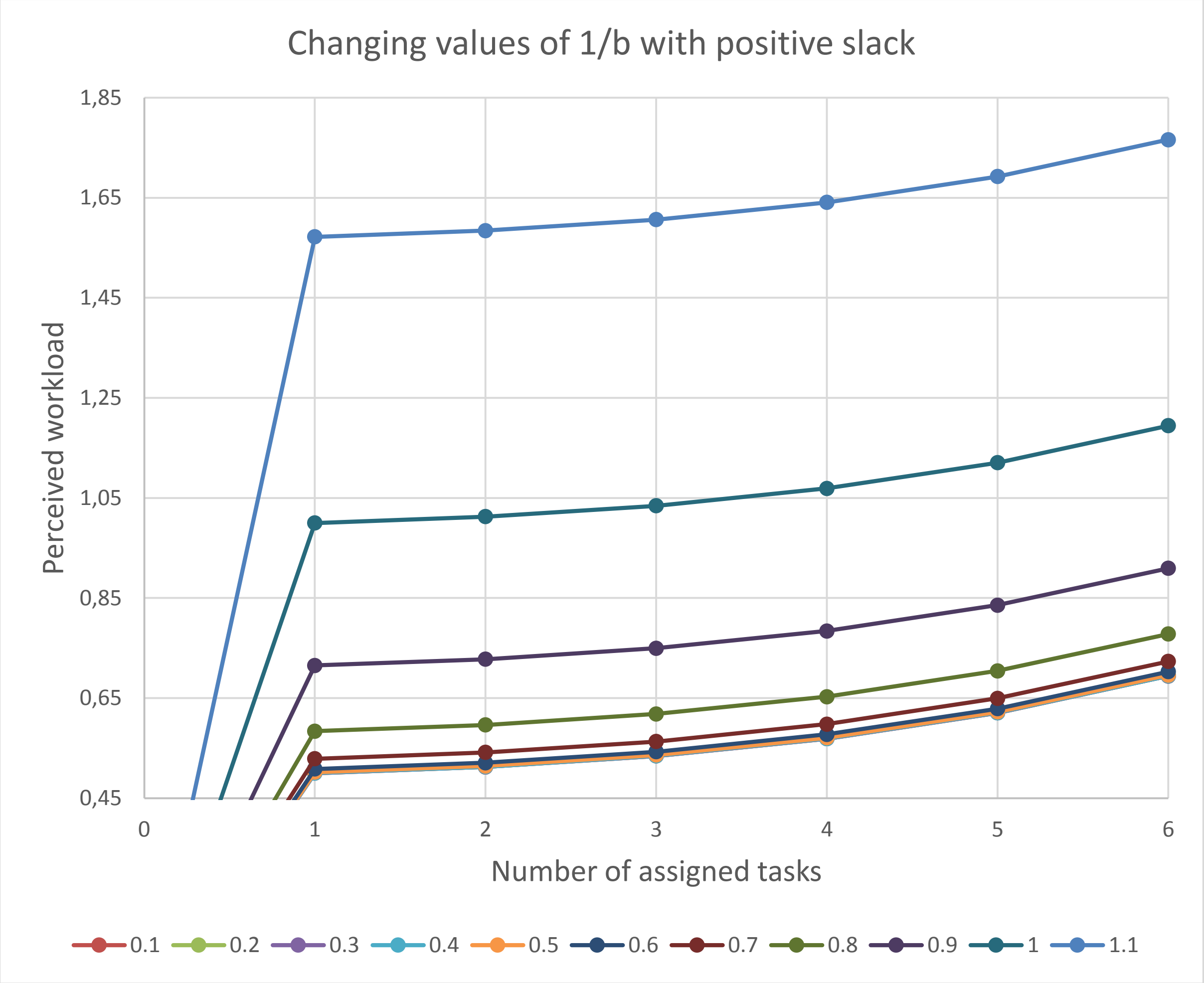


Graph 11: Perceived workload vs number of allocated tasks – changing values of constant a

It is found that the change in PWL diminishes for values of *a* higher than 6. In a previous section *a* was chosen as 9 by way of logical deduction. By looking at this graph it can be seen that if the chosen value was 8 or 10 instead, it would make little to no difference to the PWL. This means that the initial choice of *a* = 9 is acceptable.

3.6.3 SENSITIVITY TO SLACK EXPONENT BASE 1/*b*

To test the sensitivity of the exponent base number controlling the effect of task slack on PWL the exponent base number *b* was varied. Stiffness of all tasks was fixed at 0.5, workday length was set to 8 hours, the total number of hours worked was always adjusted to sum to 7, one hour less than a full day's work, and all other constants were chosen as defined in section *3.5.2 Component Parameters*. Graph 12 shows PWL for different values of *1/b*:

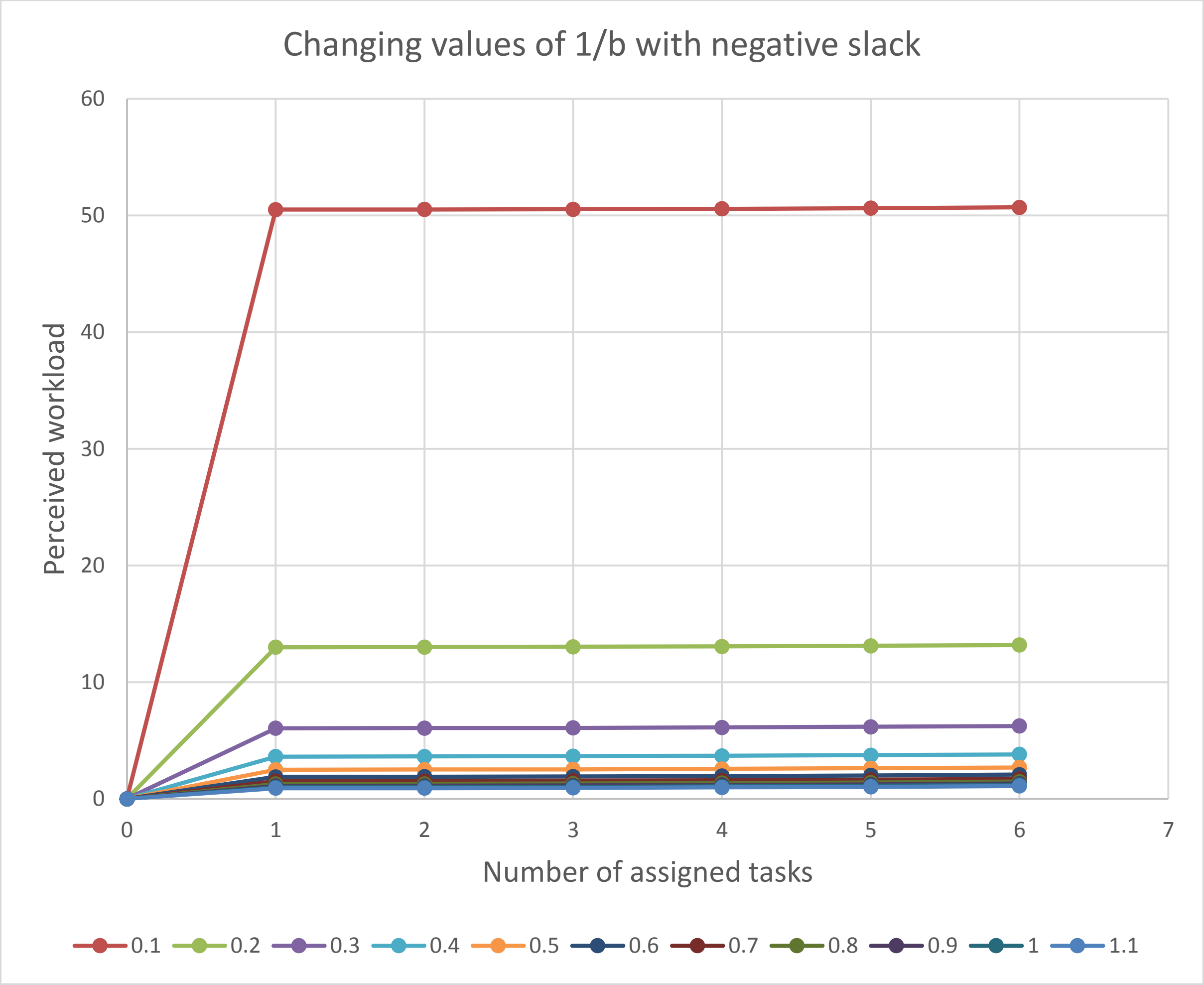


Graph 12: Perceived workload vs number of allocated tasks – changing values of constant 1/b

If the values of *1/b* are varied between 0.1 and 1, results for perceived workload vary between 70% and 180%. This is a big difference and implies that the equation is sensitive to changing this value. Values are however parallel to one another, meaning that chaning the *1/b* is merely a matter of scale. The PWL is less sensitive for low values of *1/b*, i.e. 1/*b* < 0.8. The chosen value of

0.841 is slightly bigger, but was chosen to fulfil a logical need, as described in section *3.5.2 Component Parameters.*

To investigiate the case where overtime is involved, task durations were changed to yield 10 hours, i.e. 2 hours overtime per day.

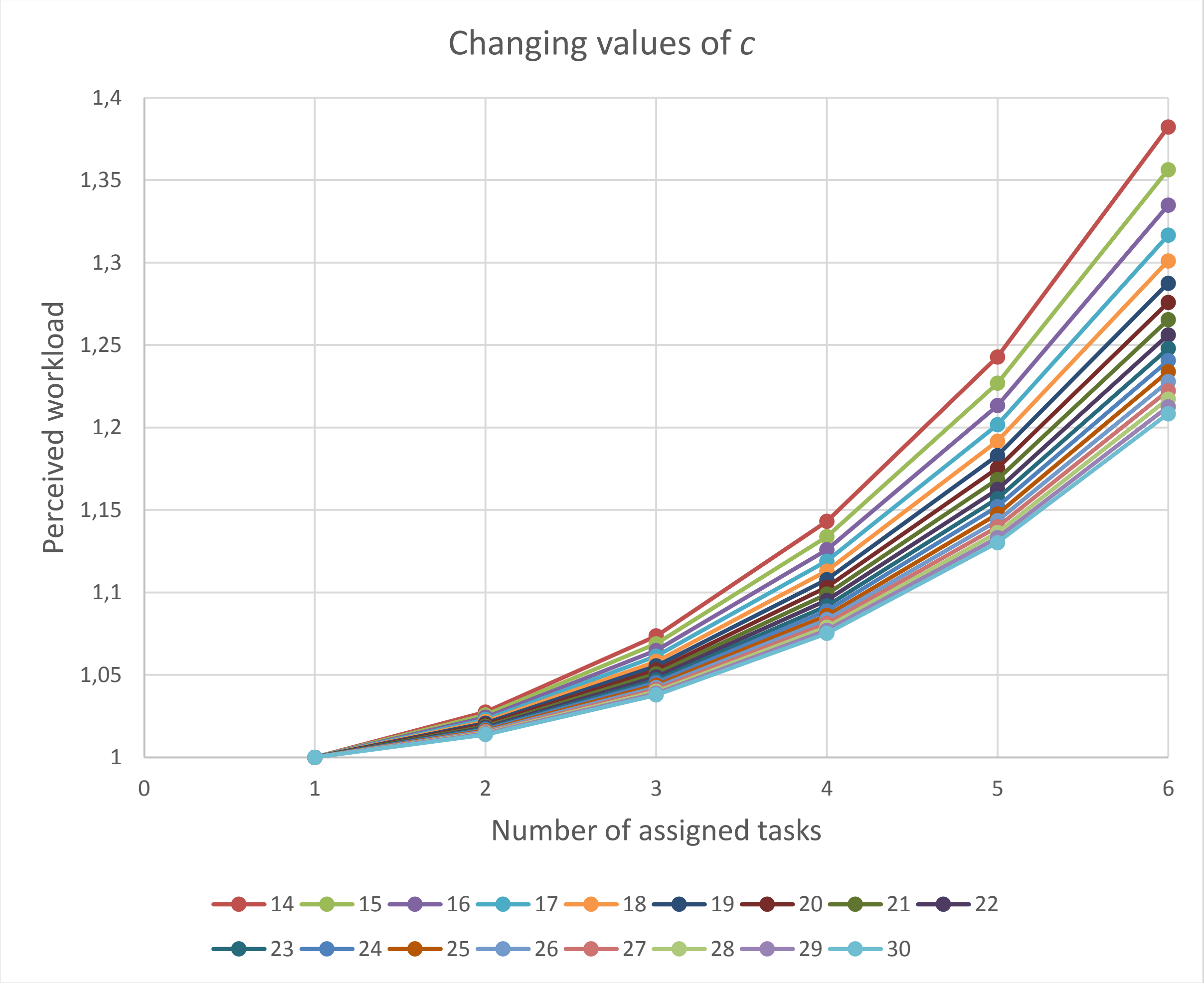


Graph 13: Perceived workload vs number of allocated tasks – changing values of constant 1/b for overtime

If the value of *1/b* is again varied, PWL groups together for values approaching 1, was shown in graph 13, with very little difference between choosing a value of 0.9 or 1. This suggests that if the chosen value of *0.959* is changed slightly, the effect will be insignificant.

3.6.4 SENSITIVITY TO EXPONENT DENOMINATOR *c*

To test the sensitivity of the exponent denominator governing the effect of additional tasks on PWL, the exponent denominator *c* was varied. Stiffness of tasks was fixed at 0.5, workday length was set to 8 hours, the total amount of hours worked was always adjusted to sum to 8 hours, and all other constants were chosen as defined in section *3.5.2 Component Parameters*. Graph 14 shows the results graphically for different values of *c*:

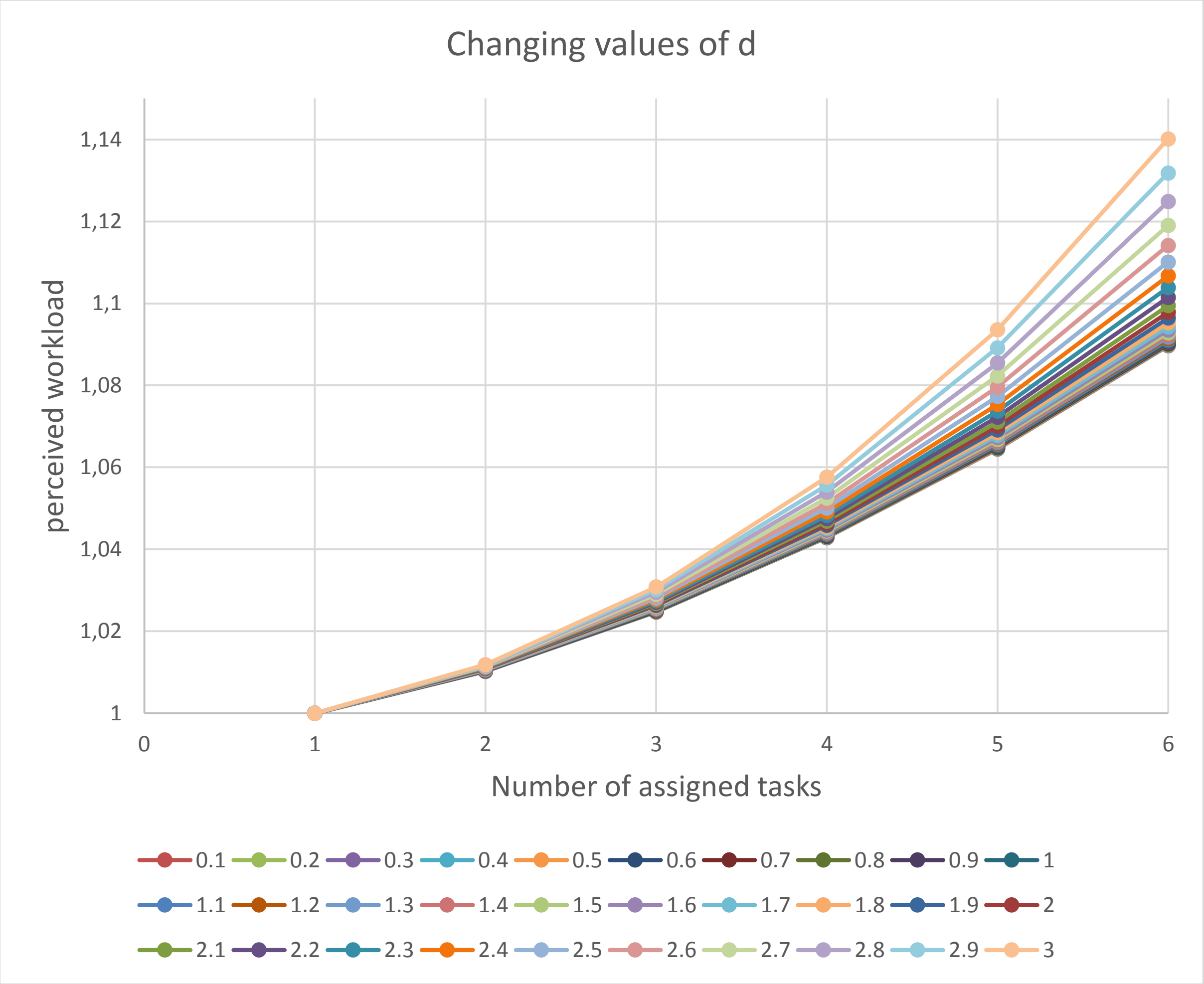


Graph 14: Perceived workload vs number of allocated tasks – changing values of constant c

Graph 14 shows that the greater the value of *c* becomes, the smaller the difference between values become. This means that the chosen value of 33.92 is acceptable and PWL is not sensitive to small changes of this value.

3.6.5 SENSITIVITY TO EXPONENT DENOMINATOR *d*

To test the sensitivity of the exponent denominator governing the growth of task stiffness on PWL, the exponent denominator *d* was varied. Stiffness of tasks was fixed at 0.5, workday length was set to 8 hours, the total amount of hours worked was always adjusted to sum to 8 hours, and all other constants were chosen as defined in section *3.5.2 Component Parameters*. Graph 15 shows the results graphically for different values of *d*:

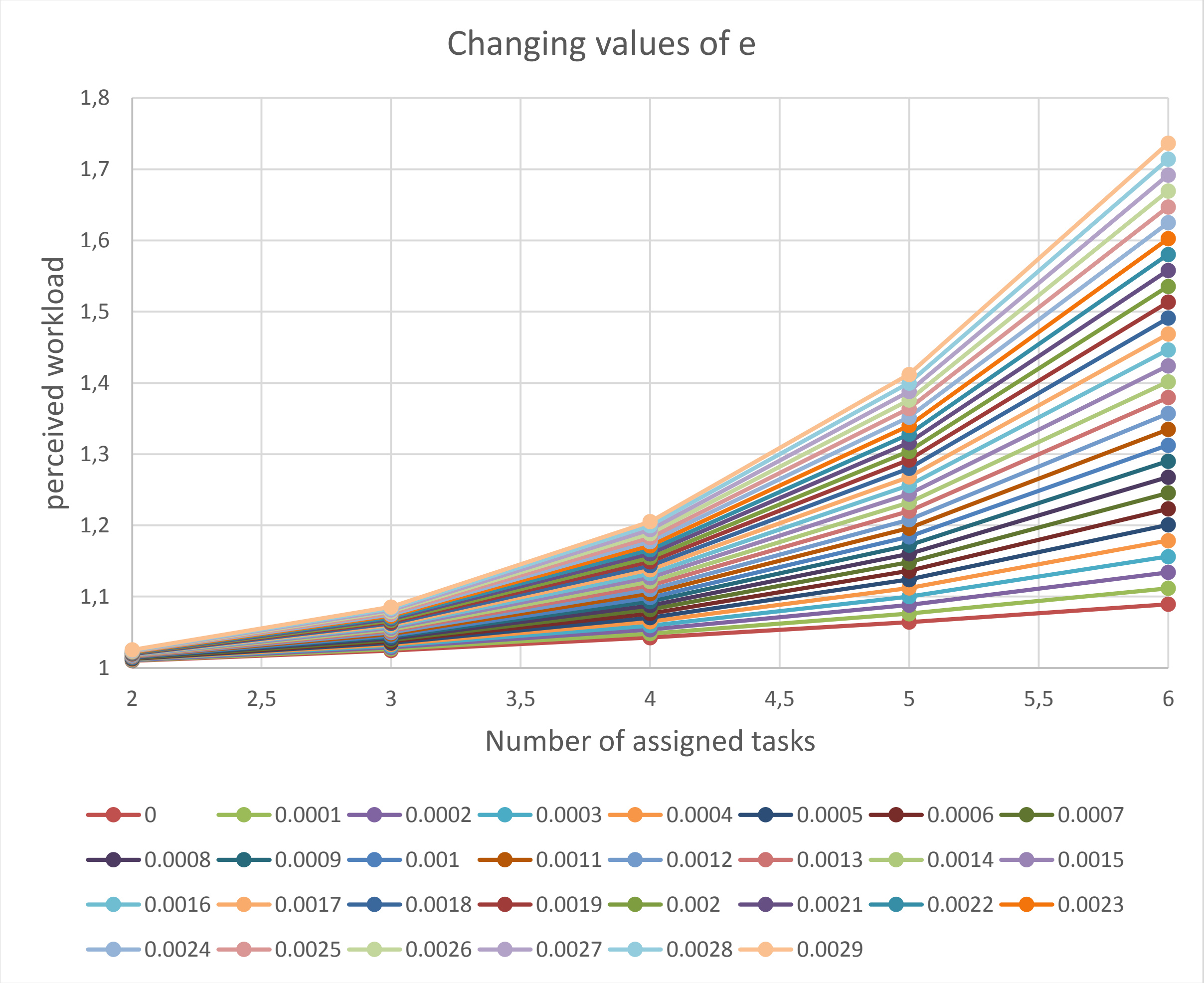


Graph 15: Perceived workload vs number of allocated tasks – changing values of constant d

The results indicate that perceived workload is moderately dependent on the value of *d,* especially if the number of tasks is greater than 4. This is to be expected since the equation was designed to be sensitive for high values of stiffness and tasks. Although PWL is sensitive to this constant, it is still only a matter of scale, and the chosen value of *d* = 3.405 is deemed acceptable.

3.6.6 SENSITIVITY TO TASK STIFFNESS CONTROL *e*

Changing the value of *e*, the constant controlling the effect of stiffness on the equation, has a similar effect as changing the value of d, as shown in graph 16. Graph 16 starts at 2 assigned tasks because the contribution of stiffness is designed to be zero for zero or one assigned tasks.



Graph 16: Perceived workload vs number of allocated tasks – changing values of constant e

The equation is quite sensitive to the value of *e*, indicating that the value should be chosen carefully. The assigned value, *e* = 0.000472, was chosen to conform to values tested against the opinion of engineering managers.

3.7 EVALUATION OF PERCEIVED WORKLOAD

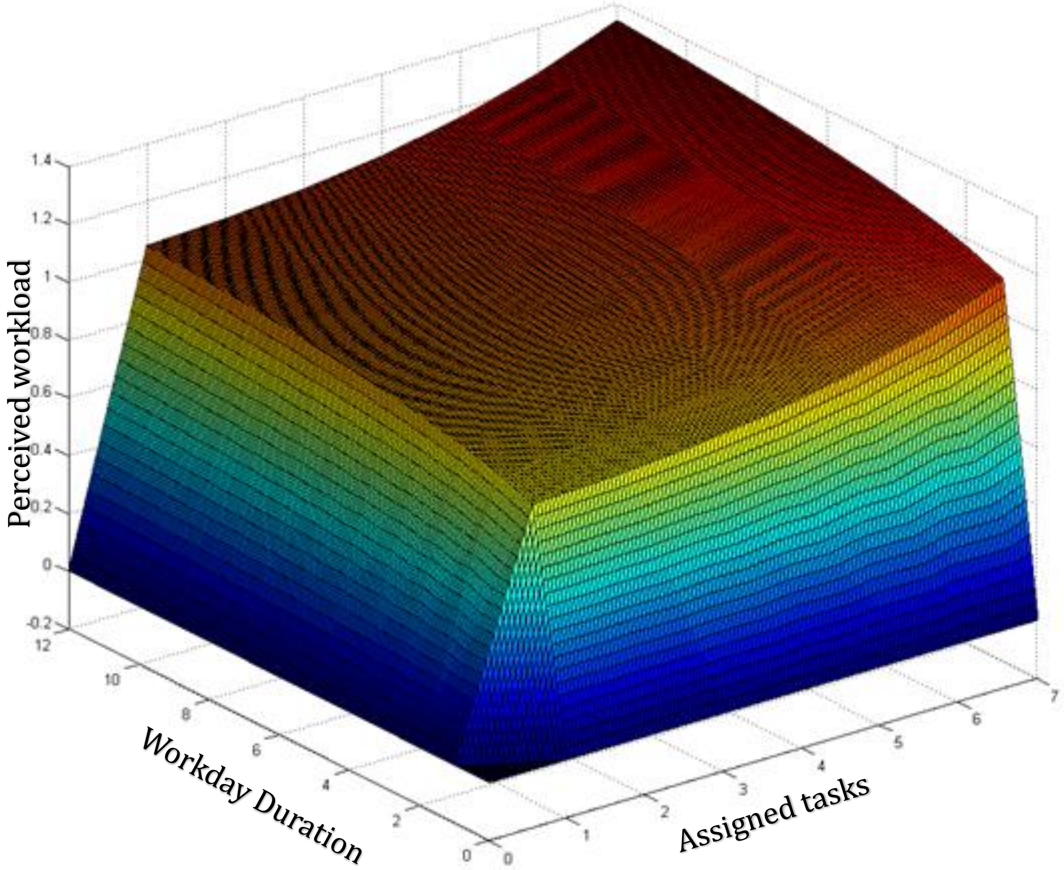
The values of the weight factors and parameters of the PWL-equation 13 have been determined in section *3.5 Choosing constants*. The sensitivity analysis described in the previous section indicated that the values scale the result that is obtained for the PWL, but, for the chosen values, the PWL is proportionally balanced. The variation of the PWL-equation is investigated and discussed in this section.

$$
\begin{aligned}
&\text{perceived workload} \\
&\quad = t_{\text{ineff}} \\
&\quad * \left(\left(\left(\begin{cases} 0, & T = 0 \\ 0, & \displaystyle\sum_{i=1}^{n}(t_i(1+o_i) = 0 \\ 0.841^{(T-\sum_{i=1}^{n}(t_i(1+o_i)}, & \displaystyle\sum_{i=1}^{n}(t_i(1+o_i) \le T \\ 0.959^{(T-\sum_{i=1}^{n}(t_i(1+o_i)}, & \displaystyle\sum_{i=1}^{n}(t_i(1+o_i) > T \end{cases}\right)(0.5)\right.\right. \\
&\quad + (\begin{cases} 0, & L = 0 \\ 0, & n = 0 \\ \log_9(L+1), & n \ne 0 \end{cases})(0.26) + \left(\begin{cases} 0, & n = 0 \\ n/_{33.96}, & n \ne 0 \end{cases}\right)(0.24) \\
&\quad \left.+ \left(\begin{cases} 0, & n \le 1 \\ 0.000472 * s_{max} * n^{3.405}, & n > 1 \end{cases}\right)\right) + \begin{cases} 0, & L = 0 \\ 0, & n = 0 \\ IF, & n \ne 0 \end{cases}
\end{aligned}
$$

Equation 16: Equation final form

3.7.1 ASSIGNED TASKS – WORKDAY DURATION

Individuals that are responsible for a large number of tasks usually have to work longer hours. In graph 17, the PWL is shown as the number of assigned tasks varies between 0 and 7 and the workday duration between 0 and 12 hours. Engineering tasks are typically not trivial and will take a while to complete. If 4 tasks are assigned at 2 hours each it will make up an average workday, because of this it was decided the interesting area is between 0 and 7 assigned tasks. PWL is plotted on the *z*-axis, number of assigned tasks on the *x-axis* and workday duration on the *y-axis*.

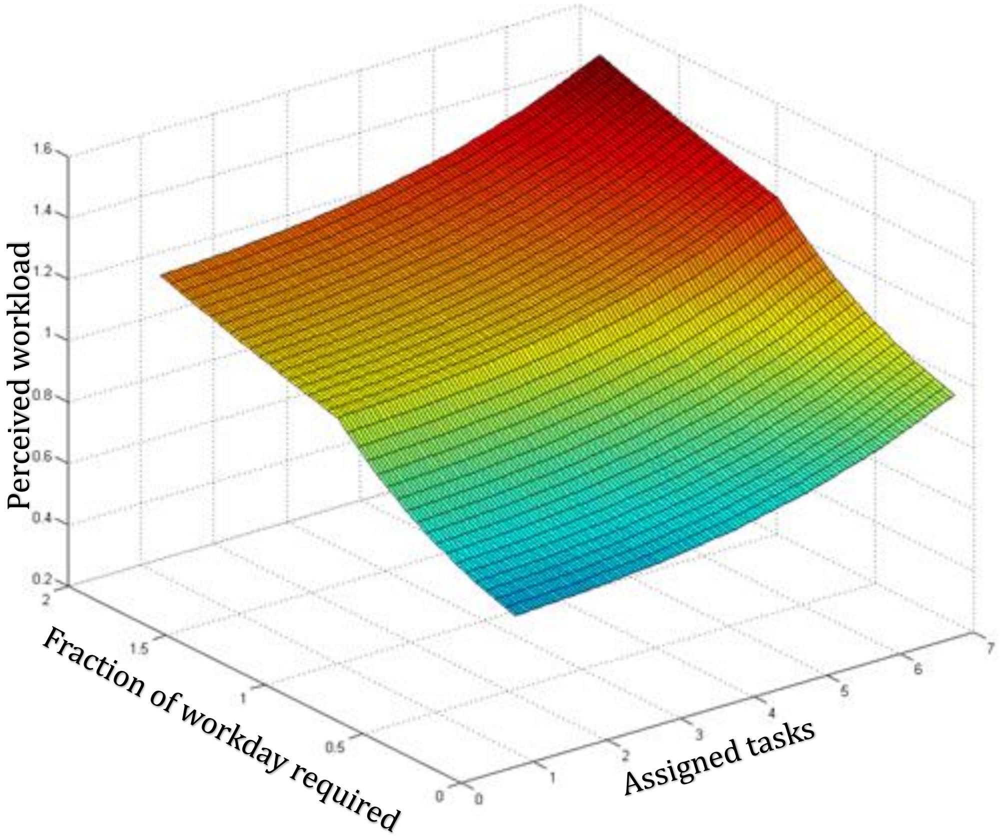


Graph 17: 3D Plot of equation results: perceived workload vs number of tasks vs workday length

By design, PWL is equal to 1 when workday duration = 8 and assigned tasks = 1. Few tasks combined with a short workday results in a PWL less than 1.0. The graph however shows the exponential increase of the resulting PWL as the number of tasks increase to 6 and the workday duration to 12 hours.

3.7.2 ASSIGNED TASKS – PERCENTAGE OF DAY REQUIRED

Even if individuals are responsible for a large number of tasks, it does not necessarily mean it will take a complete day to perform all the assigned tasks. In graph 18, the PWL is shown as the number of assigned tasks varies between 0 and 7 and the fraction of a workday required to perform assigned tasks varies between 0 and 2. A fraction of 2 represents an effective workday length of 16 hours, this variation was limited as is to keep the graph in the interesting area. PWL is plotted on the *z*-axis, number of assigned tasks on *x* and fraction of workday required on *y*.

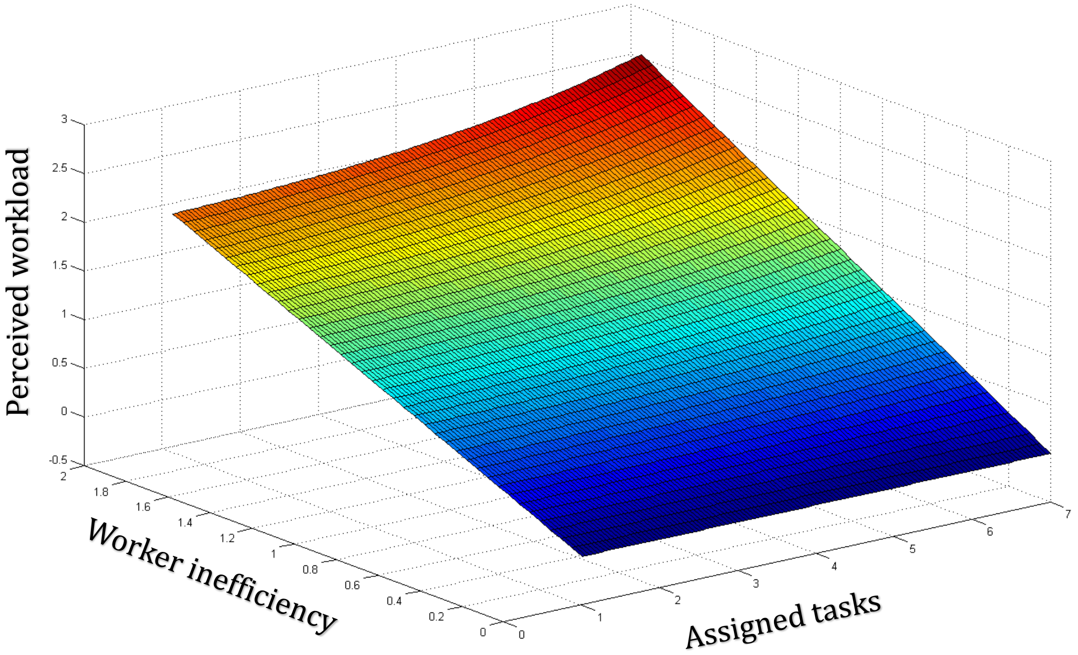


Graph 18: 3D Plot of equation results: workload vs number of tasks vs fraction of workday required

By design, PWL is equal to 1 when fraction of workday required = 1 and assigned tasks = 1. When the fraction-worked varies between 0 and 1, workload is seen to increase exponentially. When the fraction increases further than 1, the surface increase flattens out slightly. A fraction between 0 and 1 implies an individual is working less than a full day's work, while a fraction of more than 1, point to an individual working overtime. The curve is broken up into the two parts to realise a more realistic expression of workload as explained in a previous section.

3.7.3 ASSIGNED TASKS – WORKER INEFFICIENCY

In graph 19, the PWL is shown as the number of assigned tasks varies between 0 and 7 and the individual's inefficiency varies between 0 and 2. PWL is plotted on the *z*-axis, number of assigned tasks on *x* and individual inefficiency on *y*.



Graph 19: 3D plot of equation results: workload vs number of tasks vs worker inefficiency

By design, PWL is equal to 1 when fraction of workday required = 1 and worker inefficiency = 1. Both an increase in tasks and inefficiency results in an increase in workload. As can be seen in the graph, the effect of an individual's inefficiency is linear while increasing number of assigned tasks has an exponential effect.

3.8 SURVEY OF ENGINEERING MANAGERS' OPINIONS

A survey was set up and sent out to members of the engineering community, with focus on individuals with real project management responsibilities. The survey was primarily intended to test their opinion regarding the assumption that workload should increase when the number of assigned tasks is increased, even if the combined hours to complete the tasks were constant. The concept that worker inefficiency should be directly proportional to workload was also tested.

A scenario was presented, with a few stated workload values. In all cases an 8 hour workday was assumed.

When a scenario reads: "1 Task of 8 hours", it means that an individual has one task assigned to him for a given day and that task is estimated to take 8 hours to complete.

Please select your most appropriate workload value for all of the following scenarios:

Scenario Description	***Please highlight the most appropriate workload percentage***		
1 Task of 8 hours.	95%	100%	105%
2 Tasks of 4 hours each.	100%	101.5%	115%
5 Tasks, durations add up to 8 hours.	100%	112%	124%
1 Task of 8 hours, but person works twice as fast as everybody else.	50%	75%	100%
1 Task of 8 hours, but person works twice as slow as everybody else.	100%	200%	400%
3 Tasks of 1.33 hours each.	25%	52%	70%
10 short tasks, each just 24 minutes long. (4 Hours)	50%	82%	100%
1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.	50%	70%	100%

The results of the survey are summarised in the table below:

Scenario Description	***Number that voted for suggested workload percentage***		
1 Task of 8 hours.	**(9)** 95%	(7) 100%	(3) 105%
2 Tasks of 4 hours each.	**(9)** 100%	(7) 101.5%	(3) 115%
5 Tasks, durations add up to 8 hours.	(6) 100%	**(8)** 112%	(5) 124%
1 Task of 8 hours, but person works twice as fast as everybody else.	(8) 50%	**(9)** 75%	(2) 100%
1 Task of 8 hours, but person works twice as slow as everybody else.	(3) 100%	**(15)** 200%	(1) 400%
3 Tasks of 1.33 hours each.	(2) 25%	**(10)** 52%	(7) 70%
10 short tasks, each just 24 minutes long. (4 Hours)	(4) 50%	**(11)** 82%	(4) 100%
1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.	(5) 50%	**(12)** 70%	(2) 100%

Results 1: Survey Results

3.8.1 SURVEY CONCLUSION

According to the survey, people believe that one task that takes up an entire work day, provides an individual with a workload of a bit less than 100%. For the survey conducted the participants selected a value of 95%. This is interesting because per definition the workload in that case should be a full 100%. The same people voted for a workload of 100% for two tasks, and for 112% for 5 tasks. The fact that they feel a single task provides a workload of 95% indicate that they feel task durations have an inherent safety factor imbedded into their duration. People then voted that two tasks have a combined workload of 100%, which indicates that the safety factor has dissolved. The 112% for five back-to-back tasks not only confirms the hypothesis that increasing the number of tasks increases the workload, but also roughly quantifies the effect of the number of tasks on perceived workload. The fact that the 112% was selected by participants in the survey, and not 124% indicates that the results calculated with the workload formula are acceptable.

The equation for calculating workload assumes a linear relationship between efficiency and workload. People who completed the survey however said, that they believe an individual assigned one task that lasts a whole work day, but who works twice as fast as everybody else is loaded with a workload of 75%. Which does not match up to the expected workload value of 50%. These same people agreed, quite convincingly, that if an individual would work twice as slow as everybody else he would be loaded with 200%, which again confirms the linear relationship. The difference between the numbers of people who voted for the 75% versus 50% was not significant, but it could be possible that the relationship should stay linear for efficiency values larger than one, but get slightly exponential with smaller values. The linear relationship was however kept as is.

In conclusion, the survey indicates general agreement with the assumptions made when calculating individual office workload. Where the people disagreed, it was by a very small margin. It was thus decided to leave the calculation method as is.

4. DESIGNING THE OBJECT MODEL

A model is an abstract image of a selected part of the world. It is essential for engineers to be able to map specific snapshots of the world into such models. The key is knowing exactly what information to store, and what should be omitted. Models consist of component objects, and the part of the world that is mapped to these objects depends on the purpose for which a model is constructed. Every object is unique and can be distinguished from all other objects in the model. An object is also identifiable by a unique identifier, such as a name or number which differs from all other names or numbers in the model. Objects have attributes which define specific properties of the object, and can be the identifier of another object. Any number of attributes can be stored in the attribute-set of an object, it depends on the properties of the world that are to be mapped to the object.

An object is characterised by its state, in the form of its attributes and their values.

Expressed mathematically, if the set of objects that represents the state of a model is called the object set M, then:

$$M := \{a \ \in M \mid a \textit{ is an object of the model}\}$$

Equation 17: Definition of object set

Furthermore, object modelling requires the classification of the objects [21]. An object is an instance of a class. In object-oriented programming, a class is a construct that is used as a blueprint or template to create objects of that class.

The programming language Java uses this object oriented paradigm to implement object models, and as such is very suitable to be used as a tool to create the model. [22]

In this chapter an overview of the object model's components is presented. This provides an indication of the functionality that can be expected of the prototype implementation. Furthermore it forms the basis of the data model that is described in the next chapter.

4.1 COMPONENTS OF THE OBJECT MODEL

The allocation of tasks takes place within the context of an engineering company, its staff and projects that are executed according to planned schedules.

Specific components of a model that accounts for this context are now conceptualised. These concepts are transferred to an object model for the software implementation.

Although the focus is on task allocation, it will be shown that useful, additional functionality can be obtained from the task allocation information. The model is extended to support the implementation of these functionalities.

4.1.1 TASKS

Tasks are the most important components that the model have. In scheduling, tasks are the activities that create deliverables. Some tasks require certain deliverables of other tasks before they can be started. For example, a foundation has to be constructed before the walls of a house can be built. The relationships between tasks and deliverables are used to create a graph in the set of tasks. From this graph a schedule can be created and optimised. The schedule is used by the task allocation software under consideration. It was decided not to store the deliverable data of any individual task, but only the tasks themselves and how the tasks fit together in relation to one another, i.e. the task graph. The task graph can be used to aid in task allocation, ensuring personnel usage is spread evenly across the project, and that individuals do not have to complete multiple tasks back to back. Each task is mapped to only one project.

Figure 1 illustrates that a project is a collection of tasks which each require one or more skills to complete.

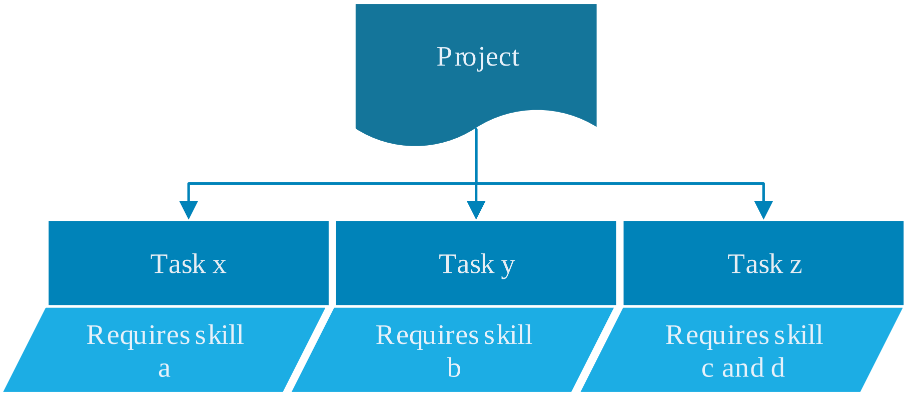


Figure 1: Project and task relationship

Along with the task graph, the schedule also discerns earliest start dates and latest end dates for each task. These dates are dependent on task durations and the "has-to-be-executed-before" relation in the set of tasks. The tasks' durations, and start and end dates can then be used to calculate the latest date a task has to be started before detrimentally affecting the rest of the project. A task can consequently be flagged as more important than others when its latest start date approaches.

Another important aspect of tasks is the fact that not everybody can complete every task. Tasks require a specific skill set of the people assigned to complete it. This skill set can subsist of multiple individual skills or roles within the industry.

4.1.2 RESOURCE: PERSONNEL

Personnel is in general the only resource considered when allocating tasks in an engineering design office. The solution can be expanded to include other resource constraints, e.g. financial considerations, office space availability or software licenses. Licenses will very rarely be a real constraint that keeps employees from working concurrently and can be solved very easily by obtaining more licences. Other possible constraints are considered inconsequential for the typical use of the allocation technique, where all possible personnel are already settled in offices with the organisation being able to pay salaries.

There are two types of personnel accounted for in this model, internal personnel and external personnel. Internal personnel are permanent employees of the organisation. External personnel are temporary personnel that are hired to perform a specific task or work on a specific project. They are also known as outsource personnel. Each member of the internal personnel is assigned to a specific department and a specific office. External personnel are only hired on a temporary basis, and as such do not work in any particular department or office.

Both types personnel have a rates ascribed to them. A rate describes how much the company pays an individual for doing hourly work, also how much the company earns from having him/her do work. Having a reasonably comprehensive account of personnel rates is important for both task allocation and creating cost estimates.

Personnel, internal and external, have certain skills ascribed to them. Skills are expertise or roles within the industry attributed to individuals. “Cleaner”, “Draftsman” or “Civil Design Expert” are all examples of *skills*. Multiple skills can be ascribed to an individual.

Figure 2 illustrates the relationship between skills and personnel, as well as rates and personnel.

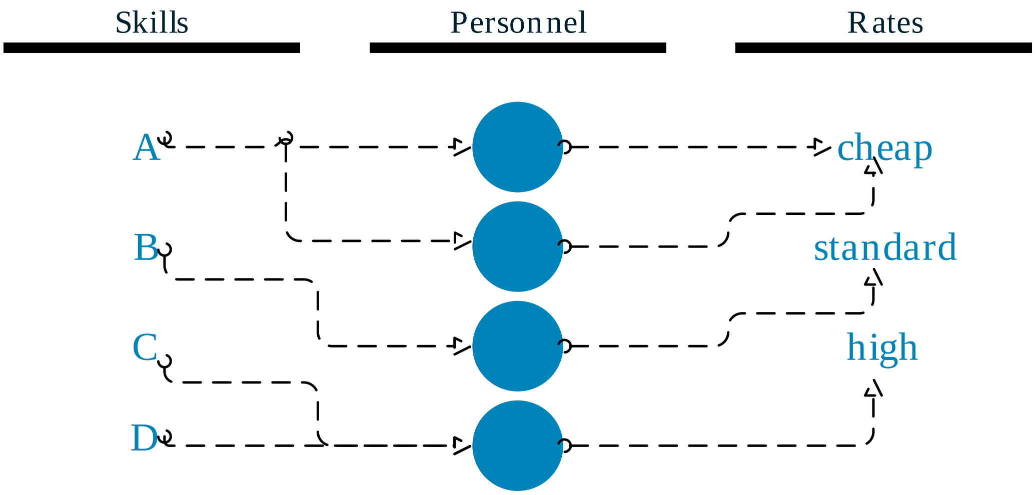


Figure 2: Personnel, skills and rates relationships

4.1.3 LOCATION

Locations are an important aspect of the model to conceptualise. Projects occur, or are completed, at a specific location. This automatically links all tasks attributed to a specific project to its location. Engineering offices are the other important concept that have locations, and with all personnel being ascribed to an office, individuals store the location of their office. This capacity is significant because it links where individuals work to where they have to complete tasks.

Locations consist not only of a description, e.g. a city name, but also of actual geographical coordinates. These coordinates are used to accurately calculate the distance between distinct locations, which may play a role in the task allocation procedure.

4.1.4 OFFICES

An organisation is considered to be a collection of offices, each with a unique location, departments, and workforce. Different combinations of departments can be accredited to an office, and an individual can then work in one of these departments.

Figure 3 illustrates how offices and projects are located in specific locations, and how members of specific offices can be assigned to different projects.

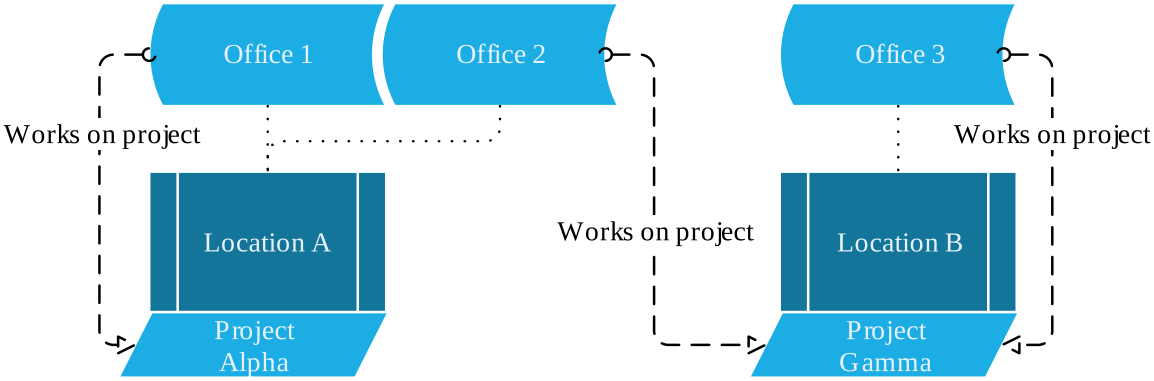


Figure 3: Locations

The organisation structure within the office can also be modelled by defining employee to superior links. If the link only connects an individual with his immediate superior, the whole command structure can be envisioned, ending with the CEO of the organisation who has no superior.

4.1.5 FILTERS

Filters are at the root of the task allocation procedure since they are used to select appropriate personnel. It is not necessary to store filters in the data model, except to build a history of which filters were used when realising a project's task allocations.

Combinations of filters are called a filter configuration and can be stored when task allocations are made. Filter configurations can be included in project cost quotations so that a specific allocation can be recreated.

4.1.6 ALLOCATION

A task allocation, or booking, is made when a task is assigned to an individual with the understanding that he/she should complete it. It can never be assured that a task will be completed perfectly and on time, but the risk of the individual failing to perform is significantly lower when the allocation process is done properly.

When an initial task allocation is made, it is at first only a provisional booking. This provisional booking can then be manually upgraded to a permanent booking when the project manager is satisfied with the choice. The provisional booking system also prevents accidental concurrent bookings for multiple projects while allocation is still in progress.

If only one project was executed at a time, the task allocations could be done systematically and optimally. Unfortunately this is not how the majority of the engineering community goes about business. Most engineering design offices work on multiple projects concurrently, and this will always be the case. This means that when allocating tasks, project managers may want to book the same individual for different projects. The provisional booking system will allow possible concurrent bookings to be evaluated and the most pressing one to be accepted.

Figure 4 illustrates the process gone through by a task to select the most appropriate personnel allocation. Different types of filters are discussed in chapter 6, Enhancing the Model.

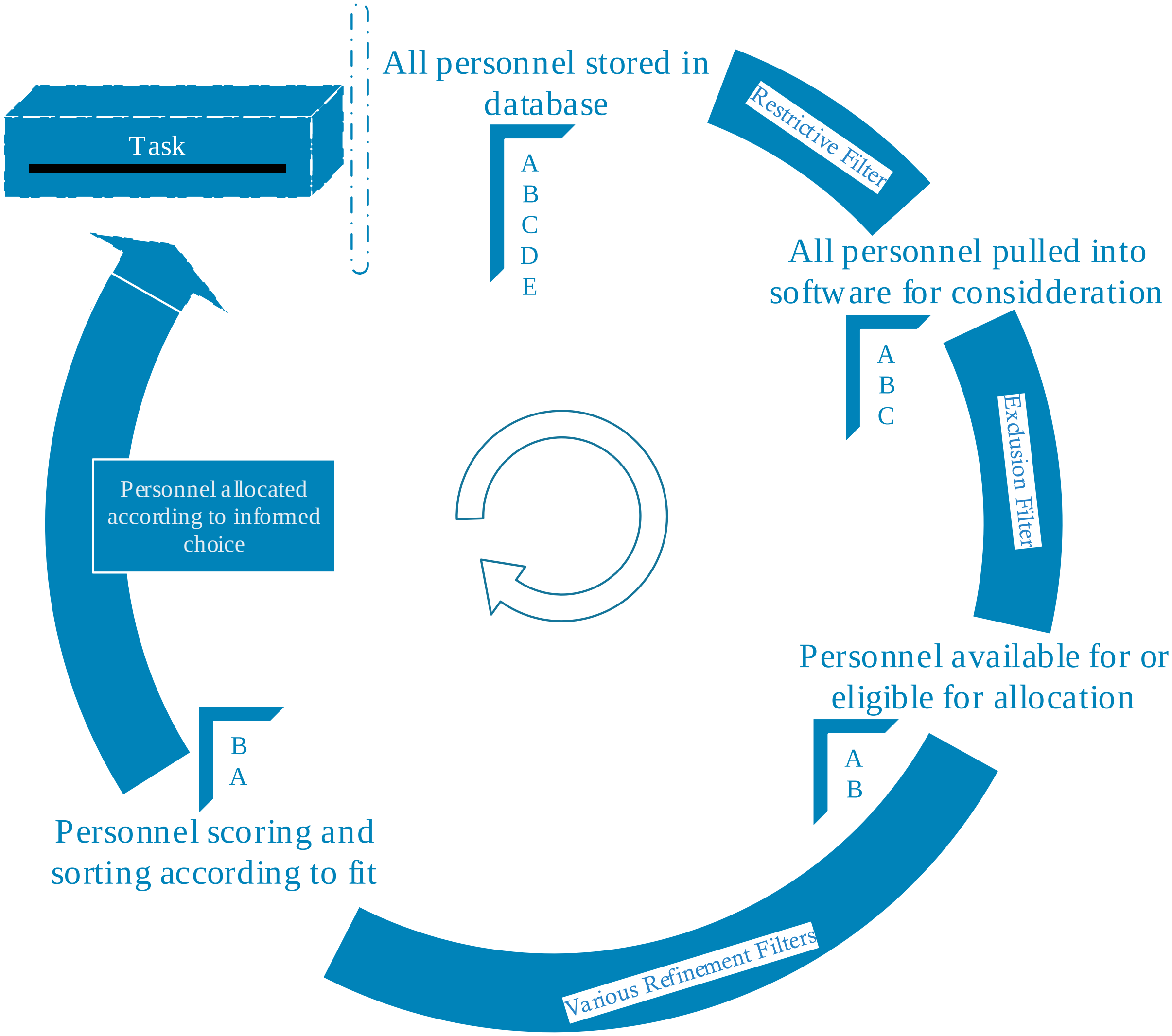

Task
All personnel stored in database
A
B
C
D
E
Restrictive Filter
All personnel pulled into software for considderation
A
B
C
Exclusion Filter
Personnel available for or eligible for allocation
A
B
Various Refinement Filters
Personnel scoring and sorting according to fit
B
A
Personnel allocated according to informed choice

Figure 4: Process of task allocation

4.1.7 COMPUTATION OF PROJECT COST

The ability to create a quotation or budget for how much a project is estimated to cost is very useful. It can be used to provide an estimated price for a client, or to judge whether the organisation should take a project on. Since the model has the task allocation data at its disposal, a fairly accurate quotation can be worked out.

A *quotation resource* can be created at the time of quotation, which creates a copy of an existing human resource. This copy is necessary to maintain an accurate history of the project. For example, if an individual's rate changes, the organisation would be unable to accurately calculate the work done or the money owed without the copy.

It is also useful to know how exactly task allocations were done. For that purpose the filter configuration is also stored inside a quote. With all the information in place in the data model, a project can be recreated at any time.

Multiple provisional allocations can be made and quotations used to compare employee allocation strategies with one another before finally deciding on the strategy that fits the organisation best.

5. DATABASE MODELLING

Data models are developed to support the structured storage of data used in information systems. If a well-structured data model is used to store and access data, then different applications can share data seamlessly. This requires that the mathematics behind the model logic and structure to be well defined and the database well organised.

Various database technologies exist, of which the relational database has proven to be the most popular choice. Databases are used to store sets of data. A data type as used in a typical relational database might be a list of integers, a few dates or true or false statements. The relational theory behind the data structure does not dictate what data types are supported, this is something decided by function and design.

In relational data models a table corresponds with a set, and a record corresponds with an element. If a database table were to have three records, the table can be thought of as a set with each record being one of the three elements in it. These records represent instances of tuples. In mathematics a tuple is an ordered list of elements. In set theory an *n*-tuple is a sequence of *n* elements, where *n* is a non-negative integer. In database theory the relational model uses a tuple definition similar to tuples as function, but with each element identified by a distinct name, called an attribute, instead of a number. A tuple in relational models is formally defined as a finite function that maps attributes to values [23].

Relational databases are made up of tables. Tables consist of columns and records. A column specifies one single piece of data that a record has. This piece of data can be new data, unique to the record, or data found in another table in the database. A record is uniquely identified by its primary key, which is an entry in one of its columns or a combination of entries in more than one column. Columns that are not part of the primary key are called the non-key columns.

The entire database can be visualised with an entity-relationship diagram, or ERD, which shows the tables the database consist of along with the relationships between them. In a later section the final modelled database's ERD is shown to give a complete overview of how all the database tables connect to form the information network that stores the model.

5.1 DATABASE NORMALISATION

Normalisation is the process of restructuring the logical data model of a database to eliminate redundancy, organise data efficiently and to moderate the potential for anomalies during data operations. Data normalisation may also improve data consistency and simplify future extension of the logical data model. A non-normalised data structure may suffer from data anomalies by storing data representing a particular value in multiple locations. An update to such data could result in inconsistent data. A normalised database prevents such an anomaly by storing data in only one location. Similarly, such redundancies in non-normalised databases can hinder deletion.

Database normalisation consist of conforming a database to a few rules, called normal forms. A database is considered normalised once it achieves the Third Normal Form.

5.1.1 FIRST NORMAL FORM

In mathematics, a singleton, also known as a unit set, is a set with exactly one element. The term is also used for a *1*-tuple, a sequence with one element [24]. An indicator function is a function defined on a set *X* that indicates membership of an element in a subset *A* ⊆ *X*, having the value 1 for all elements of *A* and the value 0 for all elements of *X* not in *A*.

If *S* is a mathematical class defined by an indicator function

$$b: X \to \{0,1\}$$

Then *S* is called a singleton if and only if:

$$b(x) = (x = y) \mid x \in \mathrm{X}, \mathrm{y} \in \mathrm{X}$$

Equation 18: Singleton definition

By definition, the First Normal Form is a relation in which the intersection of each row, *R*, and column, *C*, contains one and only one value, so that:

$$\{R \cap C \coloneqq attribute \mid attribute \in S\}$$

Equation 19: First Normal Form definition

5.1.2 SECOND NORMAL FORM (2NF)

The Second Normal Form is a relation that is in First Normal Form and where every non-primary key attribute is fully functionally dependent on the primary key.

If:

- *a* is an attribute of a record,
- *K* a set containing only the primary key,
- *nK* a set containing all the non-key values then the following relationship holds:

$$a \in nK => \bigwedge_{a} \bigvee_{f} a = f(K)$$

Equation 20: Second Normal Form definition

5.1.3 THIRD NORMAL FORM (3NF)

The Third Normal Form is a relation that is in First and Second Normal Form, and in which no non-key attribute is transitively dependent on the primary key [25]. This means that any non-key attribute may only be dependent on its primary key and not on any other non-key attribute.

If a relation $K \rightarrow nK_1$ exists, then there may not exist a relation $nK_1 \rightarrow nK_2$, as that would indicate nK_2 is transitively dependent on K via nK_1.

The Third Normal form can be expressed as follows:

$$\{a \in \mathrm{nK} \,|\, a = f(b \in \mathrm{nK} \setminus \{\mathrm{a}\})\} = \{\}$$

Equation 21: Third Normal Form definition

5.2 TABLES

Each record inside a database table consists of three parts: A primary key, foreign keys and non-key attributes. A foreign key is a referential constraint between two tables. It constrains the column to what values it may contain and enables cross-referencing between tables. This builds the database network. All foreign keys in this database references primary keys which are integers as a rule. Non-key attributes are independent values which do not link to any tables and only adds information to the database entry. [26]

The tables that make up the relational database of the task allocation application will now be examined and explained. Primary keys are denoted as **PK**[n] in the graphical representation of the tables. Foreign keys are shown as ending in *_fk***.** Columns that do not end in either *_fk* or *_pk* denotes a non-key attribute. Some tables model concepts or classes. These tables contain the bulk of the information. Other tables only link tables together to form relationships inside the database, called linking tables. All linking tables are in principle mathematical mappings of entries to other table entries.

The database tables are an important part of the model. It is believed that should another researcher ever wish to recreate the database, he/she should be able to do so using the descriptions provided below. The tables are discussed in no specific sequence.

[n] Column names also end in _pk

5.2.1 LOCATION

LOCATION is a table referenced by multiple other tables, and details an exact location. The **longitude** and **latitude** are optional columns, used to calculate the distance between two points around the curvature of the earth. The method used to do this is explained in chapter 6.2.4 Refinement Filters.

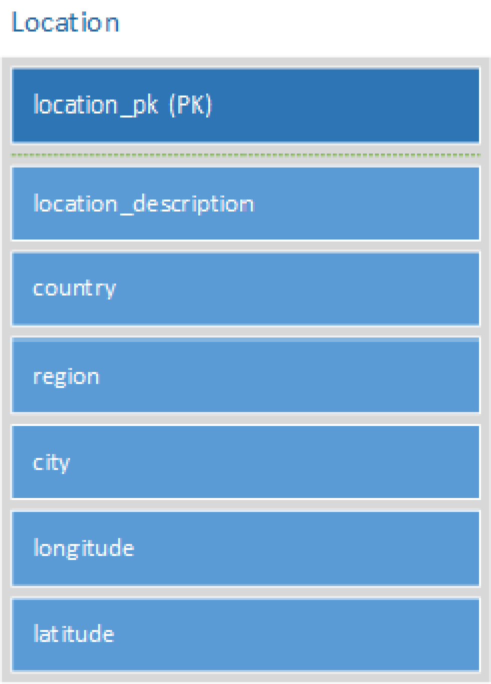
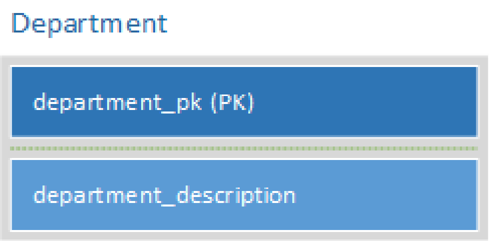


Table 1: LOCATION layout

Column	Data type	Function
location_description	String	Text describing the location, and providing an identifiable name.
country	String	Text identifying the country where the location is situated.
region	String	Text identifying the region inside the country.
city	String	Text identifying the closest city to the location.
longitude	Double	Double value that stores the longitude of the location.
latitude	Double	Double value that stores the latitude of the location.

5.2.2 DEPARTMENT


Department

department_pk (PK)

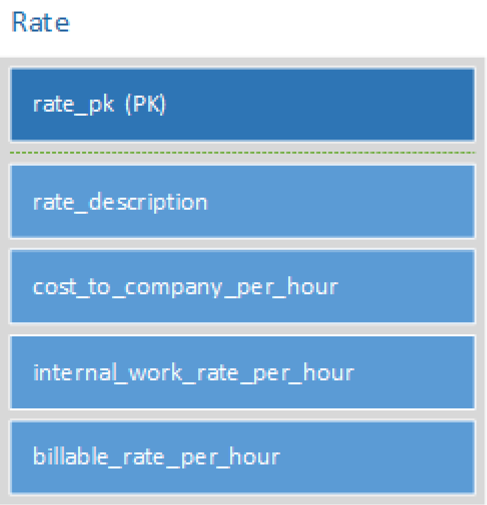
department_description

Table 2: DEPARTMENT layout

DEPARTMENT is used to describe the different segments an office is broken up into, for example “Structural Department”, or “Technical Drawings”. Every employee of an organisation is assigned to one of these departments. A single office may also only be one single department.

Column	Data type	Function
department_description	String	Text stating the department’s name.

5.2.3 RATE


Rate

rate_pk (PK)

rate_description

cost_to_company_per_hour

internal_work_rate_per_hour

billable_rate_per_hour

Table 3: RATE layout

RATE is a table used to fully describe an individual’s salary details as well as the amount the employee will generate for the company. The table comprises a column that describes the rate type and columns of double values for Cost to Company, internal work rate and billable work rate. Cost to Company[o], is the salary package of an employee. It indicates the total amount the employer is spending on an employee per hour including overhead cost. Internal work rate is how much it will cost the organisation if the individual is doing in-house work. Internal work rate would realistically be greater than Cost to Company, as you have to pay the employee for his skills, but less than the Billable rate as a discount is expected for doing work inside the same company. Billable rate is the hourly rate the employee will generate for the organisation by doing external work.

[o] CTC

If the employee is an external human resource, the billable, internal work and cost to company rates could possibly be the same, excluding a possible mark-up to the billable rate to allow for administration costs.

Column	Data type	Function
rate_description	String	Text describing the rate in words. E.g. Standard Rate.
cost_to_company_per_hour	Double	Value detailing the individual's hourly Cost to Company.
internal_work_rate_per_hour	Double	Value detailing the individual's hourly internal work rate.
billable_rate_per_hour	Double	Value detailing the individual's hourly billable rate.

5.2.4 TASK

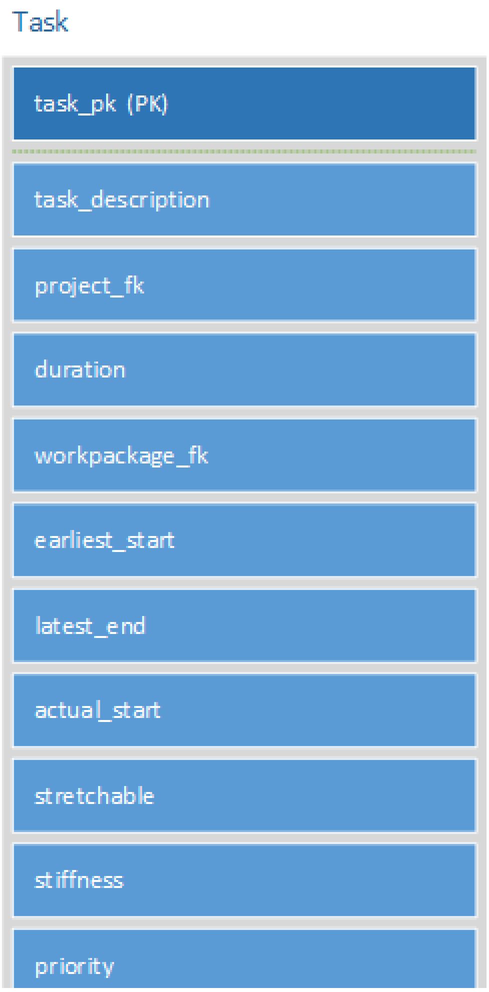


Table 4: TASK layout

TASK is one of the most central and important tables in the database. This table is where all the tasks that have been completed, and those yet to be completed are stored. Tasks have to be stored in the data model. Most of the attributes required by tasks can be stored in a single database entry. However, concepts such as task overhead and task skills are stored in separate tables and linked to tasks using relationship links in the database. Tasks belong to a specific project and is also packaged into a specific work package. **earliest_start** is a timestamp attribute detailing the earliest date the task may start. This is dependent on the end date of the tasks preceding this one. All tasks are presumed to be scheduled according to their precedence requirements. **latest_end** is the latest date the task can end before influencing tasks that come after this specific task. **actual_start** only contains data if the task has already started. **priority** is a numerical value between 0 and 3, with 0 being the most important, and 3 being the least. When a task is created, it will initially have a priority value of 3. The level can then be manually set when it becomes clear to the project manager that the task requires more attention. When a task's **priority** reaches 0, it implies the task is fast-tracked. Fast-tracking will be discussed in chapter 6.3, Tasks and Flat-lining. **stretchable** is a true or false value stating whether a task can be broken up into smaller pieces, or whether it should be completed in one sitting.

Column	Data type	Function
task_description	String	Text stating task's name or description.
project_fk	Integer	Foreign key reference to an entry in the PROJECT table. Connecting this task to a specific project.
duration	Double	A task's statistical time to complete in hours.
workpackage_fk	Integer	Foreign key reference to an entry in the WORKPACKAGE table. Connecting this task to a specific work package.
earliest_start	Timestamp	The earliest date the task can be started.
latest_end	Timestamp	The latest date the task has to be finished without having a detrimental effect on the project.
actual_start	Timestamp	Empty if the task still has to start. The date the task was finally started.
stretchable	Boolean	True or False value, stating whether the task is stretchable.
stiffness	Double	Decimal value between 0 and 1.0, stating the task stiffness.
priority	Integer	Value between 0 and 3, declaring task priority.

5.2.5 SKILL

SKILL is used to describe the different skills individuals have, as well as the skills that are required to do specific tasks. Both HUMANRESOURCEs and TASKs can have multiple SKILLs linked to them. A typical skill would be: "Civil Engineer".

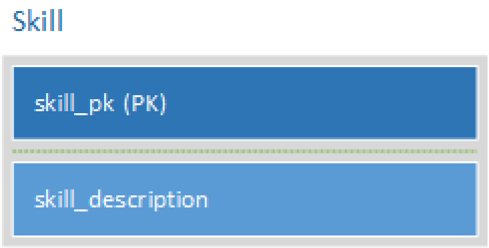


Table 5: SKILL layout

Column	Data type	Function
skill_description	String	Text giving the skill name

5.2.6 SKILL_TO_TASK

SKILL_TO_TASK links an instance in the SKILL table to an instance in the TASK table. In essence, tagging a task as requiring, among others, a specific skill to complete. Multiple skills can be tagged to the same task.

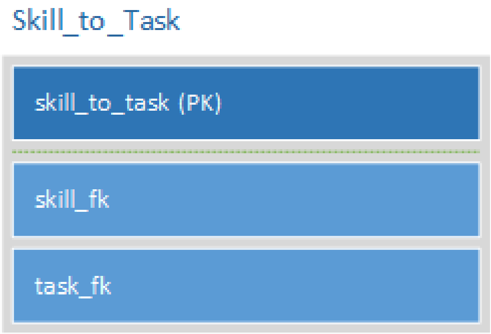


Table 6: SKILL_TO_TASK layout

Column	Data type	Function
skill_fk	integer	Foreign key reference to an entry in the SKILL table. Referencing a specific skill and tagging the task as requiring an individual with this skill.
task_fk	Integer	Foreign key reference to an entry in the TASK table. Referencing a single task.

5.2.7 SKILLTAG

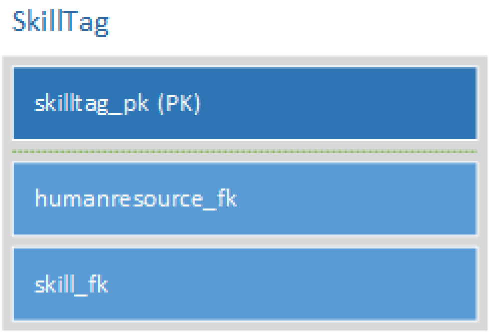


Table 7: SKILLTAG layout

SKILLTAG, similar to SKILL_TO_TASK, links instances of SKILL to entries in HUMANRESOURCE. In essence it has an individual as having a specific skill set. Multiple skills can be linked to the same individual if he/she is experienced in many things. This table is essential in the task allocation procedure, as one of the most important aspects is being able to identify who is able to complete a given task

Column	Data type	Function
humanresource_fk	Integer	Foreign key reference to an entry in the HUMANRESOURCE table. Referencing an individual.
skill_fk	Integer	Foreign key reference to an entry in the SKILL table. Referencing a specific skill, and tagging an individual as possessing said skill.

5.2.8 WORKPACKAGE

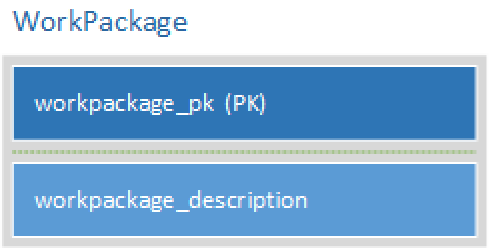
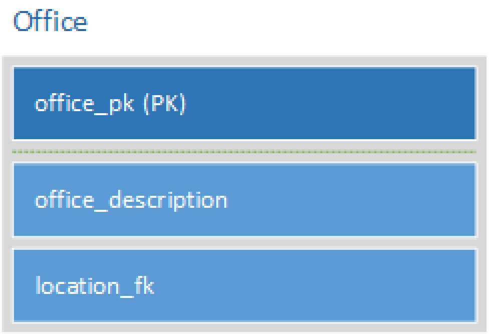


Table 8: WORKPACKAGE layout

WORKPACKAGE is a table describing a set of tasks that form a functional unit, for example the tasks involved in foundation design. Each task can be assigned to a work package, as described in the TASK table. These packages can further be useful to organise task allocation by enabling grouped individuals to be assigned a set of tasks.

Column	Data type	Function
workpackage_description	String	Text stating the work-package's name

5.2.9 OFFICE

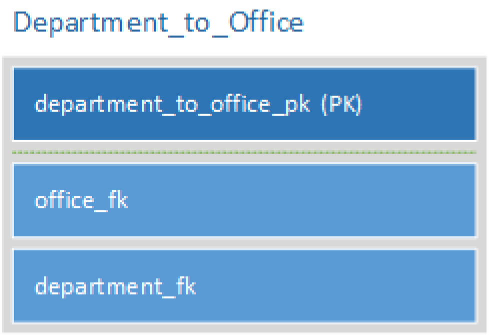


OFFICE is used to describe a normal office. Every organisation comprises of a number of offices. Every office is defined by a unique description, and a LOCATION.

Table 9: OFFICE layout

Column	Data type	Function
office_description	String	Text describing the office.
location_fk	Integer	Foreign key reference to an entry in the LOCATION table, giving the location of the office.

5.2.10 DEPARTMENT_TO_OFFICE

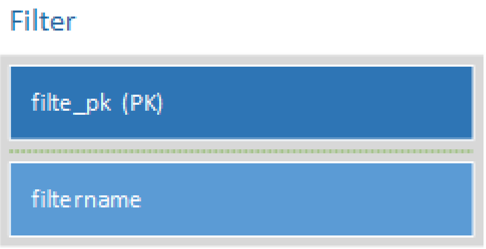


DEPARTMENT_TO_OFFICE is a table that stores no new information, but only exists to link multiple instances of OFFICE tables with multiple instances of DEPARTMENT tables. This table is necessary due to database normalisation rules. It enables a single office to have either one or multiple departments under its wing.

Table 10: DEPARTMENT_TO_OFFICE layout

Column	Data type	Function
office_fk	Integer	Foreign key reference to an entry in the OFFICE table.
department_fk	Integer	Foreign key reference to an entry in the DEPARTMENT table, linking a specific department type to an office.

5.2.11 FILTER



Filter

filte_pk (PK)

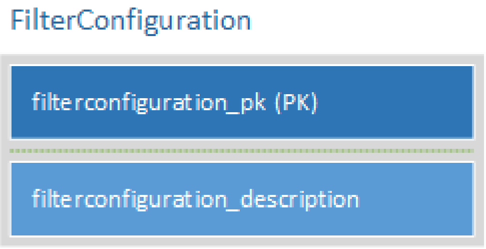
filtername

Table 11: FILTER layout

FILTER is a table used only for storing different configurations of filters. These stored configurations will then later be used to remember how a user made specific task allocations for quotation purposes. **filtername** is a String that provides the filter with an identifiable name.

Column	Data type	Function
filtername	String	Text describing the filter, and providing an identifiable name.

5.2.12 FILTERCONFIGURATION



FilterConfiguration

filterconfiguration_pk (PK)

filterconfiguration_description

Table 12: FILTERCONFIGURATION layout

FILTERCONFIGURATION is a table used in the storing of different combinations of filter settings. These stored configurations will then later be used to remember how a user made specific task allocations for quotation purposes.

Column	Data type	Function
filterconfiguration_description	String	Text giving short description of configuration

5.2.13 FILTER_TO_FILTERCONFIGURATION

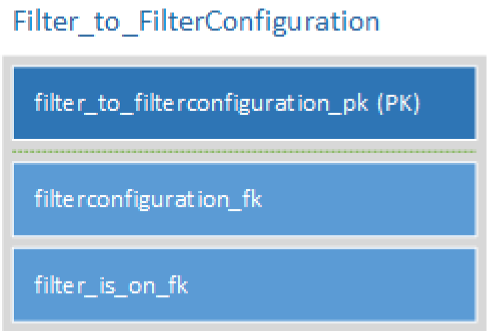
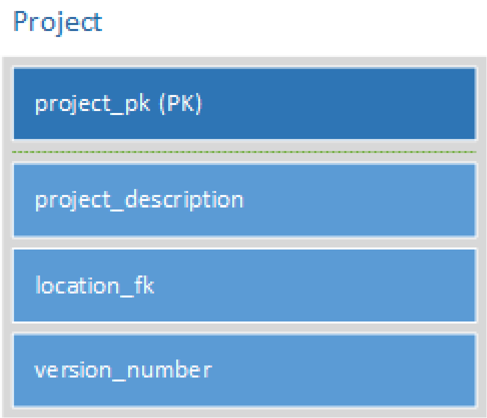

Filter_to_FilterConfiguration
filter_to_filterconfiguration_pk (PK)
filterconfiguration_fk
filter_is_on_fk

Table 13: FILTER_TO_FILTERCONFIGURATION layout

FILTER_TO_FILTERCONFIGURATION, like all linking tables, contains no new information and only exists to link multiple table entries with one another. In the case of this table the purpose is to link an entry in the FILTER table with an entry in the FILTERCONFIGURATION table. Each instance of a filter-configuration is made up of different filters being turned either on or off. FILTER_TO_FILTERCONFIGURATION stores all the filters that are turned on in a specific filter-configuration.

Column	Data type	Function
filterconfiguration_fk	Integer	Foreign key reference to an entry in the FILTERCONFIGURATION table.
filter_is_on_fk	Integer	Foreign key reference to an entry in the FILTER table. Meaning that the specific filter is turned on and is consequently linked to the entry in FILTERCONFIGURATION.

5.2.14 PROJECT



PROJECT purveys the concept of project as defined in chapter *4*. An instance of PROJECT has a description, a location and a version. The versioning is in place for version control. It is also useful when a project manager wants to rearrange task allocations to find an optimal quote. When a new version is created all the other links in the database chain will have to be recreated.

Table 14: PROJECT layout

Column	Data type	Function
project_description	String	Text stating project name or description

location_fk	Integer	Foreign key reference to an entry in the LOCATION table, giving the location of the project.
version_number	Double	Number stating the version of the project, e.g. “1.2”

5.2.15 PROJECTTEAM

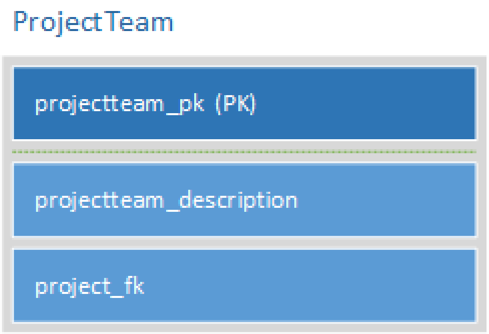


Table 15: PROJECTTEAM layout

PROJECTTEAM is a table enabling the storage of project teams. Project teams are linked to a specific project, and can comprise of any number of human resources, see table HUMANRESOURCE_TO_PROJECTTEAM. This is used to keep a record of people who worked on a project. Project teams can also be used to aid in task allocation by trying to place team members on parallel tasks.

Column	Data type	Function
projectteam_description	String	Text stating project team name or description.
project_fk	Integer	Foreign key reference to an entry in the PROJECT table, linking the team to a specific project.

5.2.16 HUMANRESOURCE

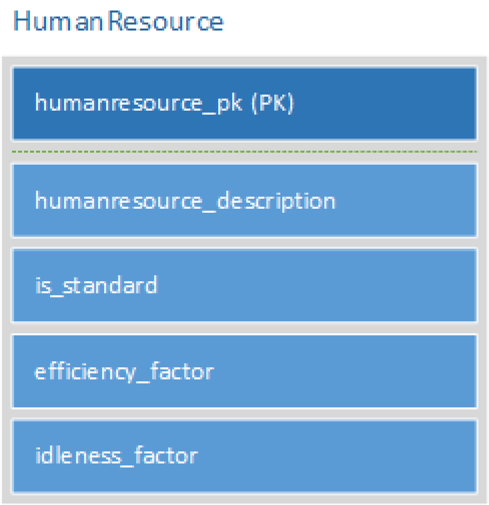
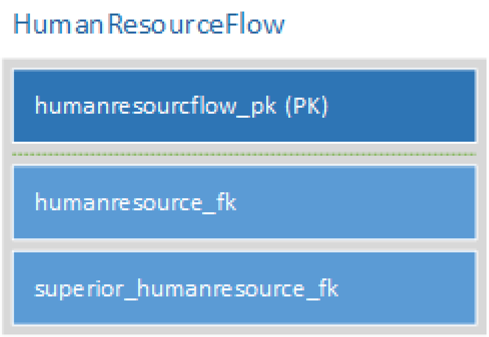


Table 16: HUMANRESOURCE layout

HUMANRESOURCE is the central table in the database and is linked to many other tables. The function of this table is to express the concept of an employee doing work for the company. If the software were ever to be expanded to accommodate not only engineering design offices, but construction sites as well, other types of resources could be introduced into the database. A human resource can be flagged as either standard, or non-standard. A standard employee follows the normal office hours, usually 9 AM to 5 PM. If this is however not the case, an entry in the NONSTANDARDHUMANRESOURCE table can be created. **efficiency_factor** and **idleness_factor** are in this version of the software constants that have to be set by the project manager. In future iterations however, more sophisticated methods of calculating real time data will be implemented. These values were discussed in a chapter 3, Quantifying Workload.

Column	Data type	Function
humanresource_description	String	Text stating human resource's description.
is_standard	Boolean	True or False value, stating whether the individual follows standard office hours.
efficiency_factor	Double	Constant defined by the project manager or a manager with intimate knowledge the individual. Describes individual's efficiency at work.
idleness_factor	Double	Constant defined by the project manager or a manager with intimate knowledge the individual. Describes the factor of a work-day individual is considered to be idle.

5.2.17 HUMANRESOURCEFLOW


HumanResourceFlow

humanresourcflow_pk (PK)

humanresource_fk

superior_humanresource_fk

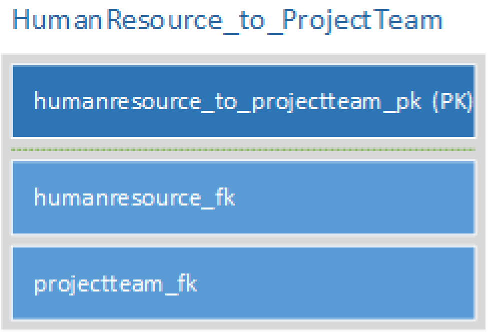
Table 17: HUMANRESOURCEFLOW layout

HUMANRESOURCEFLOW is a table designed to keep track of the command structure within a company. This is not used in the automatic task allocation procedure, but can be useful information for a project manager to have when deciding which task to give which employee. This table, like all linking tables, contain no non-key attributes and only stores the relations between two tables. In this case both relations refer to instances of HUMANRESOURCE.

Column	Data type	Function
humanresource_fk	Integer	Foreign key reference to an entry in the HUMANRESOURCE table. Referencing an individual.

superior_humanresource_fk	Integer	Foreign key reference to an entry in the HUMANRESOURCE table, referencing an individual who is the other column's superior.

5.2.18 HUMANRESOURCE_TO_PROJECTTEAM


HumanResource_to_ProjectTeam

humanresource_to_projectteam_pk (PK)

humanresource_fk

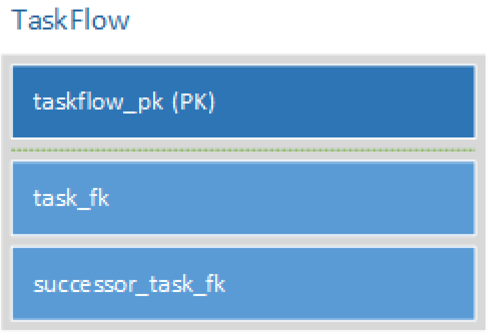
projectteam_fk

Table 18: HUMANRESOURCE_TO_PROJECTTEAM layout

HUMANRESOURCE_TO_PROJECTTEAM is a linking table used to link individual human resources to a project team. This is used to keep a record of people who worked on a project.

Column	Data type	Function
humanresource_fk	Integer	Foreign key reference to an entry in the HUMANRESOURCE table. Referencing an individual.
projectteam_fk	Integer	Foreign key reference to an entry in the PROJECTEAM table. Referencing a single team.

5.2.19 TASKFLOW


TaskFlow

taskflow_pk (PK)

task_fk

successor_task_fk

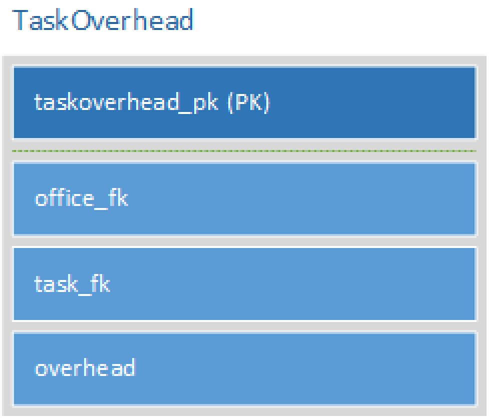
Table 19: TASKFLOW layout

TASKFLOW is a table designed to keep track of the scheduling of tasks. This table, like all linking tables, contain no non-key attributes and only stores the relations between two tables. In this case both tables are the same one, being two entries in TASK. TASKFLOW enables the storage and initiation of a chain of tasks. This helps project managers evaluate which tasks should be completed next.

Column	Data type	Function
task_fk	Integer	Foreign key reference to an entry in the TASK table. Referencing a single task.

successor_task_fk	Integer	Foreign key reference to an entry in the TASK table, referencing a task which is the other column's successor.

5.2.20 TASKOVERHEAD



TaskOverhead

taskoverhead_pk (PK)

office_fk

task_fk

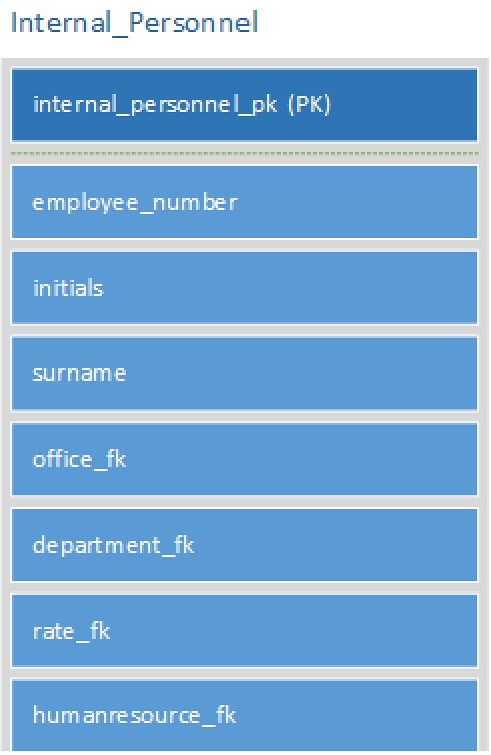
overhead

TASKOVERHEAD is a database table that links a specific combination of task -> office with an overhead value. Although the task -> office relation is a unique ordered pair, this relation is not enforced by the database structure. It is up to the implementation to enforce this rule. If this is not the case, there would be confusion as to what the overhead value is supposed to be.

Table 20: TASKOVERHEAD layout

Column	Data type	Function
office_fk	Integer	Foreign key reference to an entry in the OFFICE table. Referencing a specific office.
task_fk	Integer	Foreign key reference to an entry in the TASK table. Referencing a single task.
overhead	Double	Factor value stating task-office overhead. The value indicates the fraction of the task duration considered to be overhead.

5.2.21 INTERNAL_PERSONNEL



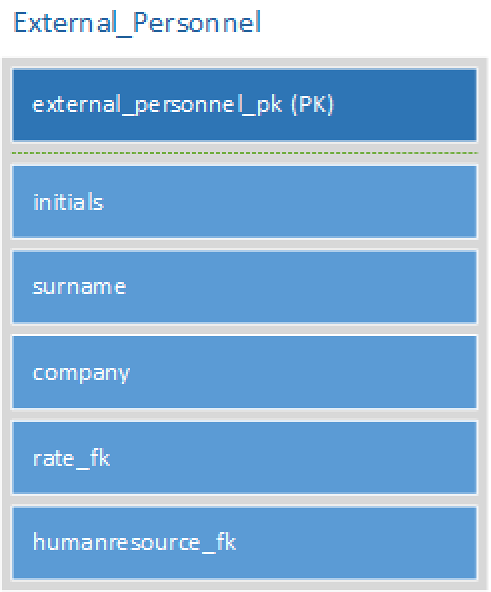
INTERNAL_PERSONNEL is responsible for conveying the concept of “employee” alongside HUMANRESOURCE. INTERNAL_PERSONNEL specifically models employees that have a permanent position in the organisation and are assigned a department and an office. As such these employees have a unique employee number to identify them. Each entry in this table is linked to a single unique entry in the HUMANRESOURCE table.

Table 21: INTERNAL_PERSONNEL layout

Column	Data type	Function
employee_number	Integer	Unique number assigned to all permanent personnel.
initials	String	Individual’s initials.
surname	String	Individual’s surname.
office_fk	Integer	Foreign key reference to an entry in the OFFICE table. Referencing a specific office.
department_fk	Integer	Foreign key reference to an entry in the DEPARTMENT table, indicating to which department the individual owes allegiance.
rate_fk	Integer	Foreign key reference to an entry in the RATE table. Referencing a specific salary rate.
humanresource_fk	Integer	Foreign key reference to an entry in the HUMANRESOURCE table, referencing a unique entry that table.

5.2.22 EXTERNAL_PERSONNEL

EXTERNAL_PERSONNEL is responsible for conveying the concept of “employee” alongside HUMANRESOURCE. EXTERNAL_PERSONNEL specifically models employees that do not have a permanent position in the organisation and is only temporarily part of a project as outsource personnel. Unlike a permanent member of staff, outsource personnel do not have employee numbers to identify themselves, nor are they connected to just one office or department. They can, but not necessarily, work for a single department. External personnel are presumed to do work for the whole company.



External_Personnel

external_personnel_pk (PK)

initials

surname

company

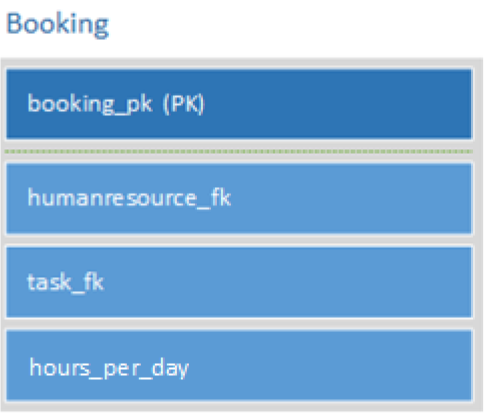
rate_fk

humanresource_fk

Table 22: EXTERNAL_PERSONNEL layout

Column	Data type	Function
initials	String	Individual’s initials.
surname	String	Individual’s surname.
company	String	The company which is the individual’s permanent employer.
rate_fk	Integer	Foreign key reference to an entry in the RATE table. Referencing a specific salary rate.
humanresource_fk	integer	Foreign key reference to an entry in the HUMANRESOURCE table, referencing a unique entry that table.

5.2.23 BOOKING



Booking

booking_pk (PK)

humanresource_fk

task_fk

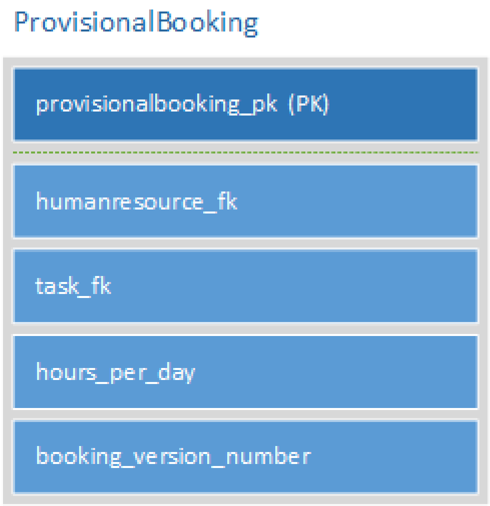
hours_per_day

Table 23: BOOKING layout

BOOKING is an important table that manages all task allocations. This table links an entry from HUMANRESOURCE, an individual, to an entry in TASK. The average hours per day the individual is booked to spend on the task is also stored as **hours_per_day**.

Column	Data type	Function
humanresource_fk	Integer	Foreign key reference to an entry in the HUMANRESOURCE table. Referencing an individual.
task_fk	Integer	Foreign key reference to an entry in the TASK table. Referencing a single task.
hours_per_day	double	The average hours per day the individual is booked to work on this specific task.

5.2.24 PROVISIONALBOOKING



ProvisionalBooking

provisionalbooking_pk (PK)

humanresource_fk

task_fk

hours_per_day

booking_version_number

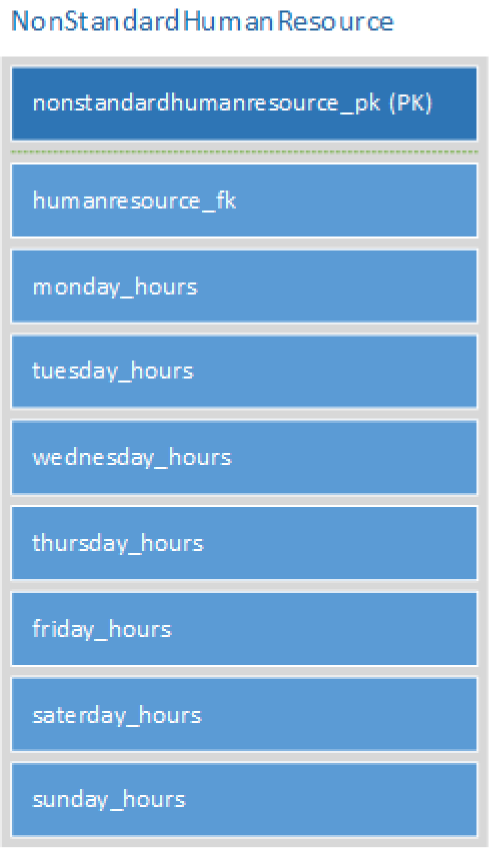
Table 24: PROVISIONALBOOKING layout

PROVISIONALBOOKING is a duplication of BOOKING, with the exception of having a **booking_version_number** field. During the task allocation procedure, the semi-automatic process first provisionally books an individual for tasks. These provisional bookings can then be reviewed by the project manager if he/she so desired. If slight alterations are made to the provisional bookings, in order to find the task allocation best suited to the needs of the organisation, the **booking_version_number** is incremented. PROVISIONALBOOKING can also be used to aid in setting up hypothetical task allocations for quotation purposes. When the project manager is satisfied with a project's task allocations, the software implementation can upgrade the provisional bookings to fixed bookings.

Column	Data type	Function
humanresource_fk	Integer	Foreign key reference to an entry in the HUMANRESOURCE table. Referencing an individual.
task_fk	Integer	Foreign key reference to an entry in the TASK table. Referencing a single task.
hours_per_day	Double	The average hours per day the individual is booked to work on this specific task.
booking_version_number	Double	Versioning system in place to remember what possibilities was already explored in the allocation process.

5.2.25 NONSTANDARDHUMANRESOURCE

NONSTANDARDHUMANRESOURCE is a table that describes an individual's working hours if he/she does not conform to the company standard. When an individual stored in HUMANRESOURCE is flagged as being non-standard, an entry in this table is expected and required. This table is necessary for people like off-peak repair technicians, who only work on weekends for example. This data can then be used to indicate work availability and billing.



NonStandardHumanResource

nonstandardhumanresource_pk (PK)

humanresource_fk

monday_hours

tuesday_hours

wednesday_hours

thursday_hours

friday_hours

saterday_hours

sunday_hours

Table 25: NONSTANDARDHUMANRESOURCE layout

Column	Data type	Function
humanresource_fk	Integer	Foreign key reference to an entry in the OFFICE table. Referencing a specific office.
monday_hours	Double	Number stating the hours the individual is expected to work on Monday.
tuesday_hours	Double	Number stating the hours the individual is expected to work on Tuesday.
wednesday_hours	Double	Number stating the hours the individual is expected to work on Wednesday.
thursday_hours	Double	Number stating the hours the individual is expected to work on Thursday.
friday_hours	Double	Number stating the hours the individual is expected to work on Friday.
saturday_hours	Double	Number stating the hours the individual is expected to work on Saturday.
sunday_hours	Double	Number stating the hours the individual is expected to work on Sunday.

5.2.26 VACATION

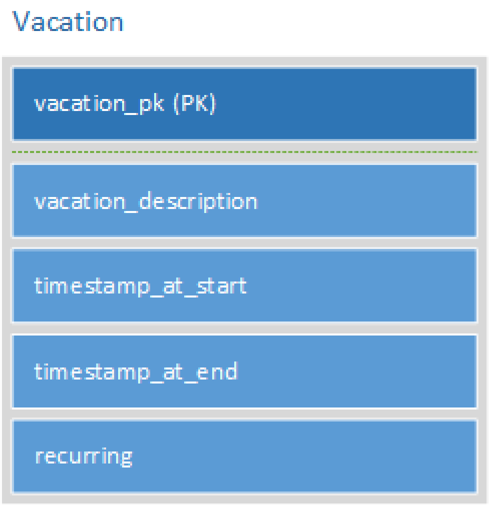
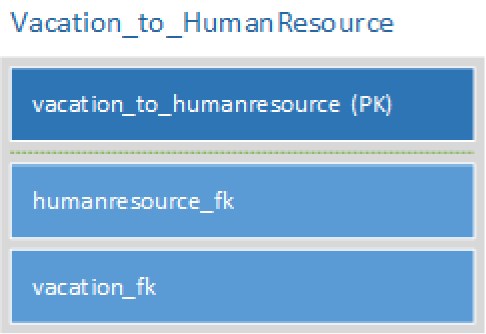


Table 26: VACATION layout

VACATION is a table defining the concept of a vacation, or an off day. A vacation can be seen as similar to a task, and as such can be assigned to individuals. A vacation can also signify sick days or public holidays. A vacation has a start date, stored as **timestamp_at_start**, and an end date, stored as **timestamp_at_end**. If a vacation is in fact a public holiday, the administrator may want to flag the holiday as recurring. To incorporate different intervals of recurrence, the data type of **recurring** is a string describing the recurrence. The software solution will interpret these strings and incorporate the data into a calendar.

Column	Data type	Function
vacation_description	String	Text declaring vacation description.
timestamp_at_start	Timestamp	Date of vacation start.
timestamp_at_end	Timestamp	Date of vacation end.
recurring	String	Empty if not a recurring vacation. Text describing the recurrence if it is, “weekly” or “yearly” are some valid descriptions.

5.2.27 VACATION_TO_HUMANRESOURCE



Vacation_to_HumanResource

vacation_to_humanresource (PK)

humanresource_fk

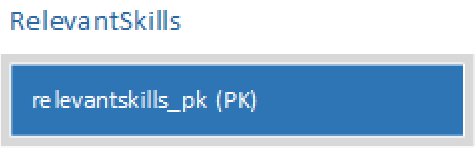
vacation_fk

VACATION_TO_HUMANRESOURCE links entries from the VACATION table to entries in the HUMANRESOURCE table. In essence, tagging an individual as taking a specific vacation.

Table 27: VACATION_TO_HUMANRESOURCE layout

Column	Data type	Function
humanresource_fk	Integer	Foreign key reference to an entry in the HUMANRESOURCE table. Referencing an individual.
vacation_fk	integer	Foreign key reference to an entry in the VACATION table. Referencing a specific vacation.

5.2.28 RELEVANTSKILLS



RelevantSkills

relevantskills_pk (PK)

Table 28: RELEVANTSKILLS layout

RELEVANTSKILLS is a table that is used as an assembly point to link skills which are deemed similar to one another in some way. This table is essential in increasing the accuracy of the task allocation procedure by enabling the possibility of adding a search filter which scores for a similar skill set than what is required.

5.2.29 SKILL_TO_RELEVANTSKILLS

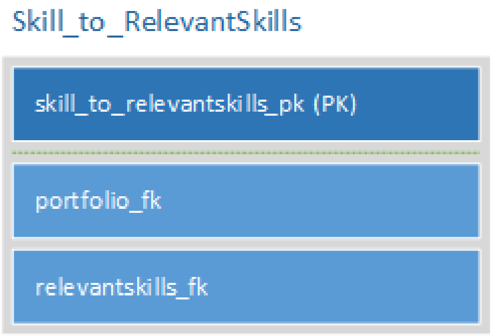


Table 29: SKILL_TO_RELEVANTSKILLS layout

SKILL_TO_RELEVANTSKILLS is a table that links entries in SKILL to instances of RELEVANTSKILLS. In principle this table stores combinations of skills which are similar to one another. For example the skill "Draftsman" could be seen as similar to "Structural analyser" as both may involve drawing structures with software. It could be reasoned that both skills share common ground. This pool of skills that are comparable is used to receive partial scorings for task allocations. If there is absolutely no individual trained to complete a task, rather the task be assigned to an individual who is skilled in something similar, than to someone who is not.

Column	Data type	Function
skill_fk	Integer	Foreign key reference to an entry in the SKILL table. Referencing a specific skill.
relevantskill_fk	Integer	Foreign key reference to an entry in the RELEVANTSKILLS table.

5.2.30 QUOTE

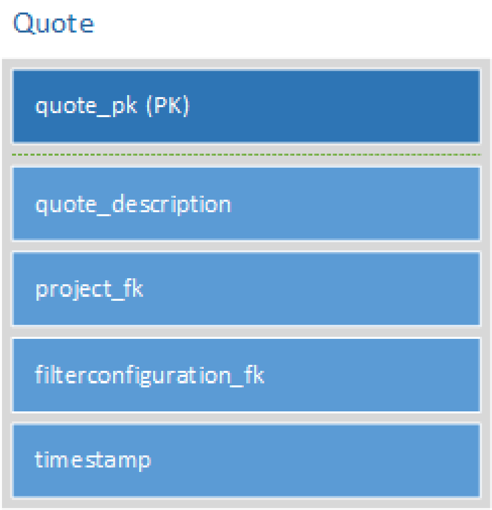
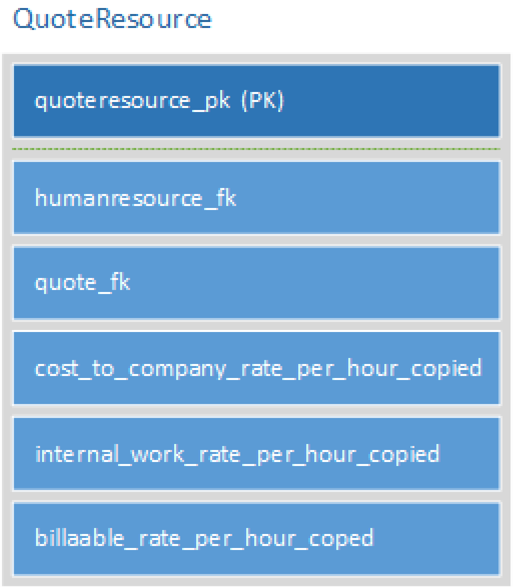


Table 30: QUOTE layout

QUOTE purveys the concept of quotations. It would sometimes be prudent for an organisation to estimate the cost of a project, either to the client, or to the organisation, to decide whether to take a project on. At the time a quotation is desired, an entry in this table will be made, linking to an entry in PROJECT and giving a current date. QUOTE also links to an entry in FILTERCONFIGURATION, to store which filters were on at the time the quote was created. This can aid in the duplication of an allocation if so desired. A quote is assembled with aid from the QUOTERESOURCE table.

Column	Data type	Function
quote_description	String	Text giving summary of quote.
project_fk	Integer	Foreign key reference to an entry in the PROJECT table. Referencing a specific project.
filterconfiguration_fk	Integer	Foreign key reference to an entry in the FILTERCONFIGURATION table. Referencing a specific configuration of filters.
timestamp	Timestamp	Date when quote was created.

5.2.31 QUOTERESOURCE



QuoteResource

quoteresource_pk (PK)

humanresource_fk

quote_fk

cost_to_company_rate_per_hour_copied

internal_work_rate_per_hour_copied

billaable_rate_per_hour_coped

Table 31: QUOTERESOURCE layout

QUOTERESOURCE works in combination with QUOTE to create a useable quotation. As the only type of resource currently supported by the model is human resources, this table basically links a human resource to a quote in a one-to-many relationship. This implies that many human resources can be mapped to the same quote. As it is possible for individuals' asking rate to change with time, an individual's CTC – Cost to Company, internal work rate and billable rate are copied into this table to enable recreating quotation history.

Column	Data type	Function
humanresource_fk	String	Foreign key reference to an entry in the HUMANRESOURCE table. Referencing an individual as a resource.
quote_fk	Integer	Foreign key reference to an entry in the QUOTE table. Referencing a specific quote. Linking this QUOTERESOURCE to a QUOTE.
cost_to_company_rate_per_hour_copied	Double	Value detailing the resource's hourly Cost to Company, duplicated at the time of quotation.
internal_work_rate_per_hour_copied	Double	Value detailing the resource's hourly internal work rate, duplicated at the time of quotation.
billable_rate_per_hour_copied	Double	Value detailing the resource's hourly billable work rate, duplicated at the time of quotation.

5.3 ENTITY-RELATIONSHIP DIAGRAM

This diagram provides an overview of the tables and the relationships between them.

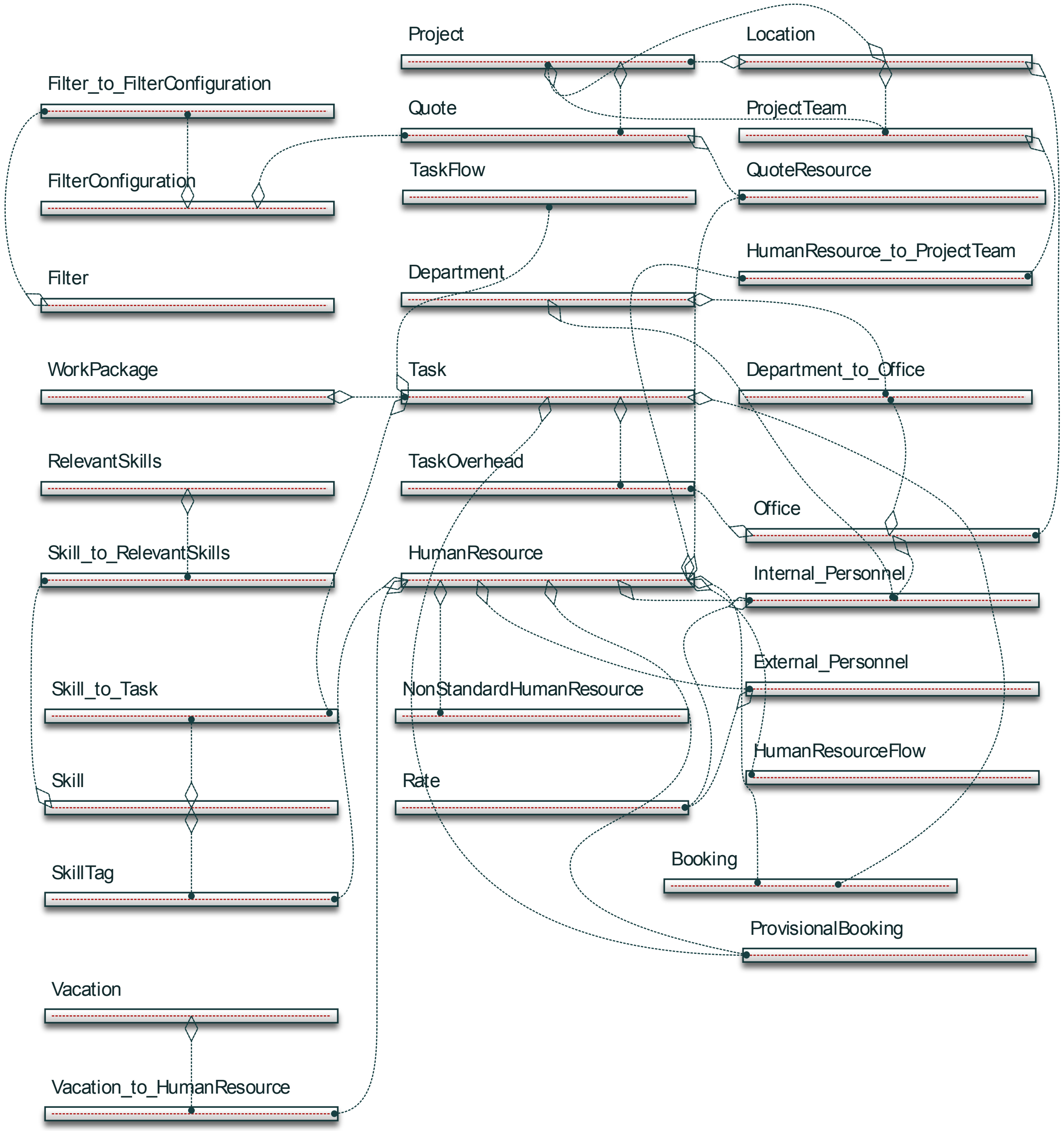


Figure 5: Database ERD

6. ENHANCING THE MODEL

Following the aims of the research project described here, the core components of an object model aimed at task allocation was developed in chapter 4, and its supporting database model in chapter 5. The functionalities of the models were hinted at, but not described in detail. In this chapter the key functionalities and how the model was enhanced to incorporate them, are described.

6.1 HIERARCHY

Relationships between database records are formed using primary and foreign keys. Some database tables exist purely to link records in many-to-many relationships. In Java, relationships are maintained using references.

A need was identified for specific concepts to be able to able to link to separate instances of that same concept. This is needed for two specific purposes, namely to create a task hierarchy and a human resource hierarchy. In the case of the task hierarchy the relations describe the order in which the tasks should be completed by linking a task to its successor. In the case of the human resource hierarchy, the command structure of the organisation is described, where every human resource is linked to his/her immediate superior.

6.2 OPTIMISATION AND FINDING PEOPLE

There are many types of technical optimisation available, from conjugate gradient methods and Quasi-Newton methods to evolutionary optimisation. It is not practical to run an optimisation procedure to select the most suitable human resource for a given task. Instead, a powerful and efficient filtration procedure was implemented.

Three layers of filters were designed to aid in finding the best choice for task allocations. These layers are Restrictive Filters, Exclusion Filters, and Refinement Filters.

6.2.1 SCORE

Score is the concept which the search filters use to determine the most appropriate candidate for a task. A score maps a single human resource to single task. A new `Score` is created for each person-task combination where said person is eligible to execute the task. Each `Score` has a numerical value, called the score, to numerically track how appropriate the person is to do the specific task. `Scores` are stored in a sortable set, or list, called `ScoreList`. Each refinement filter sorts the `ScoreList` is its own unique way and increments the score value of each `Score` appropriately. All sorting is done using Java's comparable interface. Once all the refinement filters enabled by the project manager have run their course, the `ScoreList` is analysed to find the highest scoring individuals for each task.

The highest scoring individual, along with a percentage of the runner-ups, are displayed on the Graphical User Interface[p] for the project manager to review.

6.2.2 RESTRICTIVE FILTERS

The concept behind restrictive filters is to limit the amount of data queried from the database. If an organisation is large and the initial query for data returns too much information, the local computer would struggle to store the entire result-set inside memory and deconstruct the information. This problem can be averted if the initial search is given some directional focus. If the project manager only wants to recruit individuals from a specific office, region, or even country, a restrictive filter is the way to accomplish this.

Restrictive filters are the only filters that explicitly alter the SQL String sent to the database.

6.2.3 EXCLUSION FILTERS

Exclusion filters serve to exclude invalid individuals from the search, further narrowing the search and potentially increasing the speed at which further filter actions are taken. The only exclusion currently implemented is a filter checking for individuals who are currently working on a flat-

[p] GUI

lined task, called Exclude Task Priority. Per definition, an individual who is assigned to a flat-lined task must spend 100% of his time on finishing it, and as such is not available to work on other tasks.

6.2.4 REFINEMENT FILTERS

Refinement filters are the third layer of filters and perform the brunt of the optimisation. Each filter can be turned on or off, depending on the needs and wants of the project manager. Sliders can be used to determine an individual filter's relative importance to the current search. This functionality means a project manager can truly customise his/her search to suit the need of the organisation.

The refinement filters that are currently developed will now be discussed. It should be noted that all score alterations are affected by the user's choice of slider value.

6.2.4.1 EXACT SKILL MATCH

The Exact Skill Match filter is a search that sorts through the `ScoreList` and assigns a fixed value to the score of each individual who possess a skill required by the task. If a task has multiple skill requirements, a single individual can get multiple score alterations if he/she possess the correct skills.

6.2.4.2 RELEVANT SKILLS

The Relevant Skills filter works similarly to the Exact Skill Match filter. The filter searches through the skills of all the applicable `Scores` and assesses whether one of an individual's skills are in the same `SkillGrouping` as a skill required by a task. `SkillGrouping` is a Java class that extends `Set<Skill>`, and as such is a set of skills which are all considered relevant to each other. Skill relevance has to be determined manually by a user before allocation can occur, and is persistently stored in the database in the RELEVANTSKILLS table. Relevant skill matches are treated the same way as exact skill matches, but carries less weight in the filter procedure.

6.2.4.3 MAXIMISE PROFITS

The Maximise Profits filter assigns scores according to individuals’ billable rate per hour, retrieved from the **billable_rate_per_hour** column in the RATE table. The scores are assigned with an increasing order of rate values. A linear counter, starting with a value of one goes down the list, adding the counter’s value to the individual’s score and incrementing by one when the human resource changes. Consequently the same human resource always receives the same amount of points for all of his possible tasks. Individuals that generate the lowest income for the organisation will receive the least amount of points, while individuals with the highest billable rate receive the most.

6.2.4.4 MINIMISE COST TO COMPANY

The Minimise Cost to Company filter sorts individuals according to decreasing cost to company values, retrieved from the **cost_to_company_per_hour** column in the RATE table. Again a linear counter with a starting value of one goes down the list of scores, increasing the value of the counter every time the human resource changes. This results in individuals with a higher cost to company receiving less points than individuals with a low cost to company.

6.2.4.5 CALCULATE WORKLOAD

Calculate Workload is arguably the most important and innovative refinement filter in the filter system. This filter uses the individual workload calculation defined earlier in this document.

$$\begin{aligned} &\text{perceived workload} \\ &\quad = t_{\text{ineff}} \\ &\quad * \Bigg(\Bigg(\left(\begin{cases} 0, & T = 0 \\ 0, & \sum_{i=1}^{n}(t_i(1+o_i) = 0 \\ 0.841^{(T-\sum_{i=1}^{n}(t_i(1+o_i)}, & \sum_{i=1}^{n}(t_i(1+o_i) \le T \\ 0.959^{(T-\sum_{i=1}^{n}(t_i(1+o_i)}, & \sum_{i=1}^{n}(t_i(1+o_i) > T \end{cases}\right)(0.5) + (\begin{cases} 0, & L = 0 \\ 0, & n = 0 \\ \log_9(L+1), & n \neq 0 \end{cases})(0.26) \\ &\quad + \left(\begin{cases} 0, & n = 0 \\ {n^n}/{33.96}, & n \neq 0 \end{cases}\right)(0.24) + \left(\begin{cases} 0, & n \le 1 \\ 0.000472 * s_{max} * n^{3.405}, & n > 1 \end{cases}\right)\Bigg) + \begin{cases} 0, & L = 0 \\ 0, & n = 0 \\ IF, & n \neq 0 \end{cases} \end{aligned}$$

Equation 22: Equation for quantifying perceived workload

The problem with using this equation is that it computes PWL for a single day, which is not very useful as there is not a specific day that needs analysing. The method that can, however, be used to retrieve useful information is to take the average PWL over a period of days.

The time from the earliest start date among all the tasks that are part of the search, to the latest end date of these tasks was taken as the period. Each day's PWL is calculated, added together, and then divided by the total number of days within the period. This provides a reasonable estimation of an individual's workload.

Another obstacle to overcome when trying to use the workload equation, is to find the amount of hours spent on each task every day. This is difficult because of the way tasks are assigned. An individual is booked for a task, from the earliest start date, to the latest end date; and the individual has to find time within that timeframe to complete the task. This makes it impossible to pinpoint exactly when, and how much, the individual will work on a specific task.

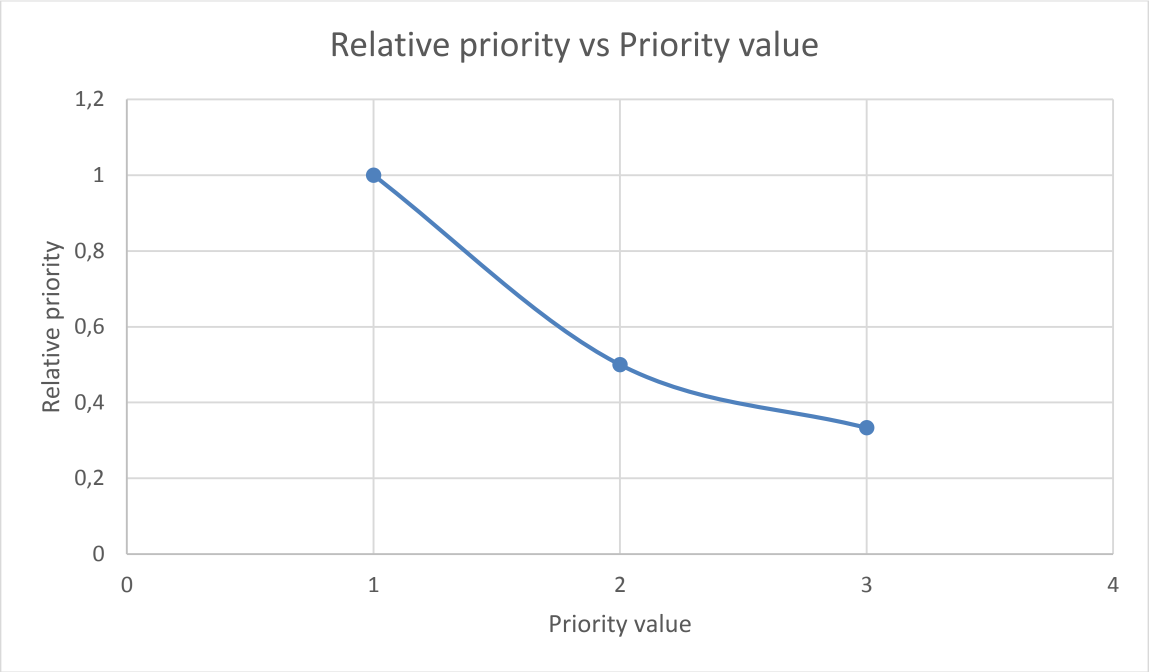
The importance, or priority, of a task can however be used effectively to ascertain how much of an individual's day he/she should spend working on that specific task. The individual's workday duration is split up into as many parts as there are tasks assigned to him on that day. Tasks with a higher importance ranking receive more time, while less important tasks receive less time. This method is called time distribution. If one of the tasks is flat-lined, the individual's workload is automatically set to 100% and he/she becomes unsuitable for further loading.

Assume that the summation of all the time allocated to tasks within a day, always equals the workday duration. The question is, how long should an individual spend on each task?

The fraction, or weight, of an individual's workday spent on a single task can be written as $(workday\ duration) * weight_i = taskTime_i$, with *i* referencing a specific task. A task's weight is a function of that specific task's priority.

Task priority is defined as inversely proportional to numerical value, meaning that low values of priority actually denote more importance than high values of priority. Therefore, a task's *relative priority* is taken as $^1/_{priority}$.

The *relative priority* version of the three available priority settings is shown in the graph below.



Graph 20: Task relative priority vs priority value

It should be noted that the final priority, the fast-track priority, when the priority equals zero, yields a relative priority of infinity. This does not matter overly much as we know that when an individual is assigned a task with a zero priority value, that person may not be assigned any other tasks and should spend the full workday on that task.

As can be seen from graph 20, the relative priority forms a hyperbola as priority increases. This is ideal as it states that a task with a priority of 1, is exponentially more important than a task with a priority of 2, and so on.

Per definition, task weights should also always sum up to 1. To enable this the following equation for task weight is proposed:

$$weight = \frac{relative\ priority}{\sum relative\ priority},$$

Equation 23: Weight of task duration out of full day

Where $\sum relative\ priority$, is the summation of all the day's *relative priorities*.

This would mean, written mathematically correctly, that:

$$taskTime_i = (workday\ duration) * \frac{relative\ priority_i}{\sum_{k=1}^{n} relative\ priority_k},$$

Equation 24: Equation for time spent on specific task

With *n* denoting the number of tasks for the day, and *i* denoting the current task.

A very descriptively named method, `ifEndDateisTodayAndTaskDurationIsLessThanDay()`, checks for the special cases where a specific day coincides with a task's end date, or when the duration of a task is less than a day. In these cases the full task duration is fit into the one day, in preparation for focusing on that one task.

6.2.4.6 DISTANCE TO PROJECT

Distance to Project is a filter that sorts individuals according to the straight-line distance between their offices and the project location. Distances are sorted in descending order, meaning the distance gets smaller as the list goes down. A linear counter with a starting value of one crawls down the list, increasing the value of the counter with every line. This counter is added to the score of the individual. This results in individuals who work farther away from the project destination receiving less points than those who work closer.

It was deemed acceptable to assume the earth is a sphere for calculation purposes. The shortest distance along the surface of a sphere between two points is along a great-circle which contains the two points. This method of calculating distance between coordinates results in a possible error of 0.5%. [27]

The calculation method will now be discussed.

Let $\Phi_1, \lambda_1$and Φ_2, λ_2 be the geographical latitude and longitude of two points 1 and 2. If $\Delta\Phi, \Delta\lambda$ was the absolute differences, then $\Delta\sigma$, the dominant angle between the two points would be given by the spherical law of cosines as [28]:

$$\Delta\sigma = \arccos(\sin\Phi_1 \sin\Phi_2\ + \cos\Phi_1 \cos\Phi_2 \cos\Delta\lambda)$$

Equation 25: Spherical law of cosines

The distance *d*, for a sphere of radius *r* and angle $\Delta\sigma$ would then be calculated as $bd = r * \Delta\sigma$

The shape of the Earth closely resembles a flattened sphere with an equatorial radius of 6478 km. If a completely spherical Earth is assumed, accepting the error of 0.5%, a good estimation for the radius of the sphere would be the mean earth radius: 6371 km.

6.3 TASKS AND FLAT-LINING

Tasks have a duration, start, and end dates that can be used to calculate the latest date a task has to be started before it cannot statistically be completed in time. If a task is not yet started and this latest start date nears, the task can be flagged as critical and that it should be flat-lined. The latest start date can be calculated by subtracting the task duration from the task's latest end date.

A rudimentary system was developed to aid in notifying project managers of important tasks. The software scans for any currently assigned tasks and their individual priority; then assigns a flag to be displayed on screen. If a task has a priority of zero, it indicates the task is to be fast-tracked, or flat-lined. The software also compares the current local date to that of the latest start date of the task. The software currently alerts the project manager if the current day is within one week of the latest day a task must be started by displaying the following message on-screen: "`(GET READY TO FLATLINE SOON)`". A similar message is displayed for when the day has arrived, or has passed.

}

7. TEST PROJECT: SOFTWARE IN ACTION

An imaginary company was simulated to test the software implementation. The company has the indicated command structure shown in figure 2 and certain characteristics of the personnel are indicated.

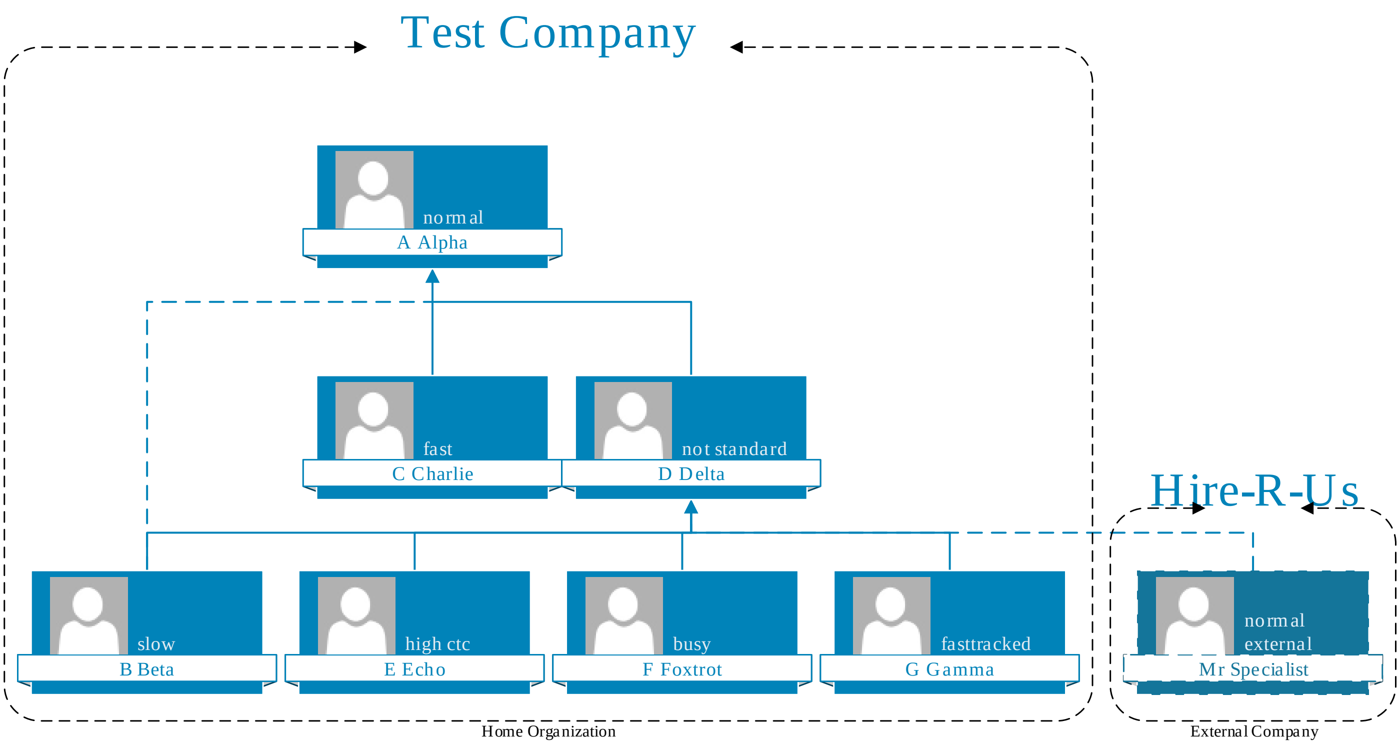


Figure 6: Test company command structure

The company has seven employees, and one external personnel member currently on payroll. Each of the employees has a unique characteristic: one is fast in his work, another is slow, another is busy, etc.

One of the projects the company has on its to-do list is a project called, “Somerset Will Thrive”, which is a small project on a building in Somerset-West. The project is comprised of only four tasks that have to be completed, with each task requiring one or more skills to complete.

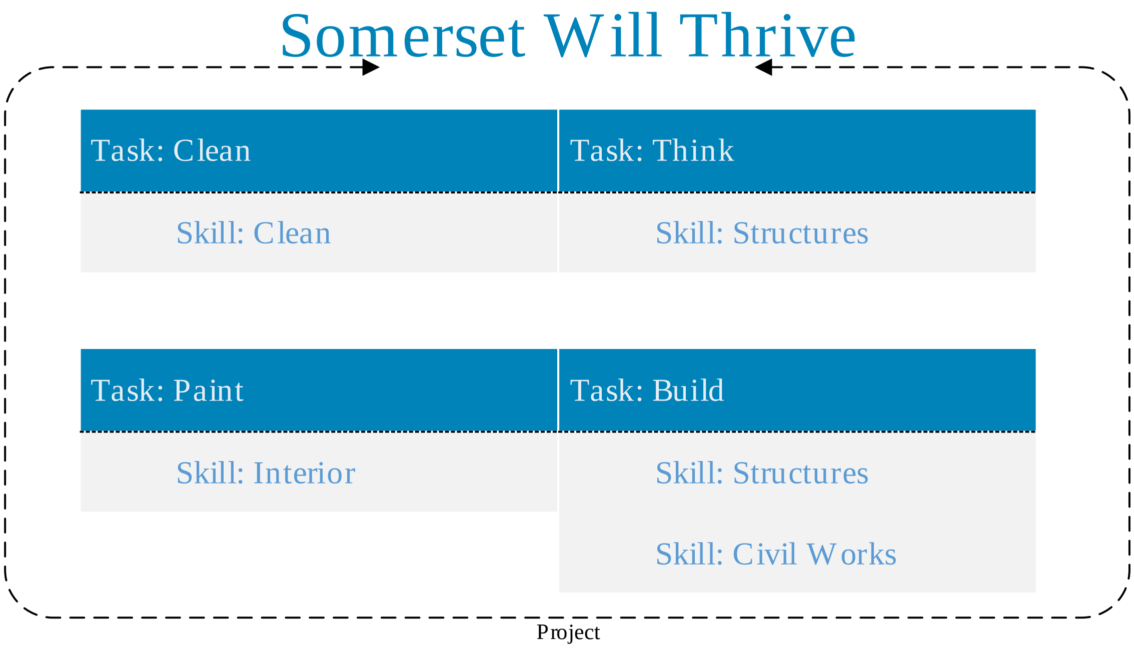


Figure 7: Example project task breakdown

The company has two offices, one in Cape Town, and one in Stellenbosch. The personnel's skills, along with which employees are native to which offices are shown in the following diagram.

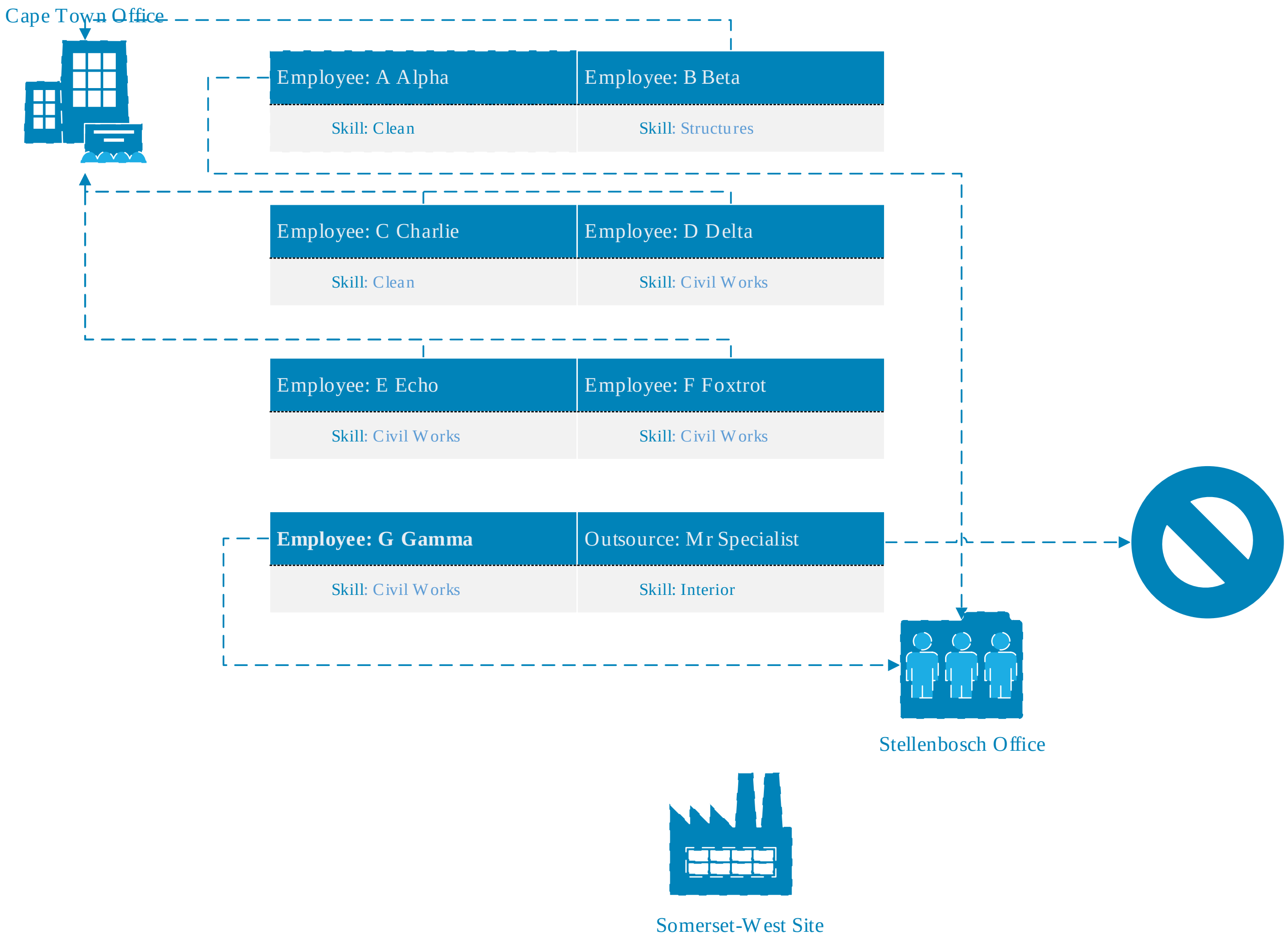


Figure 8: Test company employee skills and locations

The locations have the following details, in the format that the information is stored in the database:

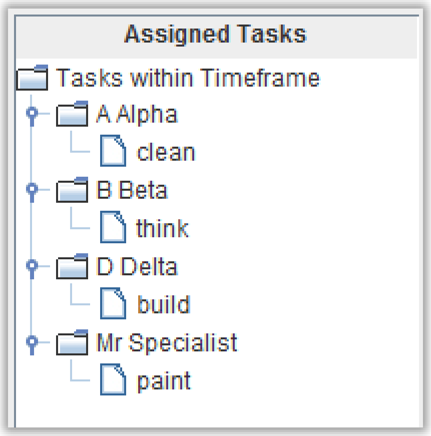
Location

location_description	Stellenbosch Office	Cape Town Office	Somerset-West Site
country	South Africa	South Africa	South Africa
region	Western Cape	Western Cape	Western Cape
city	Stellenbosch	Cape Town	Somerset West
longitude	33.92	33.9707	34.0833
latitude	18.8	18.4244	18.85

Figure 9: Location details

The project, meaning all of the tasks, now have to be assigned to individuals within the organisation. Some of the important aspects the task allocation procedure look at is: Skill Matching, availability and relative risk, and travel distance.

After the semi-automatic procedure was left to run its course, with the top choice being chosen, this is the resulting task allocations.



Assigned Tasks

- Tasks within Timeframe
  - A Alpha
    - clean
  - B Beta
    - think
  - D Delta
    - build
  - Mr Specialist
    - paint

Figure 10: Allocations as made by software

In this organisation, *B Beta* is the only individual with a *Structures* skill, and is thus the only logical choice for *Task: Think*, as it requires that exact skill. There are two possible choices for *Task: Clean*, as there are two individuals with the *Clean* skill, namely *A Alpha* and *C Charlie*. The only difference between the two individuals is that *A Alpha* works in the Stellenbosch Office, while *C Charlie* works in the Cape Town Office. As Stellenbosch is closer to Somerset West than Cape Town is, *A Alpha* is the better choice for the task. Stellenbosch is about 20 kilometres from Somerset West, while Cape Town is about 45 kilometres [29].

There are five possible choices for *Task: Build*; namely *Beta*, *Delta*, *Echo*, *Foxtrot*, and *Gamma*. All these individuals only have one of the skills required by the task. *Foxtrot* already as a task assigned to him, as he is trademarked by being busy. *Gamma* is also busy with a task, but it is flat-lined, or fast-tracked, and as such cannot be loaded with another task. *Beta* is slow and already has a task assigned to him this project. *Echo* has a high Cost to Company. This only leaves

D Delta, which although he does not keep regular office hours, is mathematically the best choice for the task.

The organisation has no permanent employees that possess the Interior skill that is required by *Task: Paint*. This would have been a sign for the project manager that an external resource would have to be approached for necessary expertise. As such, *Mr Specialist* was already on the payroll of the organisation, and automatically received the task as he was the perfect fit.

This test project is a relatively small and simple example, and an individual could easily have estimated the best task allocation his/herself. The real power of software comes into play when the amount of tasks and possible personnel grows too large for a single person to construct all the possible associations.

8. PROJECT RESOURCE LOCATOR - KEY POINTS OF THE CODE

Project Resource Locator is the working title of the software implementation. This chapter will describe key aspects of this software and discuss how it works and also look at the Java programming language implementation.

For task allocation, relevant information was retrieved from the database and stored in Java classes and objects. This happens before any search is started. If a search was done directly on the database, the connection to the database would have to remain open for an extended time, during which concurrent users could skew results by altering data. In order to populate the different tables and dropdown lists seen by the user on the Graphical User Interface (GUI), the database has to be queried directly.

The way in which several important concepts were implemented in the software is described below.

8.1 FILTER

Three layers of filters were designed, Restrictive Filters, Exclusion Filters, and Refinement Filters. These three filters were all defined as abstract super-classes `RestrictiveFilter`, `EliminationFilter` and `RefinementFilter`, with individual types of filters simply extending these super-classes.

```java
public abstract class RefinementFilter {
```

Source Code Insert 1: Typical Filter class declaration

```java
public class NoRestrictionRetrictive extends RestrictiveFilter{

	public NoRestrictionRetrictive() {
		super("No Restriction");

	}
```

Source Code Insert 2: Example RestrictiveFilter constructor

RestrictiveFilter contains two mandatory methods, influenceQueryI() and influenceQueryE(), which respectively alter the SQL query sent to the database for obtaining internal personnel and external personnel.

Both EliminationFilter and RefinementFilter have only one mandatory method, affectScores(). This method retrieves the current ScoreList, as defined in a previous chapter, and affects the results accordingly.

Instances of EliminationFilter works by removing invalid Scores from the list.

```
else
       {
              //remove from list.
              scores.remove(scores.getScoreByKey(hr.getPk(), newtask));
              System.out.println("Task removed from list: "+newtask.getPk()+" from
"+hr.getSimpleName());
```

Source Code Insert 3: Removal of Scores using instance of EliminationFilter

Instances of RefinementFilter numerically add value to specific Scores.

```
public void affectScores() throws SQLException
{
        double constant = 12.0;
        for (Score sc : scores.getScores())
        {
                HR hr = sc.getHR();
                SkillList hrSkills = hr.getSkills();
                for (Skill hrP : hrSkills)
                {
                        System.out.println(hr.getSimpleName()+" has skill: 
"+hrP.getDescription());
                        System.out.println(hrP.getDescription());
                        for (Task taskp : scores.getTasks())
                        {
                                SkillList taskSkills = taskp.getSkills();
                                for (Skill tP : taskSkills)
                                {
                                if( tP.getDescription().equals(hrP.getDescription()))
                                        {
                                        int key = hr.getPk();
                                        Score s = scores.getScoreByKey(key, taskp);

                                        s.addPoints(constant * slidervalue);
                                        }
                                }
                        }
                }
        }
}
```

Source Code Insert 4: Typical RefinementFilter affectScores() method

8.2 DISTANCE BETWEEN OFFICE AND PROJECT

The calculation used within the Refinement filter, Distance to Project, was implemented as follows:

```
private void calc()
{
        double d2r = (Math.PI /180);
        try {
                double dlong = (endpoint.getLon() - startpoint.getLon()) * d2r;
                double dlat = (endpoint.getLat() - startpoint.getLat()) * d2r;
                double a =
                                Math.pow(Math.sin(dlat / 2.0), 2)
                                + Math.cos(startpoint.getLat() * d2r)
                                * Math.cos(endpoint.getLat() * d2r)
                                *Math.pow(Math.sin(dlong / 2.0), 2);
                double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

                setDistanceKilometer(6371 * c);

        }       catch(Exception e){
                e.printStackTrace();
        }

}
```

Source Code Insert 5: Distance calculation in code

8.3 IMPLEMENTING WORKLOAD CALCULATION

Two separate classes implement the workload calculation namely, `Workload` and `WorkloadCalculations`. `WorkloadCalculations` queries the database for all relevant information. The applicable Refinement Filter receives all the necessary information from `ScoreList`. The full `WorkloadCalculations` class is listed in Appendix B.

The method used to retrieve an individual’s average workload over a period of days is shown here:

```
public double getAverageWorkload(HR hr, ScoreList s)
        {

                Calendar earliest = Calendar.getInstance();
                Calendar latest = Calendar.getInstance();
                earliest.setTime(new Date(0));
                latest.setTime(new Date(999999999));

                for (Score score : s)
                {

                        if (earliest.after(score.getTask().getEarlyStartDate()))
                                earliest.setTime(score.getTask().getEarlyStartDate());
                        if (latest.before(score.getTask().getEndDate()))
                                latest.setTime(score.getTask().getEndDate());
                }
                return getAverageWorkload(hr, earliest.getTime(), latest.getTime());

        }
```

Source Code Insert 6: Average workload over task period - scores

```
public double getAverageWorkload(HR hr, Date start, Date end)
{
        Calendar c = Calendar.getInstance();
        c.setTime(start);
        double work = 0;
        double count = 0;

        while(c.getTime().before(end))
        {
                Date today = c.getTime();
                work += compute(hr, today);
                count++;

                if (c.DAY_OF_MONTH < c.getActualMaximum(c.DAY_OF_MONTH))before(end))
                {
                        c.add(Calendar.DAY_OF_MONTH, 1);
                }
        }
        return work/count;
}
```

Source Code Insert 7: Average workload over task period - dates

The method used to distribute the available hours in a day to all the assigned tasks is described here. Note that the `booking_distribution` object is a map linking a task with its allocated time:

```
public void distributeTime(HR hr)
       {
              double total = 0;
              double base;
              double T = hr.getT();
              //remove tasks that have already been allocated time from the method
              for (Task t : booking_distribution.keySet())
              {
                     if (booking_distribution.get(t) != null)
                            T = T-booking_distribution.get(t);
              }
              for (Task t : booking_distribution.keySet())
              {
                     total += 1.0/t.getPriority();
              }
              base = T/total;
              for (Task t : booking_distribution.keySet())
              {
                     if (booking_distribution.get(t) == null)
                            booking_distribution.put(t, base/t.getPriority());
              }
       }
```

Source Code Insert 8: Time allocation among tasks

The equation for task time distribution,

$$taskTime_i = (workday\ duration) * \frac{relative\ priority_i}{\sum_{k=1}^{n} relative\ priority_k},$$

Equation 26: Task time distribution

is represented in the source code insert with the double `base` equal to $\frac{(workday\ duration)}{\sum_{k=1}^{n} relative\ priority_k}$, and `t.getPriority()` denoting a task's priority, meaning $\frac{1}{t.getPriority()}$ = *relative priority*.

`total` is equal to $\frac{1}{\sum_{k=1}^{n} relative\ priority_k}$.

8.4 RESOURCELOCATOR

`ResourceLocator` is a Java class that acts as a bridge between user input on the GUI and the search algorithms. This offers possibilities for external software wanting to use the search without going through the native GUI.

```
public ResourceLocator (DataBaseInterface dbi, List<?> tasks, FilterSettings
settings) throws SQLException
```

Source Code Insert 9: ResourceLocator class declaration

In the code insert above:

- `DataBaseInterface` is a Java class that creates a connection to the database and has the ability to send it SQL updates and queries. It is used throughout this software to communicate with the database
- Tasks is a list of the tasks that is part of the search procedure.
- `FilterSettings` is a summary of the user defined search parameters and weights.

8.5 CONNECTING TO THE DATABASE

8.5.1 JAVA REPRESENTATION OF DATABASE TABLES

Thirty-one classes were developed to create the database tables and transfer information between the Java software and the database. The classes generate the Structured Query Language (SQL) required by the database management system for the task at hand.

Each class extends superclass `DBObject` and deals with a specific table in the database. In each class the table column heads are defined with getter and setter methods and provision is made for `createTable()`, `addSql()` and `updateSql()` methods. Typical examples of these methods from the TASK table will now be given.

`createTable()` provides the SQL to create the table, complete with names and constraints.

```
public static String createTable(){
      return "CREATE TABLE "+tableName()+"(" +
             P_KEY +" SERIAL PRIMARY KEY,"+
             PROJECT +" INTEGER NOT NULL, "+
             DESCRIPTION +" VARCHAR("+length(DESCRIPTION) +") UNIQUE NOT NULL, "+
             DURATION +" FLOAT4 NOT NULL, "+
             WORKPACKAGE +" INTEGER NOT NULL, "+
             EARLY_START +" TIMESTAMP NOT NULL, "+
             LATEST_END +" TIMESTAMP NOT NULL, "+
             ACTUAL +" TIMESTAMP NOT NULL, "+
             STRETCHABLE+" BOOLEAN NOT NULL, "+
             STIFFNESS+" FLOAT4 NOT NULL, "+
             PRIORITY+" INTEGER NOT NULL, "+
             "CONSTRAINT project_fk FOREIGN KEY("+PROJECT+") REFERENCES "+
             Project.tableName()+"("+Project.pKey()+"), "+
                    "CONSTRAINT workpackage_fk FOREIGN KEY("+WORKPACKAGE+")
REFERENCES "+
             WorkPackage.tableName()+"("+WorkPackage.pKey()+")"+
         ");";
}
```

Source Code Insert 10: createTable() in code addSql() provides the SQL to add a new entry to the table.

```
public String addSql(){
        String tsend = getString(LATEST_END);
        String tses = getString(EARLY_START);
        String tsa = getString(ACTUAL);
        if(tsend == null)
                tsend = "";
        if(tses == null)
                tses = "";
        if(tsa == null)
                tsa = "";
        return "INSERT INTO "+tableName()+"("+
                ((getValue(P_KEY) == null)?"":P_KEY+",")+
                DESCRIPTION +
                ", "+PROJECT +
                ", "+DURATION +
                ", "+WORKPACKAGE +
                ", "+EARLY_START +
                ", "+LATEST_END +
                ", "+ACTUAL +
                ", "+STRETCHABLE +
                ", "+STIFFNESS +
                ", "+PRIORITY +
                ") VALUES ("+
                ((getValue(P_KEY) == null)?"":getInt(P_KEY)+",")+
                "'"+ psql(getString(DESCRIPTION))+"'"+
                ", "+getInt(PROJECT)+
                ", "+getDouble(DURATION)+
                ", "+getInt(WORKPACKAGE)+
                ", (to_timestamp('"+psql(tses)+"', 'DD/MM/YYYY %H:MI'))" +
                ", (to_timestamp('"+psql(tsend)+"', 'DD/MM/YYYY %H:MI'))" +
                ", (to_timestamp('"+psql(tsa)+"', 'DD/MM/YYYY %H:MI'))" +
                ", "+getBoolean(STRETCHABLE)+
                ", "+getDouble(STIFFNESS)+
                ", "+getInt(PRIORITY)+
        ");";
}
```

Source Code Insert 11: addSql() in code

`updateSql()` provides the SQL to edit an existing entry in the table.

```
public String updateSql() {
        String tsend = getString(LATEST_END);
        String tses = getString(EARLY_START);
        String tsa = getString(ACTUAL);
        if(tsend == null)
                tsend = "";
        if(tses == null)
                tses = "";
        if(tsa == null)
                tsa = "";
        return  "UPDATE "+tableName()+" SET "+
        DESCRIPTION + "=" + "'"+getString(DESCRIPTION)+"'"+", "+
        PROJECT + "=" + getInt(PROJECT)+", "+
        DURATION + "=" + getDouble(DURATION)+", "+
        WORKPACKAGE + "=" + getInt(WORKPACKAGE)+", "+
        EARLY_START + "=" + "(to_timestamp('"+psql(tses)+"', 'DD/MM/YYYY 
%H:MI'))"+", "+
        LATEST_END+ "=" + "(to_timestamp('"+psql(tsend)+"', 'DD/MM/YYYY 
%H:MI'))"+", "+
        ACTUAL + "=" + "(to_timestamp('"+psql(tsa)+"', 'DD/MM/YYYY %H:MI'))"+", "+
        STRETCHABLE + "=" + getBoolean(STRETCHABLE)+", "+
        STIFFNESS + "=" + getDouble(STIFFNESS)+", "+
        PRIORITY + "=" + getInt(PRIORITY)+
        " WHERE "+P_KEY +" = "+getInt(P_KEY)+
        ";";
}
```

Source Code Insert 12: updateSql() in code

Two other classes used to communicate with the database are `DatabaseInterface` and `ResultSet.`

`DatabaseInterface` is described in a previous section, while a ResultSet is a table of data which is generated by executing a statement that queries the database [30].

The class, UserInputDBSettings enables the user to connect to a database of his choice. After the settings are stored, a database connection is established via the GetDatabaseConnection class.

```
public UserInputDBSettings(String dbName, String dbUrl, char[] password, String user)
       {
              this.dbName = dbName;
              this.dbUrl = dbUrl;
              this.driverName = "org.postgresql.Driver";
              this.password = String.copyValueOf(password);
              this.user = user;
       }
       public UserInputDBSettings(String dbName, String dbIP, String dbPort, char[]
password, String user)
       {
              this.dbName = dbName;
              this.dbUrl = "jdbc:postgresql://"+dbIP+":"+dbPort+"/";
              this.driverName = "org.postgresql.Driver";
              this.password = String.copyValueOf(password);
              this.user = user;
       }
```

Source Code Insert 13: UserInputDBSettings constructors

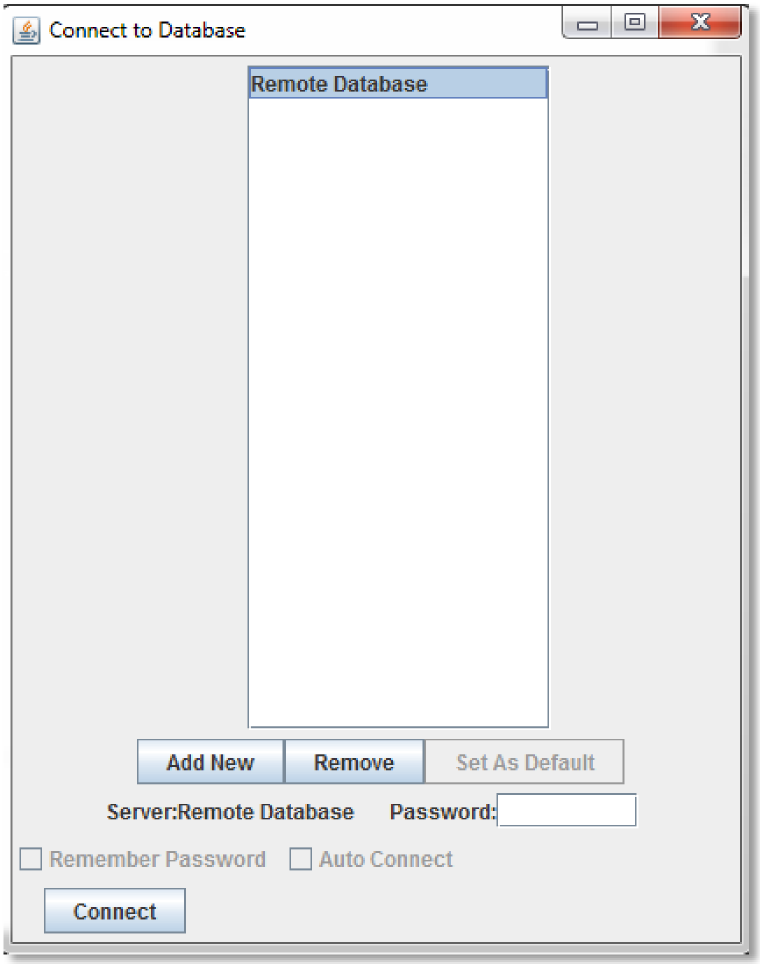
After GetDatabaseConnection establishes a connection to the database, the data values which stores the username and password is cleared to facilitate security.

```
public GetDatabaseConnection (String dbName, String dbUrl, char[] password, String user)
       {
              this.settings = new UserInputDBSettings(dbName, dbUrl, password, user);
              this.dbi = new DataBaseInterface(settings);
              dbName = null;
              dbUrl = null;
              password = null;
              user = null;
       }
       public GetDatabaseConnection (String dbName, String dbIP,String dbPort, char[]
password, String user)
       {
              this.settings = new UserInputDBSettings(dbName, dbIP, dbPort, password,
user);
              this.dbi = new DataBaseInterface(settings);
              dbName = null;
              dbIP = null;
              dbPort = null;
              password = null;
              user = null;

       }
```

Source Code Insert 14: GetDatabaseConnection constructors

The current version of the software automatically connects to a pre-created database on the local machine. Functionality is however added to manually switch to another database, whether the data is located on the local machine or on a remote server. The GUI for switching databases is shown below.



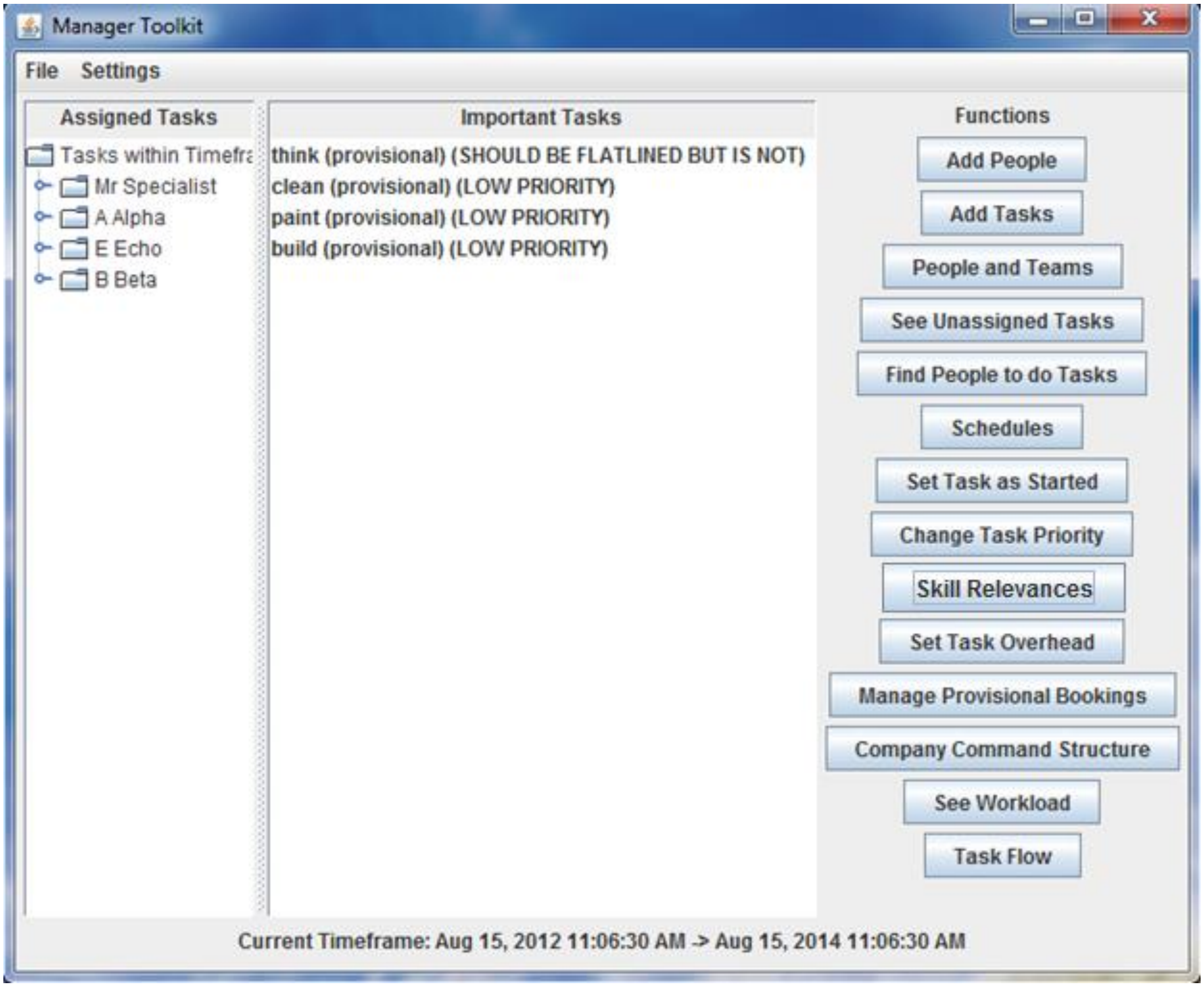
Screenshot 3: Connect to Database GUI window

8.6 DISCUSSING THE GUI IN GENERAL

A graphical user interface was designed that makes the functionalities of program Project Resource Locator available to project managers. In addition to the basic functionalities of managing personnel and tasks, capabilities that provide an overview of the state of projects and personnel are provided, as described in this section.

8.6.1 STARTING UP: THE MANAGER TOOLKIT

The Manager Toolkit screen is displayed at start-up. As shown in screenshot 4, all aspects of the application can be accessed from this screen.



think (provisional) (SHOULD BE FLATLINED BUT IS NOT)
clean (provisional) (LOW PRIORITY)
paint (provisional) (LOW PRIORITY)
build (provisional) (LOW PRIORITY)

Current Timeframe: Aug 15, 2012 11:06:30 AM -> Aug 15, 2014 11:06:30 AM

Screenshot 4: ManagerToolkit window

Important tasks and tasks requiring special attention are displayed in the central panel. This is done with the following code snippet:

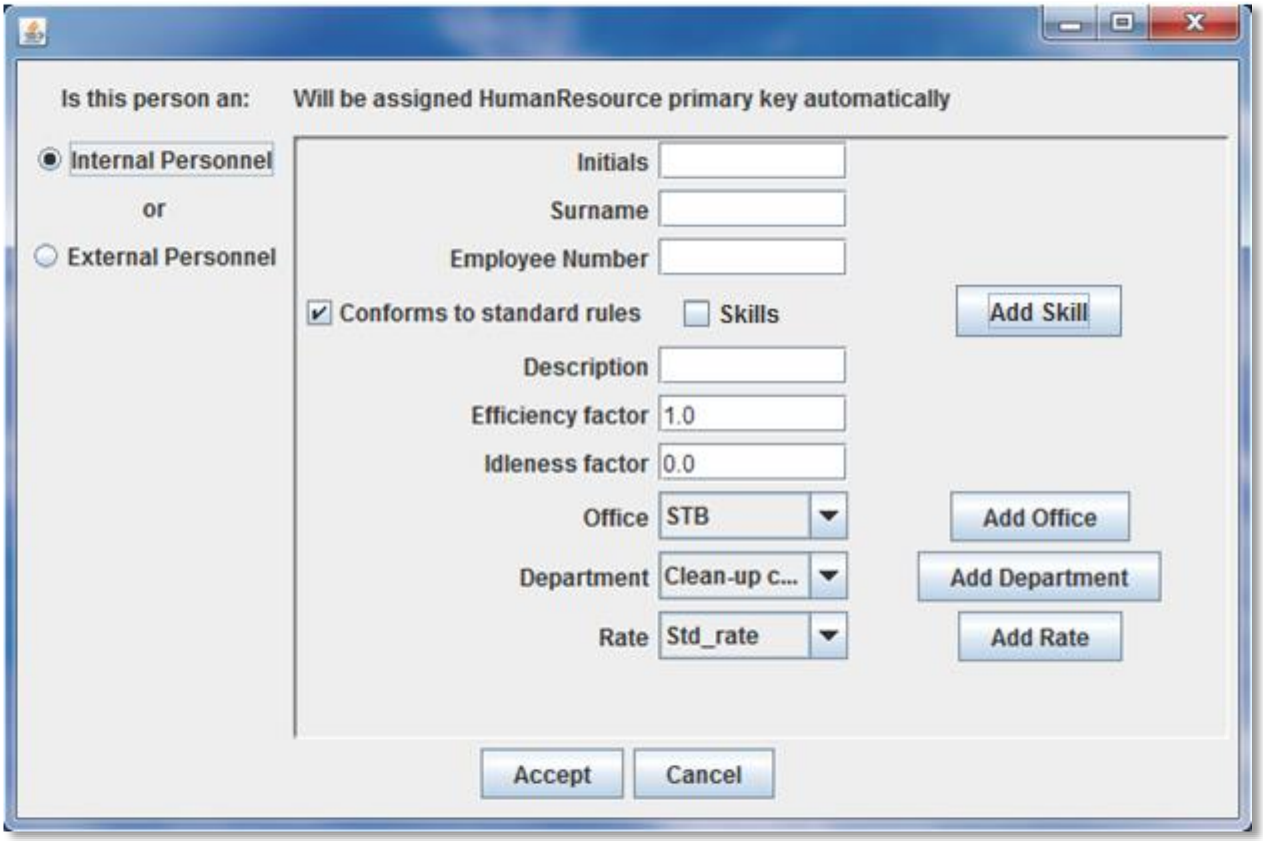
```
Calendar today = Calendar.getInstance();
       for (TaskItem task : main.taskitems)
       {
             Calendar imp = task.getLateststart();
             Calendar imponeweek = imp;
             imponeweek.add(Calendar.WEEK_OF_MONTH, -1);
             if (task.getPriority()==0)
             {
                    main.listmodel.addElement(task.toString()+" (FLATLINE)");
             }
             else if (today.after(imp))
             {
                    main.listmodel.addElement(task.toString()+" (SHOULD BE 
FLATLINED BUT IS NOT)");
             }
             else if (today.after(imponeweek))
             {
                    main.listmodel.addElement(task.toString()+" (GET READY TO 
FLATLINE SOON)");
             }
             else if (task.getPriority()==1)
             {
                    main.listmodel.addElement(task.toString()+" (HIGH PRIORITY)");
             }
             else if (task.getPriority()==2)
             {
                    main.listmodel.addElement(task.toString()+" (NORMAL 
PRIORITY)");
             }
             else if (task.getPriority()==3)
             {
                    main.listmodel.addElement(task.toString()+" (LOW PRIORITY)");
             }

       }
```

Source Code Insert 15: Creating flags for important tasks

8.6.2 ADD PEOPLE

When the *Add People* button is pressed the following screen will be displayed on to the screen:



Is this person an:

Will be assigned HumanResource primary key automatically

● Internal Personnel

or

○ External Personnel

Initials

Surname

Employee Number

☑ Conforms to standard rules

□ Skills

Description

Efficiency factor 1.0

Idleness factor 0.0

Office STB

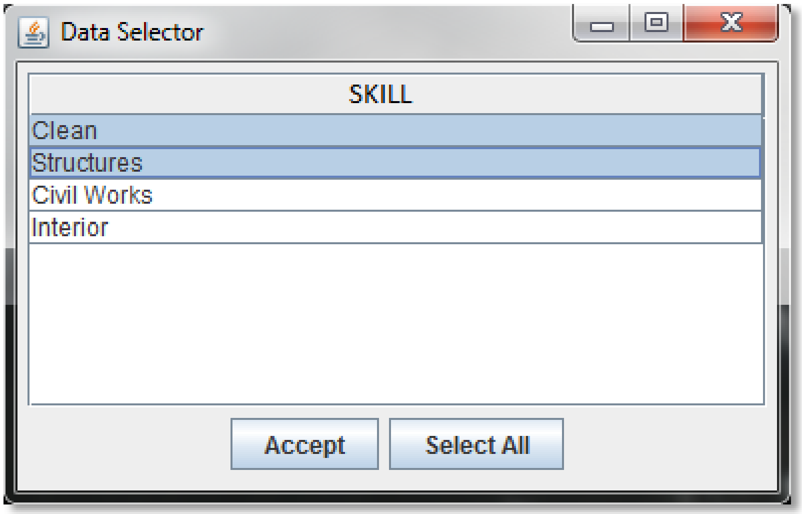
Department Clean-up c...

Rate Std_rate

Screenshot 5: Add People window

On this screen the user can either select Internal Personnel or External Personnel. The choice changes the fields available in the main panel.

Here the user can input all the necessary fields and add an employee to the system. To set which skills the individual has, the following window will be displayed when the check box is ticked:



SKILL
Clean
Structures
Civil Works
Interior

Screenshot 6: General data selector

The shown skills are obtained directly from the database. Any number or combinations of skills can be chosen. Pressing Accept will store the decision. If the desired skill is not shown, it can be manually added by pressing the Add Skill button on the previous screen. The same Data Selector window is used every time a list of attributes have to be selected and new entries can be created using the relevant add-button. To set an individual's office, department and rate, similar dialogs are displayed. The available offices, departments and rates are obtained from the database using the code shown in code insert 14 and 15.

```
DataSelectorMulti ds = new DataSelectorMulti(dbi, "select SKILL.skill_description
from SKILL", frame, "setSkills");
```

Source Code Insert 16: Creating a new DataSelector

```
dataName = sql;
String[] firstsplit = dataName.split(" ");
String [] secondsplit = firstsplit[1].split("\\.", 2);
System.out.println(secondsplit[0]);
dataName = secondsplit[0];
ResultSet rs = new RunQuery(dbi,sql).compute();
while(rs.next())
        {
                Object obj = rs.getObject(1);
                olddata.add(String.valueOf(obj));
        }
tablemodel.addColumn(dataName, olddata.toArray());
```

Source Code Insert 17: Populating the DataSelector

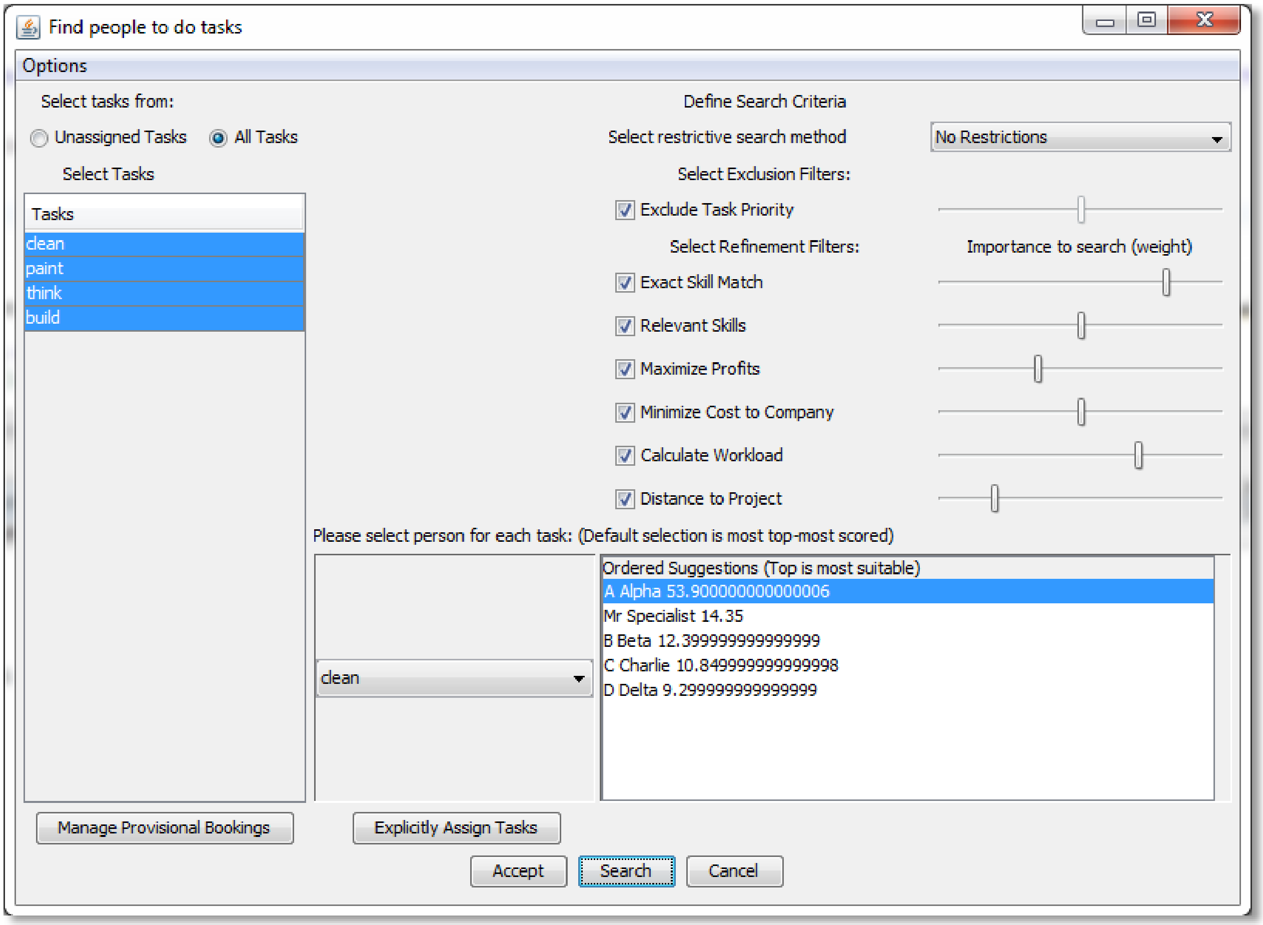
8.6.3 FIND PEOPLE

The primary functionality of the software is the allocation of tasks, or as it is designated in the GUI, Finding People to do Tasks. On the left hand side of the window, shown in screenshot 7, the user can select whether to see only the unassigned tasks, or all the tasks. Once the user sees the desired tasks, any task, or combination of them, can be selected for inclusion in the search. On the right-hand side of the window the filter options can be set. Specific filters can be turned on or off, or their relative importance can be changed. When the screen starts up for the first time the settings are set to the default values, which were deemed to represent good engineering judgement.

In Options, on the menu bar, the user can set what percentage of the total possible personnel to see in the final answer. If the user only wants to see the top 10%, this is what makes it possible.

When the user is satisfied with the filter settings, the *Search* button at the bottom of the screen can be pressed to start the procedure. The user should expect a delay since the filtration process is computationally expensive. When the calculations are done the results will be displayed in the panel near the right-hand bottom corner of the window. The user can browse through the specific tasks that were pre-selected and the display will automatically change to display the top choices for each task. The personnel are sorted according to the scores they received, with the top scoring individual always being automatically selected. If the user instead prefers the task be assigned to another individual, he/she simply has to select the individual from the list. When the user is satisfied with the allocations, pressing the *Accept* button will create provisional bookings and store them in the database.

If a user already knows who he wants to do a task, he/she does not have to go through the whole search procedure as there is a method available to explicitly assign tasks to individuals. This method does not go through any filters, and is thus not guaranteed to be optimal. The *Explicitly Assign Tasks* button at the bottom of the screen can be pressed to accomplish this.



Select tasks from:

○ Unassigned Tasks ● All Tasks

Select Tasks

Tasks
clean
paint
think
build

Define Search Criteria

Select restrictive search method: No Restrictions

Select Exclusion Filters:

☑ Exclude Task Priority

Select Refinement Filters: Importance to search (weight)

☑ Exact Skill Match

☑ Relevant Skills

☑ Maximize Profits

☑ Minimize Cost to Company

☑ Calculate Workload

☑ Distance to Project

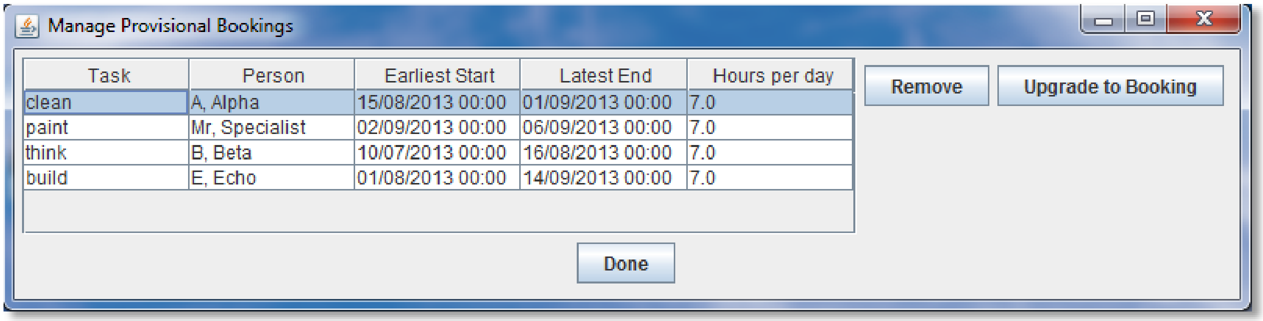
Please select person for each task: (Default selection is most top-most scored)

clean

Ordered Suggestions (Top is most suitable)
A Alpha 53.900000000000006
Mr Specialist 14.35
B Beta 12.399999999999999
C Charlie 10.849999999999998
D Delta 9.299999999999999

Screenshot 7: Find People window

After the provisional bookings have been stored, the user can view them by pressing the *Manage Provisional Bookings* button available on the *ManagerToolkit* window.



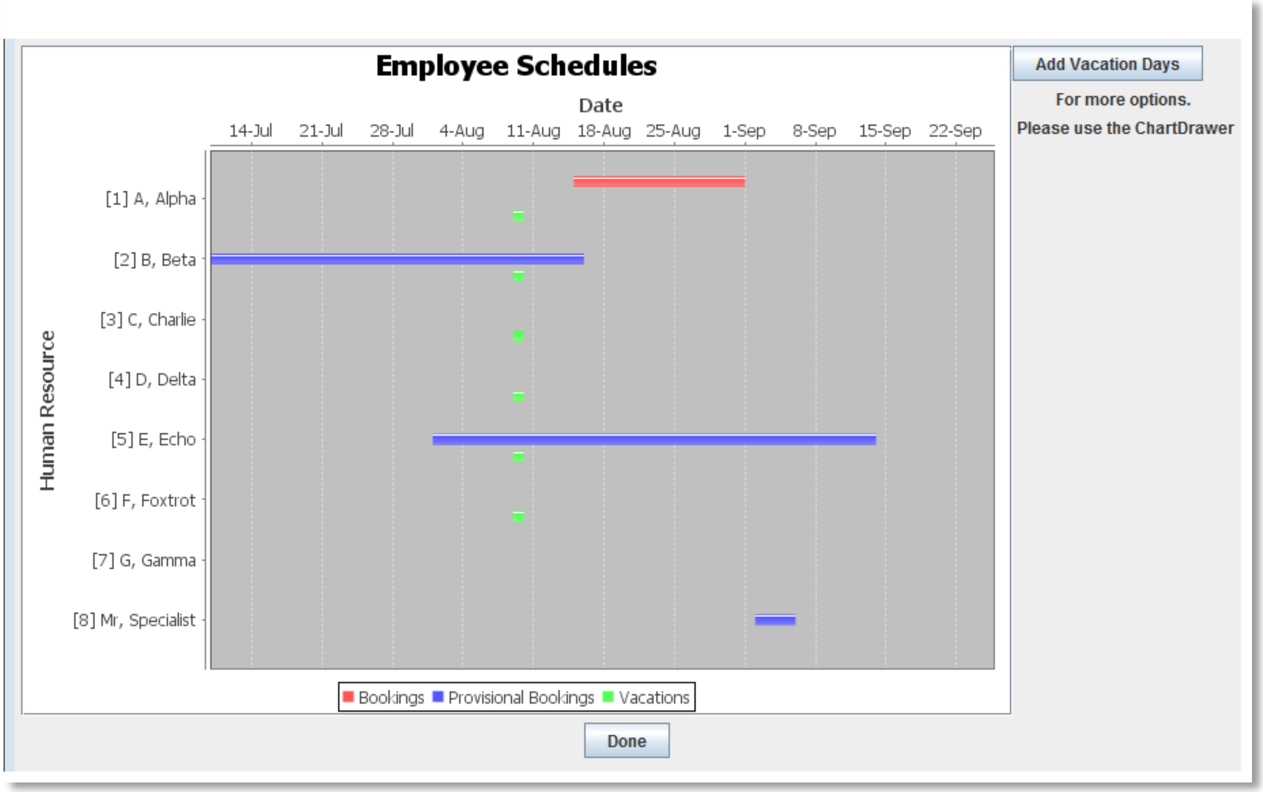
Task	Person	Earliest Start	Latest End	Hours per day
clean	A, Alpha	15/08/2013 00:00	01/09/2013 00:00	7.0
paint	Mr, Specialist	02/09/2013 00:00	06/09/2013 00:00	7.0
think	B, Beta	10/07/2013 00:00	16/08/2013 00:00	7.0
build	E, Echo	01/08/2013 00:00	14/09/2013 00:00	7.0

Screenshot 8: Manage Provisional Bookings window

From this window it is possible to either delete provisional bookings, or upgrade them to confirmed bookings.

8.6.4 SCHEDULES

If the Schedules-button on the ManagerToolkit window is depressed the window displays the data in Gannt chart format. The chart can be zoomed in by clicking and dragging the cursor to the right. The timeframe specified in the ManagerToolkit determines the outermost bounds of the schedule. Different colours on the graph represent different types of bookings. Red for normal bookings: blue for provisional bookings, and green for vacations. Vacations can be assigned to individuals by pressing the *Add Vacation Days* button on the right-hand side.

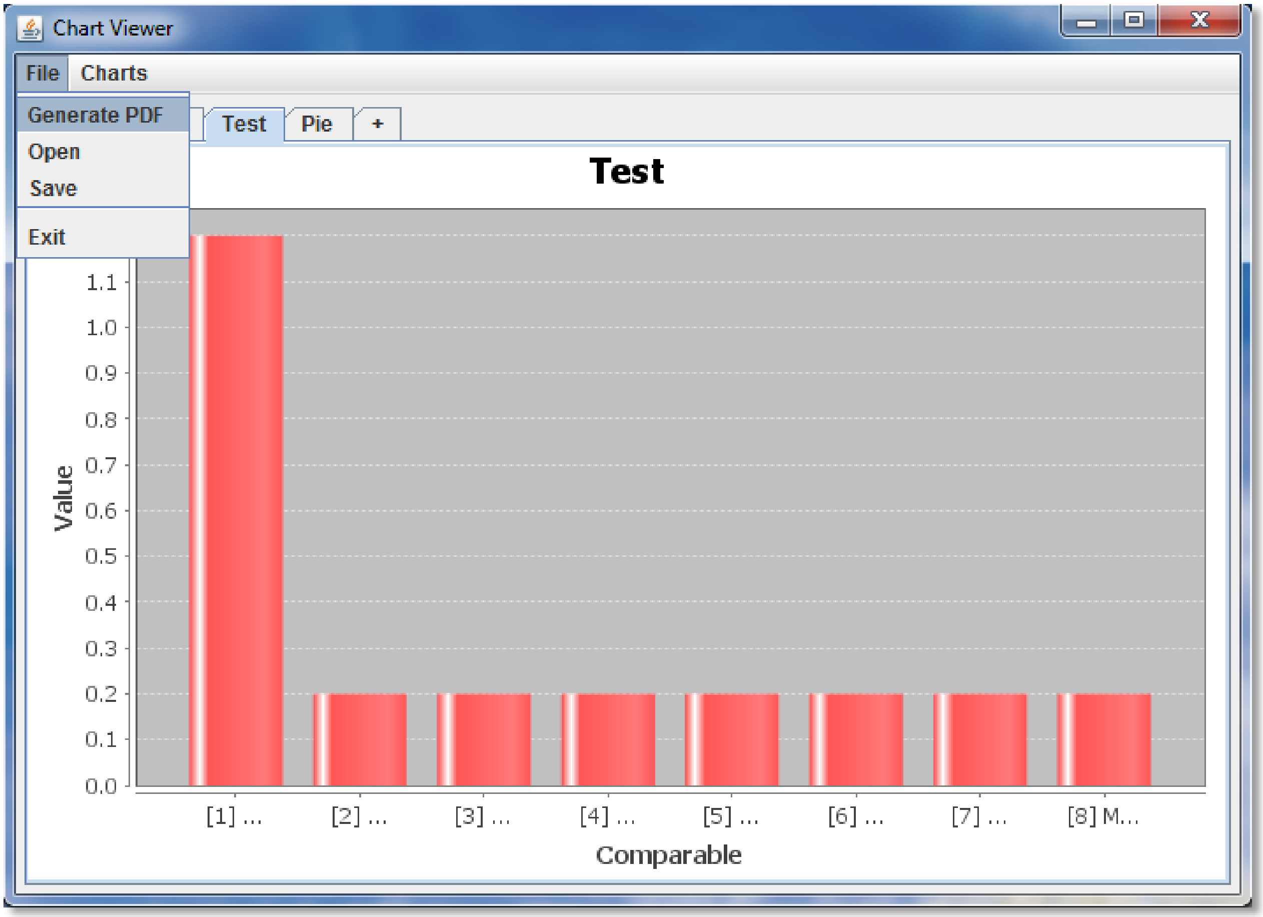


For more options.
Please use the ChartDrawer

Screenshot 9: Schedule Viewer window

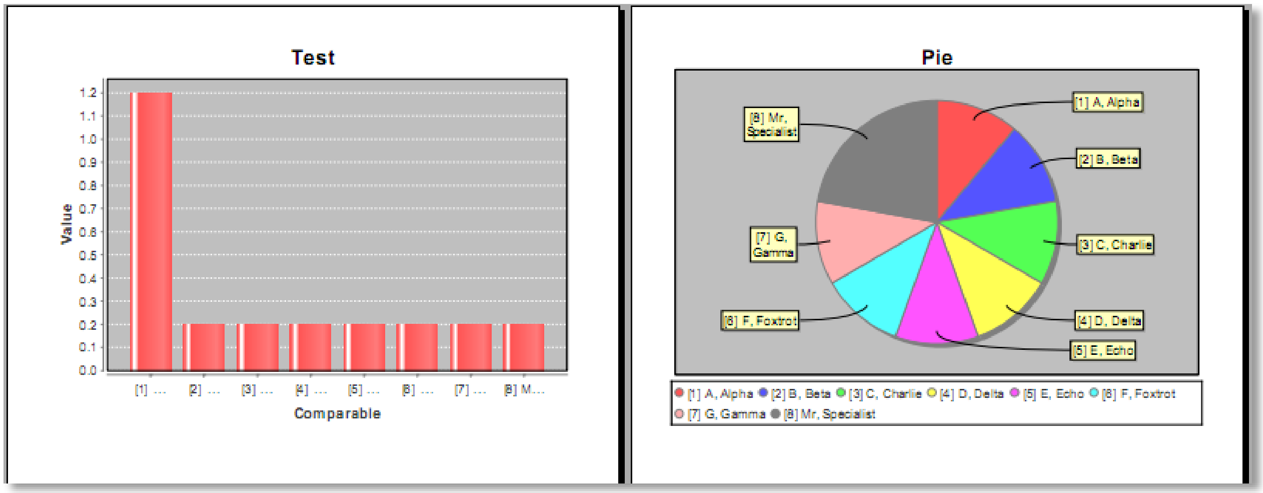
8.6.5 BUILDING REPORTS: CHART DRAWER

A Chart Viewer can be launched using the File-menu of the ManagerToolkit window.



Screenshot 10: Chart Viewer with menu window

Clicking on either the "+" sign in the tabs, or going to Charts – Create Chart, will open a new window, called the Chart Creator, which enables the user to create custom charts. While the user is creating a collection of charts, the collection can be saved using Java serialisation. To accomplish this the user should click on File – Save in the menu. Charts can be browsed via the tabs at the top of the window. Right-clicking on the chart will give the user options to completely customise the look of the chart. Some charts can also be zoomed in by left-click dragging to the right. If the user has completed adding charts and is satisfied with his report, the collection can be exported to a PDF file. Charts are projected on the top half an A4 sheet, with each chart taking up a page on its own. This can be accomplished by clicking on File – Export to PDF. This PDF can then be included into another report if so desired.



Screenshot 11: Example resulting PDF export

When creating charts with the Chart Creator, standard charts can be quickly generated via the buttons on the quick pick panel. When selected, individual datasets can be altered if so desired. A completely custom graph can be generated by using the lists inside the custom chart panel. The chart type can be set along with which data both axes should be populated. Some charts have sub-types which change how they look, for example pie charts can be viewed standard, or exploded. It should be noted that some datasets cannot be compared by certain types of charts. Bar Charts must compare a description with a numerical value. Two values cannot be compared to one another. Checking has been implemented in the software for these cases.

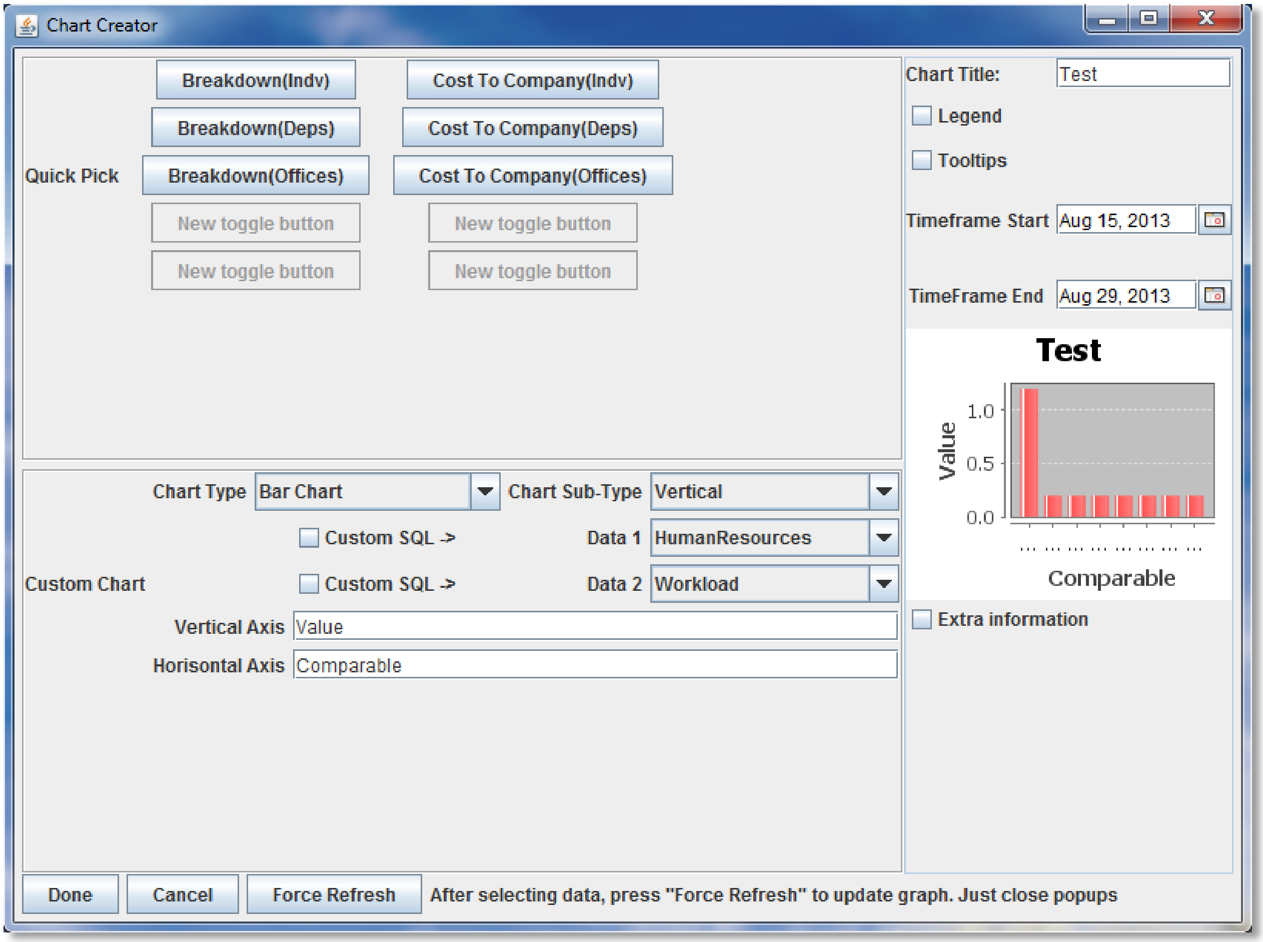


Chart Creator

Chart Title: Test

□ Legend

□ Tooltips

Timeframe Start Aug 15, 2013

TimeFrame End Aug 29, 2013

□ Extra information

Chart Type Bar Chart

Chart Sub-Type Vertical

□ Custom SQL -> Data 1 HumanResources

□ Custom SQL -> Data 2 Workload

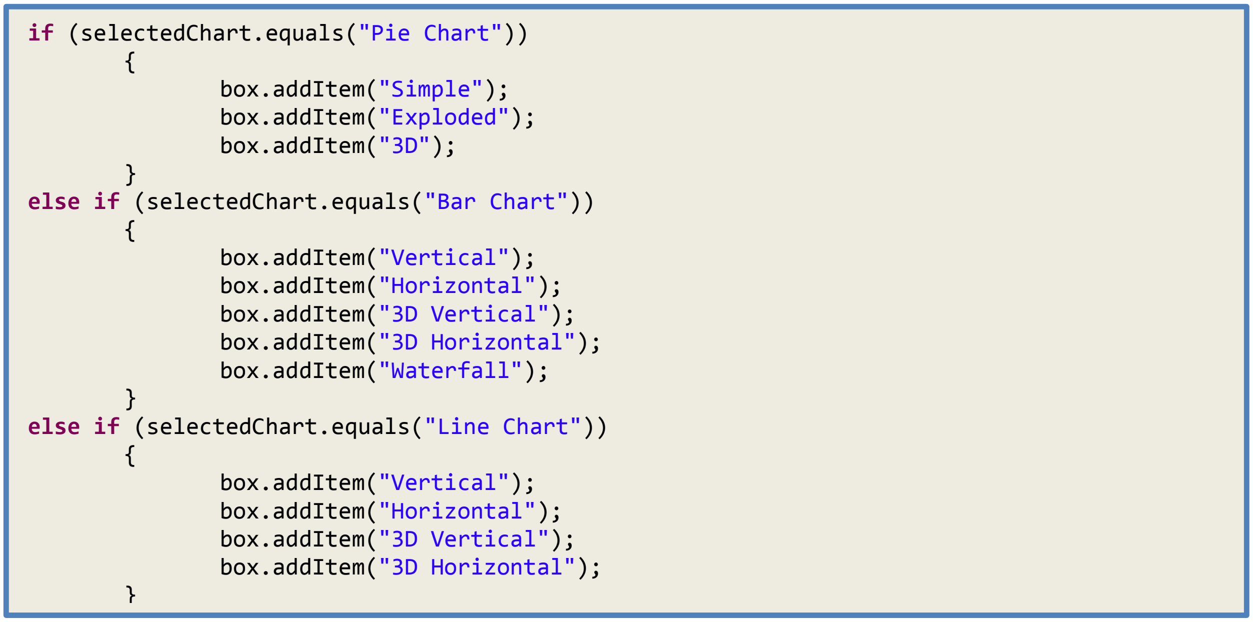
Vertical Axis Value

Horisontal Axis Comparable

After selecting data, press "Force Refresh" to update graph. Just close popups

Screenshot 12: Chart Creator window

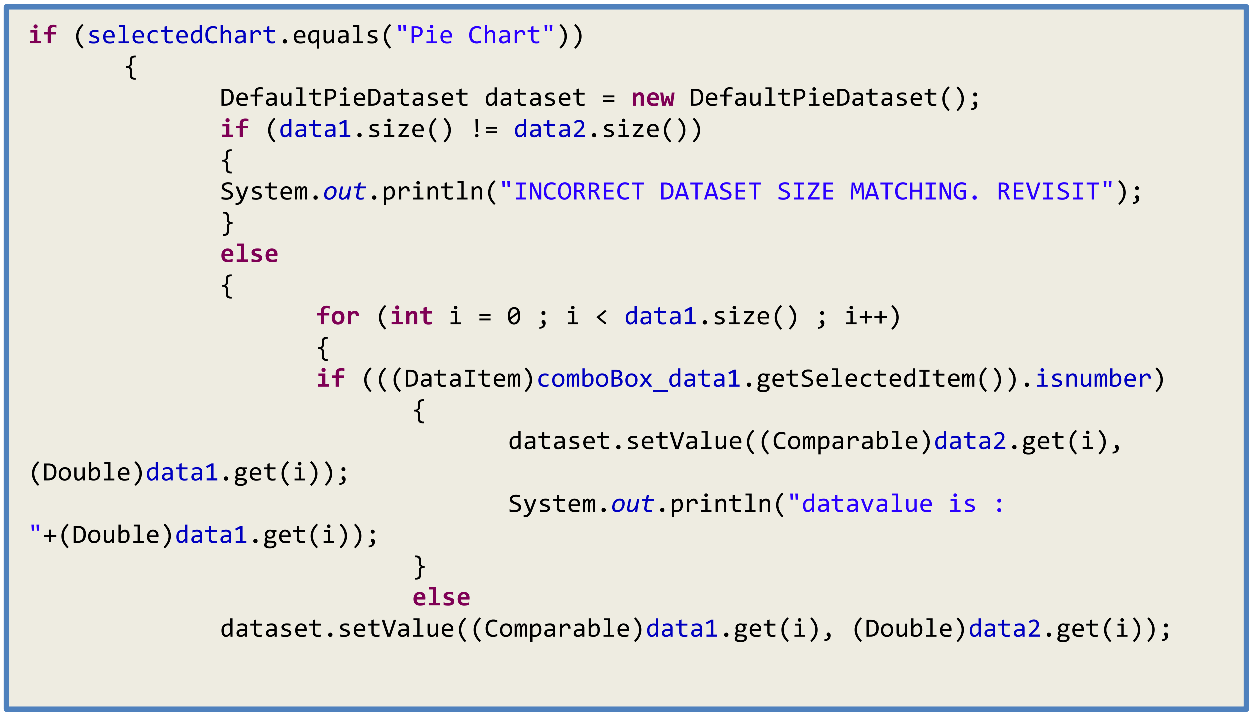
Drop down lists are populated as soon as the Chart Creator is opened, with the following code serving as an example.



```
if (selectedChart.equals("Pie Chart"))
        {
                box.addItem("Simple");
                box.addItem("Exploded");
                box.addItem("3D");
        }
else if (selectedChart.equals("Bar Chart"))
        {
                box.addItem("Vertical");
                box.addItem("Horizontal");
                box.addItem("3D Vertical");
                box.addItem("3D Horizontal");
                box.addItem("Waterfall");
        }
else if (selectedChart.equals("Line Chart"))
        {
                box.addItem("Vertical");
                box.addItem("Horizontal");
                box.addItem("3D Vertical");
                box.addItem("3D Horizontal");
        }
```

Source Code Insert 18: Populating sub-type dropdown lists

Depending on the choice of chart, datasets are constructed differently.



```
if (selectedChart.equals("Pie Chart"))
        {
                DefaultPieDataset dataset = new DefaultPieDataset();
                if (data1.size() != data2.size())
                {
                System.out.println("INCORRECT DATASET SIZE MATCHING. REVISIT");
                }
                else
                {
                        for (int i = 0 ; i < data1.size() ; i++)
                        {
                        if (((DataItem)comboBox_data1.getSelectedItem()).isnumber)
                                {
                                        dataset.setValue((Comparable)data2.get(i),
(Double)data1.get(i));
                                        System.out.println("datavalue is :
"+(Double)data1.get(i));
                                }
                                else
                dataset.setValue((Comparable)data1.get(i), (Double)data2.get(i));
```

Source Code Insert 19: Creating a chart dataset according to choices

8.7 CODE DIAGRAMS

Simplified versions of various parts of the code is visualised using Unified Modelling Language (UML) diagrams, as shown below. The diagrams provide an overview of how the code is connected together to yield various results.

8.7.1 DRAWING CHARTS

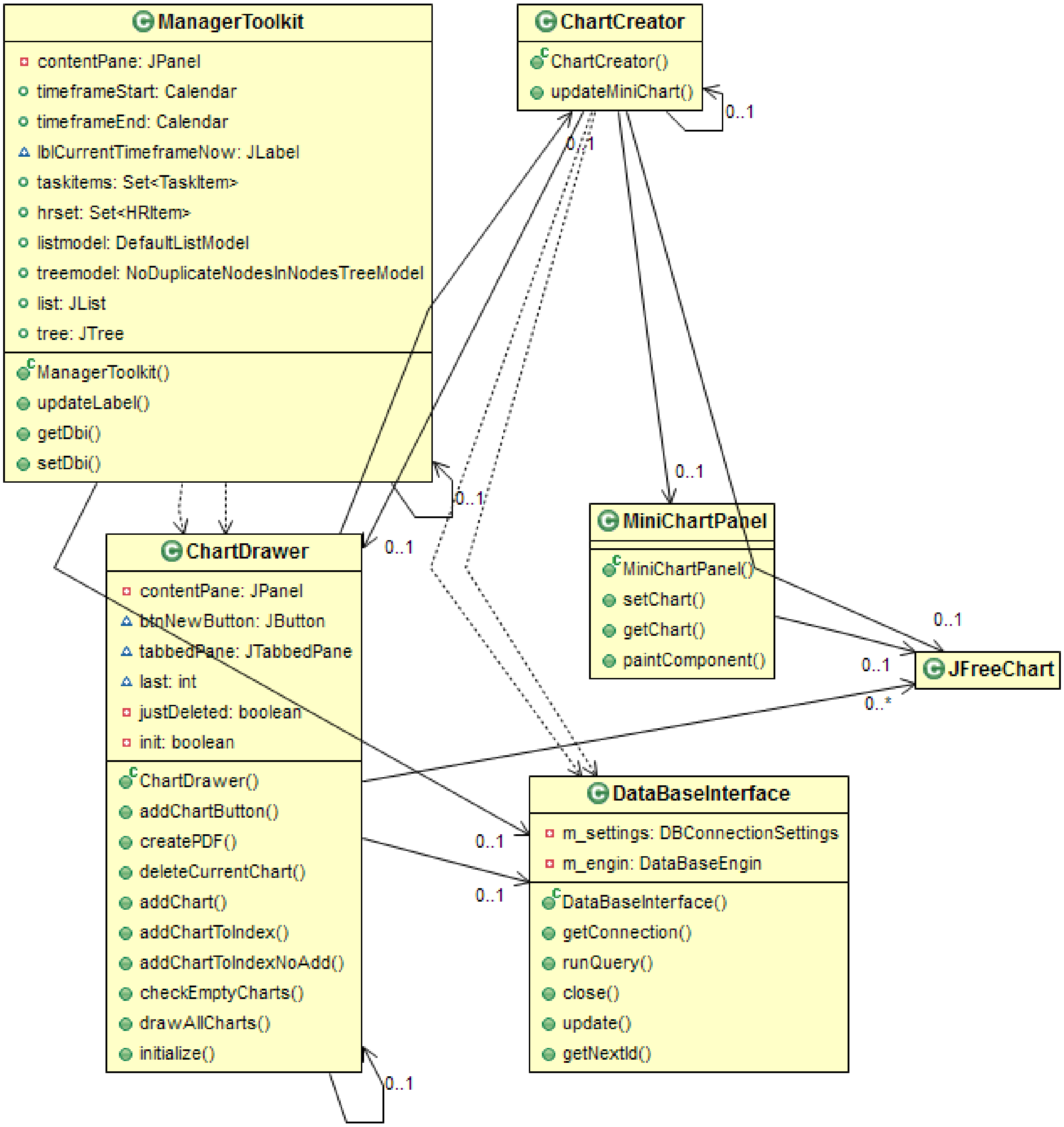


Figure 11: UML – Drawing Charts

8.7.2 FIND PEOPLE

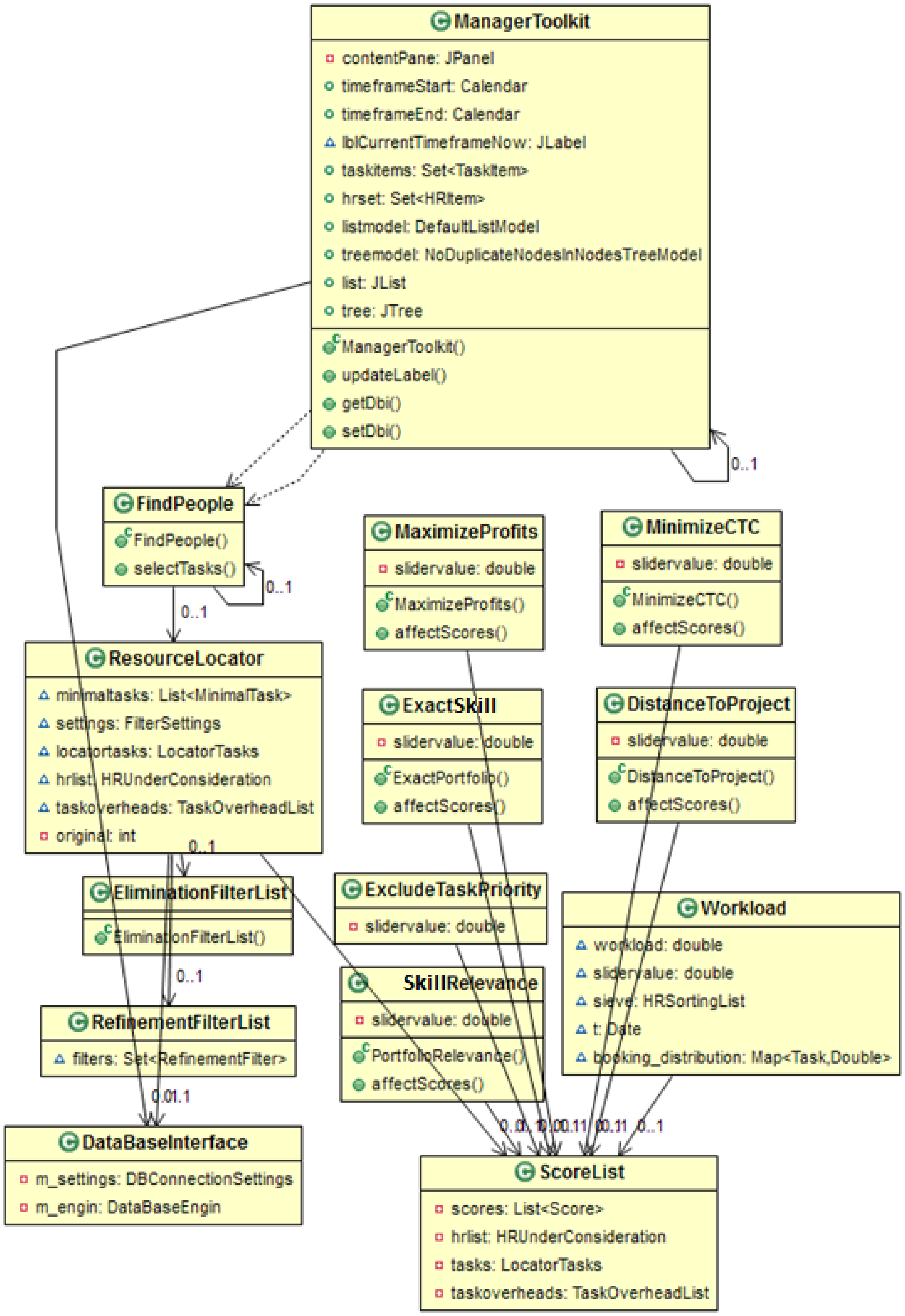


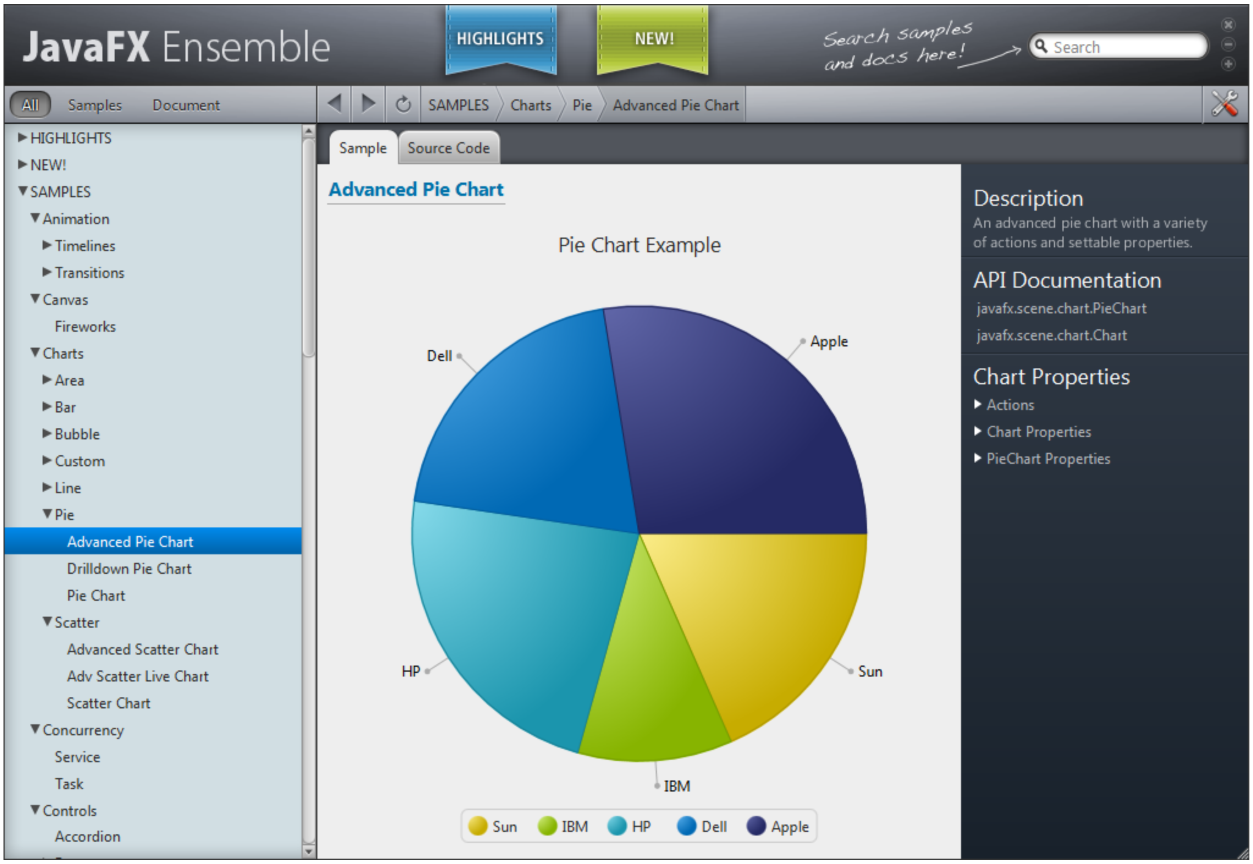
Figure 12: UML – Finding People

9. RECOMMENDATIONS FOR FURTHER WORK

In this document a technique was developed to calculate perceived workload to aid project managers to better allocate tasks. Supporting software was also developed with which an organisation with all its offices and employees can be modelled. The software also provides additional features which aims to aids project managers in management and transparency thereof by enabling the generation of reports.

The software operates and it yields proper results. However, any software package can always be improved and redesigned. A list of elements that can be improved or added now follows:

- The code can be rewritten and optimised. It is believed the program can be made smaller and more responsive.
- The GUI was developed using the Java Swing framework. It should be rewritten using Java FX. JavaFX, which will be released soon [31], will make the software look and feel much



An advanced pie chart with a variety of actions and settable properties.

more professional. Currently major workarounds is used that could have been avoided with this new technology.

- The entire project should be moved online, with the controlling database located on a remote server, and the graphical user interface imbedded into a website. This can be accomplished by using Java Server Pages or the new JavaFX.
- Currently not all types of charts are implemented in the Chart Drawer. More charts can be added to offer more functionality.
- Additional filters should be added to the search method to increase accuracy and flexibility:
    - People higher up in an organisation command structure should be allocated less work, as the pressure of command and organisation is probably increased. These individuals should be focusing on running a company.
    - A method to force the spacing of individuals throughout the length of a project is needed. This is to prevent the same skilled individual from working on a number of consecutive tasks. The task-graph is available and can be used for this purpose. The p-distribution discussed in a paper by Paul J. Maliszewski, Michael J. Kuby and Mark W. Horner called “A comparison of multi-objective spatial dispersion models for managing critical assets in urban areas” could be found useful.
- It is possible that the logic of the object model can be revisited for future implementations. It is possible that some logic of the model can be simplified.
- If this software is developed for use in a real business environment, the model can be further refined by discussing, with system analysts, what typical standardised organisation models are used to define the typical business project, office or department.
- Although physical distance from project locations are already brought into the procedure, the importance of time zones and day-night differences can possible be accentuated specifically.

10. CONCLUSION

The primary objective of the research described in this thesis was to develop and implement a technique by which project managers can allocate tasks to members of a project team in an engineering environment. As part of the process an equation to calculate an individual's perceived workload was developed and implemented. A survey tested engineering manager's response to the equation's logic, and the results were positive. Other criteria that influence the allocation of tasks were also investigated and included in the selection procedure. The prototype software enables a project manager to effectively allocate tasks to individuals within an organisation. The software also aids in the generation of charts that can be used in the management of the organisation. Most aspects of the prototype software were designed with growth in mind and can be developed further.

It is believed that this software, and software like it, will truly make a difference in the engineering world. Automation is the key to time saving, and thus to saving money. A large amount of time and money is currently spent on project management, and software can assist project managers to work more efficiently and obtain better results. The future of project management software is promising and exciting. The challenge, however, is to convince companies and organisation s to adopt and improve these advances in practise.

REFERENCES

1. Oxford University Press. Oxford Dictionaries Online. [Online]: Oxford University Press; 2007 [cited 2013 August 13. Available from: http://oxforddictionaries.com/definition/english]

2. Nicholas JM, Steyn H. Project Management for Business, Engineering, and Technology - Principles and Practise. 3rd ed. Oxford: Butterworth-Heinemann; 2008.

3. van Rooyen GC. Thesis discussion. In ; 2012; Stellenbosch. p. January - October, Once a week.

4. Brooks FP. The Mythical Man-Month. Anniversary Edition ed: Addison-Wesley; 1995.

5. Coviello D, Ichino A, Persico N. Time Allocation and Task Juggling. [Online]; 2011 [cited 2013 September 24. Available from: http://federation.ens.fr/ydepot/semin/texte1213/NIC2013TIM.pdf]

6. Algonquin College of Applied Arts and Technology. Algonquin College: Standard Workload Formula (SWF). [Online]; 2013 [cited 2013 September 24. Available from: http://www3.algonquincollege.com/hr/managers-toolkit/talent-workload-management/standard-workload-formula/#assignment]

7. Tsai YL, Huang KC, Chang HY, Ko J, Wang ET, Hsu CH. Scheduling Multiple Scientific and Engineering Workflows Through Task Clustering and Best-Fit Allocation. [Online]; 2012 [cited 2013 September 24. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6274023]

8. Kartik S, Siva Ram Murthy C. Improved Task- Allocation Algorithms to Maximize Reliability of Redundant Distributed Computing Systems. [Online]; 1995 [cited 2013 September 24. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=475976]

9. Koshijima I, Tomio U. Human Resource Allocation in Project Management: Management Science Approach. [Online]; 2003 [cited 2013 September 24. Available from: http://www.sba.muohio.edu/abas/2001/brussels/Koshijima_Koshijima-Umeda.pdf]

10. Estellon B, Gardi F, Nouioua K. High-performance local search for task scheduling with human resource allocation. [Online]; 2007 [cited 2013 September 24. Available from: http://pageperso.lif.univ-mrs.fr/~bertrand.estellon/recherche/EGN_SLS_09.pdf]

11. Microsoft. Microsoft Office. [Online]; 2013 [cited 2013 09 24. Available from: http://office.microsoft.com/en-us/project/]

12. Bonitasoft. BonitaSoft. [Online]; 2013 [cited 2013 09 24. Available from: http://www.bonitasoft.com/]

13. 24SevenOffice. 24SevenOffice. [Online]; 2013 [cited 2013 09 24. Available from: http://24sevenoffice.com/]

14. BrightWork. BrightWork - Project and Portfolio Management on SharePoint. [Online]; 2013 [cited 2013 09 24. Available from: http://www.brightwork.com/index.htm]

15. Clarity International Pty Ltd. Clarity - Simplfying Operations. [Online]; 2011 [cited 2013 09 24. Available from: http://www.clarity.com/]

16. Pringle D, Farrance M. Bonitasoft - How to Get the Human Resources Information Systems You Need. [Online] Grenoble, France; 2012 [cited 2013 August 14. Available from: http://www.bonitasoft.com/system/files/documentation_library/how_to_get_the_hris_you_need_010213.pdf]

17. Harrington JL. SQL Clearly Explained. In Harrington JL. The Relational Data Model. San Francisco: Morgan Kaufmann Publishers; 2010. p. 3-21.

18. Czerwinski M, Horvitz E, Wilhite S. A Diary Study of Task Switching and Interruptions. PDF. Redmond: Microsoft, Microsoft Research.

19. D M, A VH, D DB, S V, M G, T D, et al. Determining a set of measurable and relevant factors affecting nursing workload in the acute care hospital setting: a cross-sectional study. Study. Belgium: University Hospital of Ghent, Department of Nursing; 2011. Report No: PMID: 22030021.

20. Brunies R, Emir Z. The Revay Report- Calculating Loss of Productivity Due to Overtime. [Online] Montreal; 2001 [cited 2013 August 19. Available from: http://www.danzpage.com/Construction-Management-Resources/Calculating_Loss_of_Productivity_Due_to_OT_Using_Charts_-_Nov_2001.pdf]

21. Firmenich B. System Design of an Open Engineering Platform. Masters Course in Civil Engineering Informatics. Weimar: Bauhaus-Universit¨ at Weimar, Fakulat at Bauingenieurwesen; 2002.

22. Pahl PJ. Fundamentals Of Technical Optimization. Class Notes. Stellenbosch: Institute of Structural Engineering; Stellenbosch University, Civil Engineering Informatics; 2012.

23. Codd EF. A relational model of data for large shared data banks. Communications of the ACM. 1970 June; 13(6): p. 379.

24. Stoll RR. Set Theory and Logic. 1st ed. Dover Publications I, editor. Mineola: W.H Freeman and Company; 1963.

25. Kau A. Normalization. 2012. GPCG PATIALA.

26. Theron LF. Java Based Construction Plant Management System. Final-year Dissertation. Stellenbosch: University of Stellenbosch, Civil Engineering Informatics; 2011.

27. Movable Type Scripts. Calculate distance, bearing and more between Latitude/Longitude points. [Online]; 2013 [cited 2013 September 3. Available from: http://www.movable-type.co.uk/scripts/latlong.html]

28. Hirsch KA, Reichardt H. The VNR Concise Encyclopedia of Mathematics. 2nd ed. Gellert W, Kustner H, Hellwich M, Kastner H, editors. New York: Van Nostrand Reinhold; 1989.

29. Google Inc. Google Maps. [Online]; 2013 [cited 2013 September 2. Available from: https://maps.google.co.za/]

30. Oracle. Java SE 6 Documentation. Software Documentation. Redwood City:; 2011.

31. Oracle. JavaFX Frequently Asked Questions. [Online]; 2013 [cited 2013 September 3. Available from: http://www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html#5]

32. Stroop JR. STUDIES OF INTERFERENCE IN SERIAL VERBAL REACTIONS. Journal of Experimental Psychology. 1935;(18).

APPENDIX A

WORKLOAD QUESTIONNAIRE - EXAMPLE

Name:……………………………………………………………………….
Position:…………………………………………………………………..
Company:………………………………………………………………….

The term "workload" is used to describe how busy a person is. The aim is to use it as an indication of whether said person can handle additional tasks. An individual's workload is expressed as a percentage:

$$workload = \frac{project\ hours\ per\ day}{hours\ in\ work\ day} * 100\%,$$

Project hours are the total number of hours an individual is expected to work on all tasks assigned to him. The higher an individual's workload is, the busier he/she will be, which increases the risk when a new task is assigned to him.

It is a well-known time saving method to group tasks together in batches so that they can be focused on and completed together. This is based on the simple logic that people with more to do get more done. The question is how the number of tasks influences the workload: Is the workload of a person with 2 times 4 hour tasks higher than that of a person with one 8 hour task?

The questions are very simple. A scenario is presented, with a few stated workload values. Please highlight (or in some way mark) the most appropriate workload. In all cases an 8 hour workday is assumed.
When the scenario reads: "1 Task of 8 hours", it means that an individual has one task assigned to him for a given day and that task is estimated to take 8 hours to complete.

Please select your most appropriate workload value for all of the following scenarios:

Scenario Description	***Please highlight the most appropriate workload percentage***		
1 Task of 8 hours.	95%	100%	105%
2 Tasks of 4 hours each.	100%	101.5%	115%
5 Tasks, durations add up to 8 hours.	100%	112%	124%
1 Task of 8 hours, but person works twice as fast as everybody else.	50%	75%	100%
1 Task of 8 hours, but person works twice as slow as everybody else.	100%	200%	400%
3 Tasks of 1.33 hours each.	25%	52%	70%
10 short tasks, each just 24 minutes long. (4 Hours)	50%	82%	100%
1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.	50%	70%	100%

INDIVIDUAL SURVEY ANSWERS

1. Please give your name, company and position inside the company:

Willie Enright Wateright Consulting Managing Director

2. 1 Task of 8 hours

95%

3. 2 Tasks of 4 hours each.

100%

4. 5 Tasks, durations add up to 8 hours.

112%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

75%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

200%

7. 3 Tasks of 1.33 hours each.

70%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

82%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

70%

1. Please give your name, company and position inside the company:

Johann Malan, Aurecon, Civil Engineer

2. 1 Task of 8 hours

100%

3. 2 Tasks of 4 hours each.

101.5%

4. 5 Tasks, durations add up to 8 hours.

124%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

75%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

200%

7. 3 Tasks of 1.33 hours each.

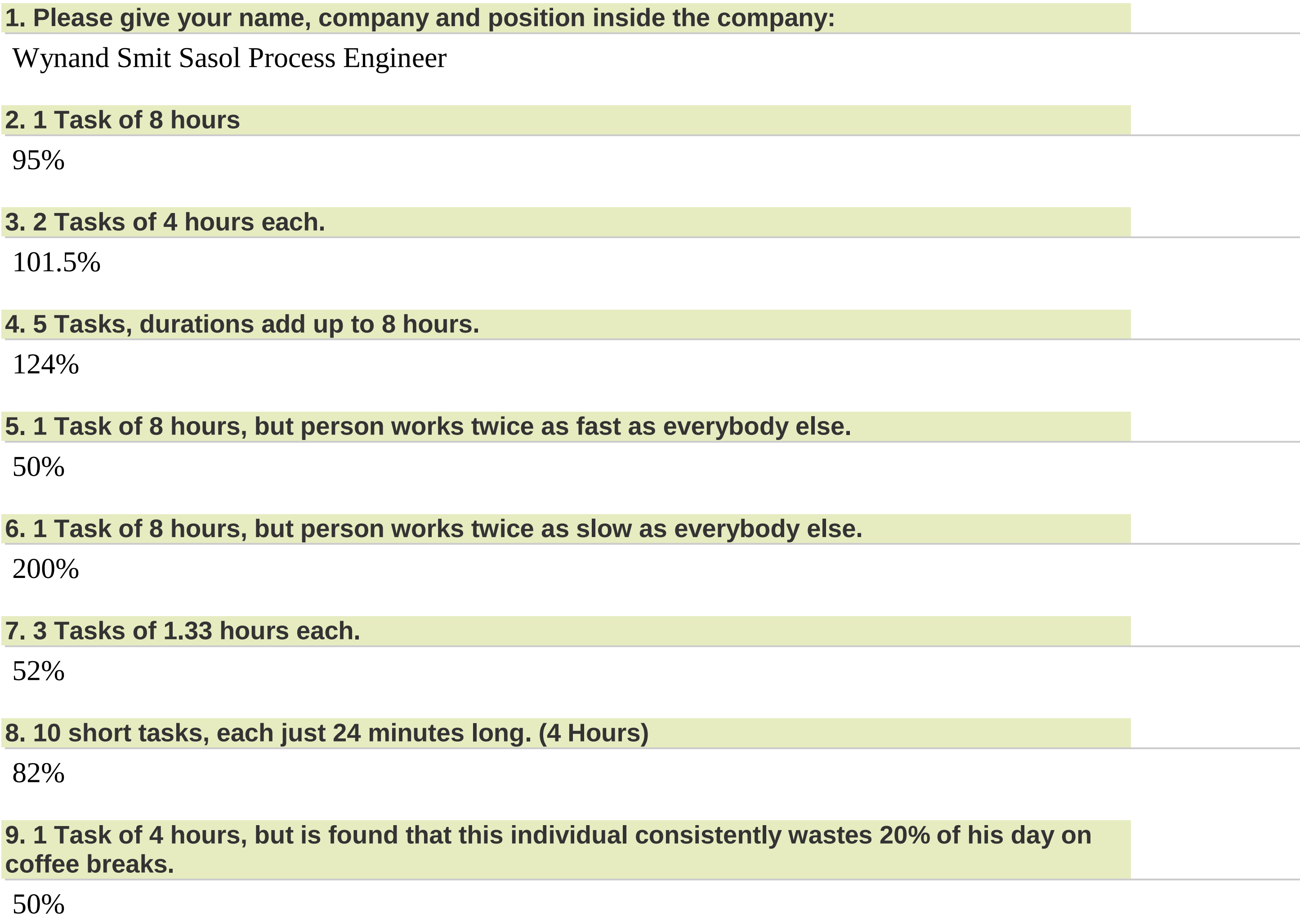
52%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

82%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

70%



1. Please give your name, company and position inside the company:

Wynand Smit Sasol Process Engineer

2. 1 Task of 8 hours

95%

3. 2 Tasks of 4 hours each.

101.5%

4. 5 Tasks, durations add up to 8 hours.

124%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

50%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

200%

7. 3 Tasks of 1.33 hours each.

52%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

82%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

50%

1. Please give your name, company and position inside the company:

Petrus Theron, Mercenary Coder

2. 1 Task of 8 hours

100%

3. 2 Tasks of 4 hours each.

115%

4. 5 Tasks, durations add up to 8 hours.

124%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

75%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

200%

7. 3 Tasks of 1.33 hours each.

70%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

82%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

100%

1. Please give your name, company and position inside the company:

Cameron Crombie Mechanical Master's Student, University of Stellenbosch

2. 1 Task of 8 hours

95%

3. 2 Tasks of 4 hours each.

115%

4. 5 Tasks, durations add up to 8 hours.

124%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

75%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

100%

7. 3 Tasks of 1.33 hours each.

70%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

82%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

100%

1. Please give your name, company and position inside the company:

Melissa Kistner, Process Engineer - Sasol, Stellenbosch University

2. 1 Task of 8 hours

105%

3. 2 Tasks of 4 hours each.

115%

4. 5 Tasks, durations add up to 8 hours.

124%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

50%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

200%

7. 3 Tasks of 1.33 hours each.

70%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

100%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

70%

1. Please give your name, company and position inside the company:

Miga JES Farms Farm Hand

2. 1 Task of 8 hours

95%

3. 2 Tasks of 4 hours each.

100%

4. 5 Tasks, durations add up to 8 hours.

112%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

100%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

200%

7. 3 Tasks of 1.33 hours each.

52%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

100%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

50%

1. Please give your name, company and position inside the company:

G VAN HEERDEN STRUCTURAL ENGINEER KLS CONSULTING ENGINEERS

2. 1 Task of 8 hours

95%

3. 2 Tasks of 4 hours each.

100%

4. 5 Tasks, durations add up to 8 hours.

112%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

50%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

100%

7. 3 Tasks of 1.33 hours each.

25%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

50%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

50%

1. Please give your name, company and position inside the company:

Pieter Eduard de Kock MFM 92.6 Presenter/Producer/Technician

2. 1 Task of 8 hours

100%

3. 2 Tasks of 4 hours each.

100%

4. 5 Tasks, durations add up to 8 hours.

100%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

50%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

200%

7. 3 Tasks of 1.33 hours each.

52%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

50%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

70%

1. Please give your name, company and position inside the company:

Nico-Ben de Villiers, MEng(Informatics) Candidate at the University of Stellenbosch.

2. 1 Task of 8 hours

100%

3. 2 Tasks of 4 hours each.

100%

4. 5 Tasks, durations add up to 8 hours.

100%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

75%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

200%

7. 3 Tasks of 1.33 hours each.

52%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

82%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

70%

1. Please give your name, company and position inside the company:

Hagen, Stellenbosch University, Master's student

2. 1 Task of 8 hours

100%

3. 2 Tasks of 4 hours each.

101.5%

4. 5 Tasks, durations add up to 8 hours.

112%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

50%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

200%

7. 3 Tasks of 1.33 hours each.

52%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

82%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

70%

1. Please give your name, company and position inside the company:

Johan du Toit, Vanguard Software Solutions, CEO/CTO

2. 1 Task of 8 hours

95%

3. 2 Tasks of 4 hours each.

100%

4. 5 Tasks, durations add up to 8 hours.

100%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

75%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

200%

7. 3 Tasks of 1.33 hours each.

70%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

100%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

70%

1. Please give your name, company and position inside the company:

Francois Malherbe Bozza.mobi Junior Developer

2. 1 Task of 8 hours

95%

3. 2 Tasks of 4 hours each.

101.5%

4. 5 Tasks, durations add up to 8 hours.

112%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

75%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

400%

7. 3 Tasks of 1.33 hours each.

52%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

82%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

70%

1. Please give your name, company and position inside the company:

Johann Potgieter, self employed

2. 1 Task of 8 hours

105%

3. 2 Tasks of 4 hours each.

101.5%

4. 5 Tasks, durations add up to 8 hours.

112%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

75%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

200%

7. 3 Tasks of 1.33 hours each.

52%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

50%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

70%

1. Please give your name, company and position inside the company:

Ruzelle van Rooyen Student University of Stellenbosch

2. 1 Task of 8 hours

100%

3. 2 Tasks of 4 hours each.

100%

4. 5 Tasks, durations add up to 8 hours.

100%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

50%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

200%

7. 3 Tasks of 1.33 hours each.

52%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

50%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

70%

1. Please give your name, company and position inside the company:

Mareleen Smit, Dosent en Assistant Garderobe Meesteres by die Universiteit van Stellenbosch

2. 1 Task of 8 hours

100%

3. 2 Tasks of 4 hours each.

101.5%

4. 5 Tasks, durations add up to 8 hours.

112%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

50%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

200%

7. 3 Tasks of 1.33 hours each.

52%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

82%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

70%

1. Please give your name, company and position inside the company:

Philip Nell Aurecon Namibia Structural Engineer

2. 1 Task of 8 hours

95%

3. 2 Tasks of 4 hours each.

100%

4. 5 Tasks, durations add up to 8 hours.

112%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

100%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

100%

7. 3 Tasks of 1.33 hours each.

70%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

100%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

50%

1. Please give your name, company and position inside the company:

Emil Pragraj Sasol Polymers Senior Process Engineer

2. 1 Task of 8 hours

95%

3. 2 Tasks of 4 hours each.

100%

4. 5 Tasks, durations add up to 8 hours.

100%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

75%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

200%

7. 3 Tasks of 1.33 hours each.

25%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

82%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

70%

1. Please give your name, company and position inside the company:

Brinton Walker Sasol Engineer

2. 1 Task of 8 hours

105%

3. 2 Tasks of 4 hours each.

101.5%

4. 5 Tasks, durations add up to 8 hours.

100%

5. 1 Task of 8 hours, but person works twice as fast as everybody else.

50%

6. 1 Task of 8 hours, but person works twice as slow as everybody else.

200%

7. 3 Tasks of 1.33 hours each.

70%

8. 10 short tasks, each just 24 minutes long. (4 Hours)

82%

9. 1 Task of 4 hours, but is found that this individual consistently wastes 20% of his day on coffee breaks.

50%

APPENDIX B

SOURCE CODE: WORKLOADCALCULATIONS CLASS

```
package assign.components;

import gui.FilterSettings;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Calendar;
import java.util.Date;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;
import util.assign.TimestampFunctions;
import util.database.CloseResultSetConnection;
import util.database.RunQuery;
import aho.util.database.DataBaseInterface;
import assign.components.HR;
import assign.components.LocatorTasks;
import assign.components.Score;
import assign.components.ScoreList;
import assign.components.Task;
import assign.components.TaskOverhead;

public class WorkloadCalculations{

       private RateList ratelist;
       FilterSettings settings;
       RefinementFilterList refinementfilterlist;
       DataBaseInterface dbi;
       LocatorTasks locatortasks;
       ScoreList scores;
       HRUnderConsideration hrlist;
       TaskOverheadList taskoverheads;
       Date t = new Date();
       double T = 8.0;
       Set<Task> tasks;
       Map<Task, Double> booking_distribution;//hours for each task


       public WorkloadCalculations(DataBaseInterface dbi) throws SQLException
       {
              this.dbi = dbi;
              hrlist = new HRUnderConsideration();
              taskoverheads = new TaskOverheadList();
              ratelist = new RateList();
              String combinedqueryI = "SELECT HUMANRESOURCE.humanresource_pk,
HUMANRESOURCE.humanresource_description," +
                             " HUMANRESOURCE.is_standard, HUMANRESOURCE.efficiency_factor,
HUMANRESOURCE.idleness_factor," +
                             " RATE.rate_pk, RATE.rate_description,
RATE.cost_to_company_per_hour, RATE.internal_work_rate_per_hour," +
                             " RATE.billable_rate_per_hour, LOCATION.location_pk,
LOCATION.location_description, LOCATION.country," +
                             " LOCATION.region, LOCATION.city, LOCATION.longtitude,
LOCATION.latitude," +
                             " OFFICE.office_pk, OFFICE.office_description,
INTERNAL_PERSONNEL.humanresource_fk as \"hr\", INTERNAL_PERSONNEL.internal_personnel_pk,
INTERNAL_PERSONNEL.initials," +
                             " INTERNAL_PERSONNEL.surname, INTERNAL_PERSONNEL.employee_number,
INTERNAL_PERSONNEL.department_fk," +
                             " SKILL.skill_pk, SKILL.skill_description," +
                             " DEPARTMENT.department_pk, DEPARTMENT.department_description" +
                             " FROM HUMANRESOURCE, SKILLTAG, RATE, LOCATION, OFFICE,
INTERNAL_PERSONNEL, SKILL, DEPARTMENT" +
                             " WHERE "+
                             " INTERNAL_PERSONNEL.department_fk = DEPARTMENT.department_pk" +
                             " AND humanresource_pk = INTERNAL_PERSONNEL.humanresource_fk" +
                             " AND RATE.rate_pk = INTERNAL_PERSONNEL.rate_fk" +
                             " AND OFFICE.office_pk = INTERNAL_PERSONNEL.office_fk" +
                             " AND OFFICE.location_fk = LOCATION.location_pk" +
                             " AND humanresource_pk = SKILLTAG.humanresource_fk" +
                             " AND SKILL.skill_pk = SKILLTAG.skill_fk";

              String combinedqueryE = "SELECT HUMANRESOURCE.humanresource_pk,
HUMANRESOURCE.humanresource_description," +
                             " HUMANRESOURCE.is_standard, HUMANRESOURCE.efficiency_factor,
HUMANRESOURCE.idleness_factor," +
                             " RATE.rate_pk, RATE.rate_description,
RATE.cost_to_company_per_hour, RATE.internal_work_rate_per_hour," +
                             " RATE.billable_rate_per_hour, " +
                             " " +
                             " EXTERNAL_PERSONNEL.external_personnel_pk,
EXTERNAL_PERSONNEL.initials,  EXTERNAL_PERSONNEL.company," +
                             " EXTERNAL_PERSONNEL.surname, EXTERNAL_PERSONNEL.humanresource_fk
as \"hr\"," +
                             " SKILL.skill_pk, SKILL.skill_description," +
                             " DEPARTMENT.department_pk, DEPARTMENT.department_description" +
                             " FROM HUMANRESOURCE, SKILLTAG, RATE, EXTERNAL_PERSONNEL, SKILL,
DEPARTMENT, DEPARTMENT_TO_OFFICE" +
                             " WHERE "
                             +"department_pk = DEPARTMENT_TO_OFFICE.department_fk" +
                             " AND rate_pk = rate_fk"+
                             " AND humanresource_pk = SKILLTAG.humanresource_fk" +
                             " AND humanresource_pk = EXTERNAL_PERSONNEL.humanresource_fk" +

                             " AND SKILL.skill_pk = SKILLTAG.skill_fk";


              ResultSet queryresultI = new RunQuery(dbi, combinedqueryI).compute();
              ResultSet queryresultE = new RunQuery(dbi, combinedqueryE).compute();
              int c = 0;
              while(queryresultI.next()) {
                     if (queryresultI.getInt("humanresource_pk") == queryresultI.getInt("hr")
&& queryresultI.getInt("department_pk") == queryresultI.getInt("department_fk"))
                                    {

                                           c++;

                                           Rate rate = addRate(queryresultI);

                                           Location location = addLocation(queryresultI);

                                           Office office = addOffice(queryresultI, location);

                                           Department dept = addDepartment(queryresultI);

                                           Internal_Personnel ip =
addInternal_Personnel(queryresultI, rate, office, dept);
```

```
                        HR hrI = addHR(queryresultI, ip);

                    }

        }
        new CloseResultSetConnection(queryresultI);
        System.out.println("added all I");

        while(queryresultE.next()) {
            if (queryresultE.getInt("humanresource_pk") ==
queryresultE.getInt("hr"))
            {
                Rate rate = addRate(queryresultE);

                External_Personnel ep = addExternal_Personnel(queryresultE,
rate);

                HR hrE = addHR(queryresultE, ep);
            }

        }
        new CloseResultSetConnection(queryresultE);
        tasks = new HashSet<Task>();
        System.out.println("added all E");
        for (HR hr : hrlist)
            tasks.addAll(hr.getCurrentTasks());
        taskoverheads = findOverheads( tasks);
        for(HR hr : hrlist)
        {
            System.out.println("I AM AN HUMANRESOURCE: "+hr.getSimpleName());
            linkSkillsHR(hr);
        }


    }
    private Department addDepartment(ResultSet rs) throws SQLException {
        int pk = rs.getInt("department_pk");
        String desc = rs.getString("department_description");

        Department dept = new Department(pk, desc);
        return dept;
    }



    public void buildSuccession(Map<Task, Integer> map) throws SQLException
    {
        String q = "SELECT TASKFLOW.taskflow_pk, TASKFLOW.task_fk,
TASKFLOW.successor_task_fk FROM TASKFLOW";
        ResultSet rs = new RunQuery(dbi, q).compute();
        while(rs.next())
        {
            int taskflowpk = rs.getInt("taskflow_pk");
            int t_fk = rs.getInt("task_fk");
            int t_succ = rs.getInt("successor_task_fk");

            for (Task t1 : map.keySet())
                if (t1.getPk() == t_fk)
                {
                    for (Task t2 : map.keySet())
                        if (t2.getPk() == t_succ)
```

```
                                                {
                                                        t1.setSuccessor(t2);
                                                        t1.setFlowpk(taskflowpk);
                                                        break;
                                                }
                                        break;
                                }
                {
                new CloseResultSetConnection(rs);

        }

        public TaskOverheadList findOverheads(Set<Task> locatortasks) throws SQLException
        {
                TaskOverheadList tol  = new TaskOverheadList();
                String q = "select * from TASKOVERHEAD";
                ResultSet rs = new RunQuery(dbi, q).compute();
                while(rs.next())
                {
                        int task_fk = rs.getInt("task_fk");
                        int office_fk = rs.getInt("office_fk");
                        double overhead = rs.getDouble("overhead");
                        String office_desc = "";
                        int loc_fk = -1;;
                        Location loc = null;
                        ResultSet officers = new RunQuery(dbi, "select * from OFFICE where
office_pk = "+office_fk).compute();
                        while (officers.next())
                        {
                                office_desc = officers.getString("office_description");
                                loc_fk = officers.getInt("location_fk");
                                break;
                        }
                        new CloseResultSetConnection(officers);
                        ResultSet locationrs = new RunQuery(dbi,"select * from LOCATION where
location_pk = "+loc_fk).compute();
                        while (locationrs.next())
                        {
                                String description =
locationrs.getString("location_description");
                                String country = locationrs.getString("country");
                                String region = locationrs.getString("region");
                                String city = locationrs.getString("city");
                                double lon = locationrs.getDouble("longtitude");
                                double lat = locationrs.getDouble("latitude");


                                GeoCoordinates geo = new GeoCoordinates(lon, lat);
                                loc = new Location(loc_fk,description, country, region, city,
geo);

                                break;
                        }
                        new CloseResultSetConnection(locationrs);
                        Office office = new Office(office_fk, office_desc, loc);
                        for (Task task : locatortasks)
                        {
                                if (task.getPk() == task_fk)
                                {
                                        TaskOverhead e = new TaskOverhead(office, task, overhead);
                                        tol.add(e);
```

```
                    break;

                }
            }
        }
        new CloseResultSetConnection(rs);
        return taskoverheads;

    }


    private double compute(HR hr, Date today)
    {

            double T = hr. getT();
            double teff = hr.getTeff();

            booking_distribution = new HashMap<Task, Double>();

            for (Task task : hr.getCurrentTasks())
            {
                if (task.getDescription().equals("test"))
                {
                    System.out.println("start" +task.getEarlyStartDate());
                    System.out.println("end" +task.getEndDate());
                }
                System.out.println("size "+hr.getCurrentTasks().size());
                System.out.println("checking "+task.getDescription()+" on 
"+hr.getSimpleName());
                if (today.after(task.getEarlyStartDate()) && 
today.before(task.getEndDate()))
                {
                    System.out.println(hr.getSimpleName()+" has +1 task within 
timeframe. Task: "+task.getDescription());
                    booking_distribution.put(task, null);
                }
                else
                {
                    //System.out.println(task.getDescription()+" is not in 
timeframe");
                }
            }
            ifEndDatisTodayAndTaskDurationIsLessThanDay(hr, today);
            distributeTime(hr);

            int n = booking_distribution.size();
            double IF = hr.getIF();//idle factor
            double ans = 0.0;
            double f1 = 0.5;
            double f2 = 0.26;
            double f3 = 0.24;
            double maxStiffness = 0.0, workSum = 0.0;
            double increasingTasks, taskStiffness, workDuration, workdayLength;

            if (T == 0 )
            {
                workDuration = 0;
            }
            else
            {
```

```
            for (Task ti : booking_distribution.keySet())
            {
                double overhead = 0.0;
                for (TaskOverhead to : taskoverheads)
                {
                    if (hr.is_internal())
                    {
                        if (ti.getPk() == to.getTask().getPk() &&

hr.getInternal_Personnel().getOffice().getPk() == to.getOffice().getPk())
                        {
                            overhead = to.getOverhead();
                            break;
                        }
                    }


                }


                workSum += booking_distribution.get(ti)*(1 + overhead);
            }

            if (workSum == 0)
                workDuration = 0;
            else if (workSum >= T)
            {
                workDuration = Math.pow(0.959, T-workSum);
            }
            else
            {
                workDuration = Math.pow(0.841, T-workSum);
            }


        }

        if (T==0 || n==0)
        {
            workdayLength = 0;
        }
        else
        {
            workdayLength = Math.log(T+1)/Math.log(9.0);
        }

        if (n <= 1)
        {
            taskStiffness = 0;
        }
        else
        {
            for (Task ti : booking_distribution.keySet())
            {
                if (ti.getStiffness() > maxStiffness)
                    maxStiffness = ti.getStiffness();
            }
            taskStiffness = 0.000472*maxStiffness*Math.pow(n, 3.405);
        }
```

```
                        if (n == 0)
                                increasingTasks = 0;
                        else
                                increasingTasks = Math.pow(n, n/33.96);

                        ans = workDuration*f1 + workdayLength*f2 + increasingTasks*f3 +
taskStiffness;
                        ans *= teff;


                        ans += IF;

                        return ans;

        }

        public double getAverageWorkload(HR hr, Date start, Date end)
        {


                Calendar c = Calendar.getInstance();
                c.setTime(start);
                double work = 0;
                double count = 0;
                while(c.getTime().before(end))
                {
                        Date today = c.getTime();
                        work += compute(hr, today);
                        count++;

                        if (c.DAY_OF_MONTH < c.getActualMaximum(c.DAY_OF_MONTH))
                        {
                                c.add(Calendar.DAY_OF_MONTH, 1);
                        }


                }
                return work/count;

        }
        public double getAverageWorkload(HR hr, ScoreList s)
        {

                Calendar earliest = Calendar.getInstance();
                Calendar latest = Calendar.getInstance();
                earliest.setTime(new Date(0));
                latest.setTime(new Date(999999999));
                for (Score score : s)
                {

                        if (earliest.after(score.getTask().getEarlyStartDate()))
                                earliest.setTime(score.getTask().getEarlyStartDate());
                        if (latest.before(score.getTask().getEndDate()))
                                latest.setTime(score.getTask().getEndDate());
                }
                return getAverageWorkload(hr, earliest.getTime(), latest.getTime());
        }
```

```
    public void ifEndDatisTodayAndTaskDurationIsLessThanDay(HR hr, Date date)
    {
        double T = hr.getT();
        for (Task t : booking_distribution.keySet())
        {
            if (date.~~getYear~~() == t.getEarlyStartDate().~~getYear~~() && date.~~getMonth~~()
== t.getEarlyStartDate().~~getMonth~~() && date.~~getDay~~() == t.getEarlyStartDate().~~getDay~~())
                if (date.~~getYear~~() == t.getEndDate().~~getYear~~() && date.~~getMonth~~()
== t.getEndDate().~~getMonth~~() && date.~~getDay~~() == t.getEndDate().~~getDay~~())
                {
                    if (t.getDuration() <= hr.getT())
                    {
                        booking_distribution.put(t, t.getDuration());
                        T = T-t.getDuration();
                    }
                    else
                    {
                        booking_distribution.put(t, hr.getT());
                        T = 0;
                    }
                }
        }

    }

    public void distributeTime(HR hr)
    {
        double total = 0;
        double base;
        double T = hr.getT();

        for (Task t : booking_distribution.keySet())
        {
            if (booking_distribution.get(t) != null)
                T = T-booking_distribution.get(t);
        }

        for (Task t : booking_distribution.keySet())
        {
            total += 1.0/t.getPriority();
        }
        base = T/total;
        for (Task t : booking_distribution.keySet())
        {
            if (booking_distribution.get(t) == null)
                booking_distribution.put(t, base/t.getPriority());
        }
    }


    private HR addHR(ResultSet rs, Internal_Personnel ip) throws SQLException
    {
        int hr_key = rs.getInt("humanresource_pk");
        String description = rs.getString("humanresource_description");
        double teff = rs.getDouble("efficiency_factor");
        double IF = rs.getDouble("idleness_factor");
        Set<Task> ct = findCurrentTasks(hr_key);
```

```
                HR hr = new HR(hr_key, description, ip, teff, IF, T, ct);
                if (!rs.getBoolean("is_standard"))
                {
                        changeToNonStandard(rs, hr);
                }

                hrlist.add(hr);

                return hr;
        }

        private HR addHR(ResultSet rs, External_Personnel ep) throws SQLException
        {

                int hr_key = rs.getInt("humanresource_pk");
                String description = rs.getString("humanresource_description");

                double IF = rs.getDouble("idleness_factor");
                Set<Task> ct = findCurrentTasks(hr_key);
                HR hr = new HR(hr_key, description, ep, T, IF,  ct);
                if (!rs.getBoolean("is_standard"))
                {
                        changeToNonStandard(rs, hr);
                }

                hrlist.add(hr);

                return hr;
        }

        private void changeToNonStandard(ResultSet rsHr, HR hr) throws SQLException {
                hr.setIs_standard(rsHr.getBoolean("is_standard"));
                String s = "SELECT NONSTANDARDHUMANRESOURCE.* FROM NONSTANDARDHUMANRESOURCE
WHERE NONSTANDARDHUMANRESOURCE.humanresource_fk ="+hr.getPk();
                ResultSet rs  = new RunQuery(dbi, s).compute();
                while(rs.next())
                {
                hr.setNonStandardkey(rs.getInt("nonstandardhumanresource_pk"));

                hr.setMonday_hours(rs.getDouble("monday_hours"));
                hr.setTuesday_hours(rs.getDouble("tuesday_hours"));
                hr.setWednesday_hours(rs.getDouble("wednesday_hours"));
                hr.setThursday_hours(rs.getDouble("thursday_hours"));
                hr.setFriday_hours(rs.getDouble("friday_hours"));
                hr.setSaterday_hours(rs.getDouble("saterday_hours"));
                hr.setSunday_hours(rs.getDouble("sunday_hours"));
                break;
                }
                new CloseResultSetConnection(rs);

        }


        private Rate addRate(ResultSet rs) throws SQLException
        {
                int pk = rs.getInt("rate_pk");
                String description = rs.getString("rate_description");
                double ctc = rs.getDouble("cost_to_company_per_hour");
                double internal = rs.getDouble("internal_work_rate_per_hour");
                double external = rs.getDouble("billable_rate_per_hour");
                Rate r = new Rate(pk, description, ctc, internal, external);
                ratelist.add(r);
                return r;


        }

        private Location addLocation(ResultSet rs) throws SQLException
        {
                String description = rs.getString("location_description");
                String country = rs.getString("country");
                String region = rs.getString("region");
                String city = rs.getString("city");
                double lon = rs.getDouble("longtitude");
                double lat = rs.getDouble("latitude");
                int location_key = rs.getInt("location_pk");

                GeoCoordinates geo = new GeoCoordinates(lon, lat);
                Location loc = new Location(location_key,description, country, region, city, 
geo);

                return loc;
        }

        private Office addOffice(ResultSet rs, Location loc) throws SQLException
        {
                int k = rs.getInt("office_pk");
                String description = rs.getString("office_description");
                Office o = new Office(k, description, loc);
                return o;
        }

        //call this after everything
        private void linkSkillsHR(HR hr) throws SQLException
        {
                ResultSet rs  = new RunQuery(dbi, "select * from SKILL, SKILLTAG where skill_pk 
= skill_fk and humanresource_fk = "+hr.getPk()).compute();

                while (rs.next())
                {
                        int hr_key = hr.getPk();
                        String description = rs.getString("skill_description");
                        int p_key = rs.getInt("skill_pk");
                        Skill p = new Skill(p_key, description);

                        hrlist.getByKey(hr_key).addSkill(p);
                        System.out.println("linking skill: "+description+" to hr: "+hr_key);

                }
                new CloseResultSetConnection(rs);

        }

        private Task linkSkillsTask(Task task) throws SQLException
        {

                int task_fk = task.getPk();
                SkillList skills = new SkillList();
```

```
               ResultSet rs = new RunQuery(dbi, "select skill_pk, skill_description from
SKILL_TO_TASK, SKILL where " +
                             "skill_pk = skill_fk and task_fk = "+task_fk).compute();
               while(rs.next())
               {
                      skills.add(new Skill(rs.getInt("skill_pk"),
rs.getString("skill_description")));
               }
               new CloseResultSetConnection(rs);
               task.setSkills(skills);
               return task;

       }

       private Internal_Personnel addInternal_Personnel(ResultSet rs, Rate r, Office o,
Department dept) throws SQLException
       {
               int pk = rs.getInt("internal_personnel_pk");
               int emp = rs.getInt("employee_number");
               String initials = rs.getString("initials");
               String surname = rs.getString("surname");
               int hr = rs.getInt("humanresource_pk");


               Internal_Personnel ip = new Internal_Personnel(pk, emp, initials, surname, o,
dept, r, hr);
               return ip;
       }
       private External_Personnel addExternal_Personnel(ResultSet rs, Rate r) throws
SQLException
       {
               int pk = rs.getInt("external_personnel_pk");
               String company = rs.getString("company");
               String initials = rs.getString("initials");
               String surname = rs.getString("surname");
               int hr = rs.getInt("humanresource_pk");


               External_Personnel ep = new External_Personnel(pk, company, initials, surname,
r, hr);
               return ep;
       }

       public Set<Task> findCurrentTasks(int hr_key) throws SQLException
       {
               String q = "select * from TASK, PROJECT, WORKPACKAGE, BOOKING, LOCATION where
BOOKING.task_fk = task_pk and workpackage_pk = workpackage_fk and project_pk = project_fk and
location_fk = location_pk and humanresource_fk = "+hr_key;
               ResultSet rs = new RunQuery(dbi, q).compute();
               HashMap<Task, Integer> map = new HashMap<Task, Integer>();
               while(rs.next())
               {

                      int t_pk = rs.getInt("task_pk");
                      System.out.println(+hr_key+" has task "+t_pk);
                      String t_description = rs.getString("task_description");

                      Date t_earlystart =
TimestampFunctions.timestampToDate(rs.getTimestamp("earliest_start"));
```

```
                      Date t_lateend = 
TimestampFunctions.timestampToDate(rs.getTimestamp("latest_end"));
                      Date t_actualstart = 
TimestampFunctions.timestampToDate(rs.getTimestamp("actual_start"));
                      Boolean t_stretch = rs.getBoolean("stretchable");
                      Double t_stiff = rs.getDouble("stiffness");
                      int t_priority = rs.getInt("priority");
                      double t_duration = rs.getDouble("duration");
                      int work_pk = rs.getInt("workpackage_fk");
                      String work_description = rs.getString("workpackage_description");
                      int proj_pk = rs.getInt("project_fk");
                      String proj_description = rs.getString("project_description");
                      Double proj_version = rs.getDouble("version_number");
                      int loc_pk = rs.getInt("location_fk");
                      String loc_description = rs.getString("location_description");
                      String loc_country = rs.getString("country");
                      String loc_region = rs.getString("region");
                      String loc_city = rs.getString("city");
                      Double loc_longtitude = rs.getDouble("longtitude");
                      Double loc_latitude = rs.getDouble("latitude");

                      Location location = new Location(loc_pk, loc_description, loc_country, 
loc_region, loc_city, loc_longtitude, loc_latitude);
                      Project project = new Project(proj_pk, proj_description, location, 
proj_version);
                      WorkPackage workpackage = new WorkPackage(work_pk, work_description);

                      Task task = new Task(t_pk, t_description, t_earlystart, t_lateend, 
t_actualstart, t_duration, null, project, workpackage, t_stretch, t_stiff, null, t_priority);
                      linkSkillsTask(task);
                      map.put(task, null);

              }
              new CloseResultSetConnection(rs);
              buildSuccession(map);
              return map.keySet();
       }

       public HashMap<HR, Double> AllHRtoAverage(Calendar timeframeStart, Calendar 
timeframeEnd)
       {
              HashMap<HR, Double> map = new HashMap();
              for (HR hr : hrlist)
              {
                      map.put(hr, getAverageWorkload(hr, timeframeStart.getTime(), 
timeframeEnd.getTime()));
              }
              return map;
       }

       public HashMap<HR, Double> AllHRtoSpecificDay(Date day)
       {
              HashMap<HR, Double> map = new HashMap();
              for (HR hr : hrlist)
              {
                      map.put(hr, compute(hr, day));
              }
              return map;
       }
}
```