Introduction

The goal of the Timesheet application is to facilitate the entry, submission, and review of an employee’s weekly time worked at Retrans. It is to be accomplished by creating a web application where all employees can login and manipulate their weekly timesheets. The application will be profile-driven, meaning that users will have different privileges based on the information entered into their profiles. The system is being created to replace the pen and paper entry currently used at Retrans. By digitizing the timesheets, the company will be much more efficient with regards to auditing the time worked by employees. The Timesheet application will be solely focused on the archival of hours worked by employees, no detailed analysis of an employee or department will be done; however, to provide a convenient process for detailed analysis, the application will allow some users to export the timesheet data to a .csv file for use in Microsoft Excel or other spreadsheet programs. The success criteria are:

* A web-based application
* Robust login authentication system
* Variable methods of hour entry – daily or bulk, depending on the type of employee
* Variable types of hours, i.e. sick, vacation, holiday
* Review of submitted timesheets
* Configurable profiles for employees to provide a hierarchy of privileges
* Ability to group employees into departments
* Exportation of timesheets to a .csv file for detailed analysis and payroll operations
* Notifications to alert employees of issues regarding to the timesheet
* On-the-fly creation of global holidays in rare circumstances

The meaning of any acronyms found in this document can be found here.

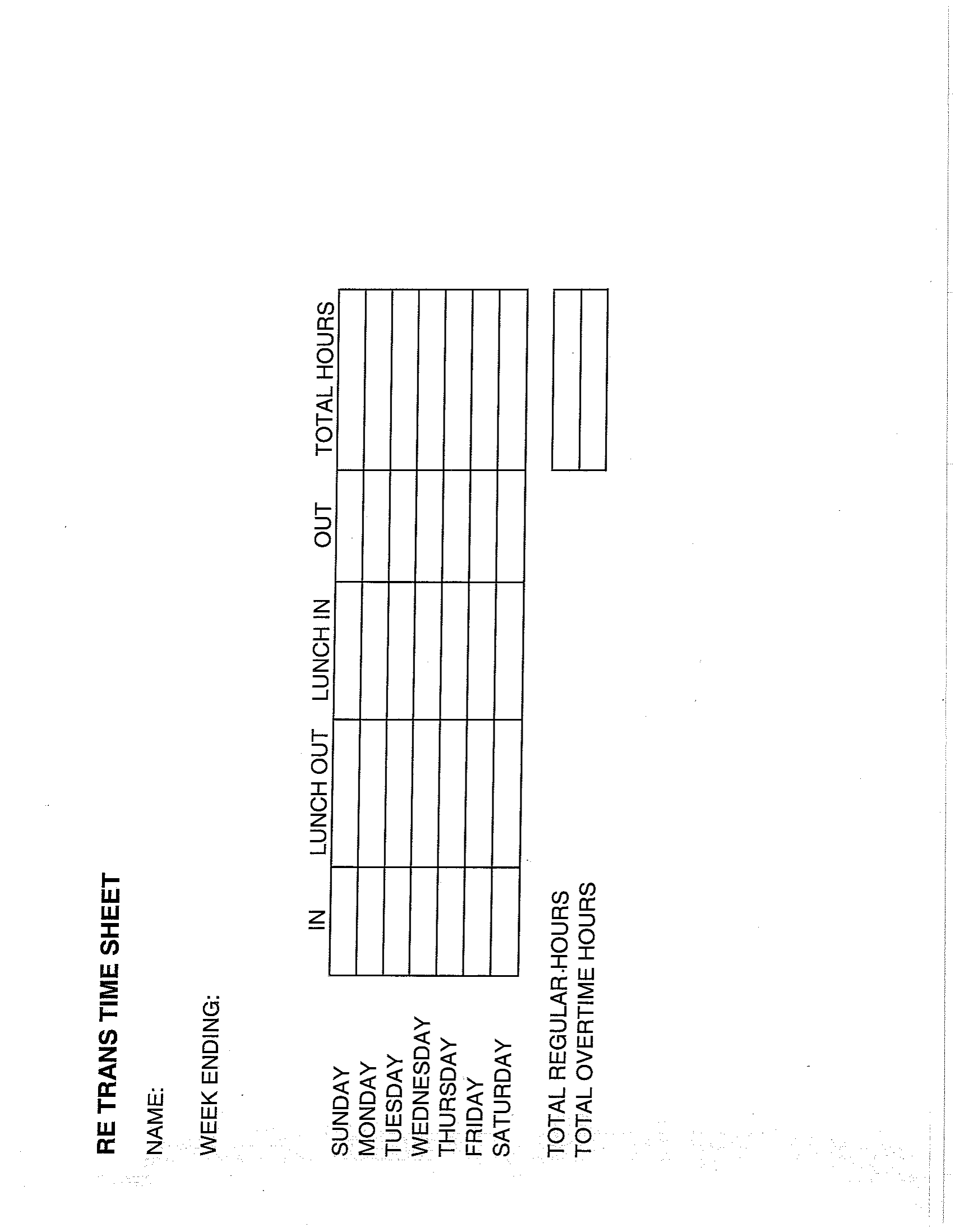
TS – timesheet

CFO – Chief Financial Officer

WBS – work breakdown structure

Current System

The current system for tracking employee times is paper-based. Employees fill out a timesheet and have their supervisor sign it and submit it to the CFO. A sample time sheet is provided. It has four sections per day labeled IN, LUNCH OUT, LUNCH IN, and OUT. TOTAL HOURS are calculated per day and assessed at the end of the week. The current system is fraught with problems. A single missing time sheet holds up the entire process, and the CFO must manually track down the offender. There is also unnecessary redundancy in this process, since the pen and paper information is transmitted to a computer. A better system would be one that is entirely digital, with a digital submission process and a digital approval process. This would allow the CFO to spend less time managing time sheets and more time on his more pressing responsibilities.



Proposed System and Functional Requirements

The proposed system is a profile-driven, web-based application that streamlines the entry, submission, and manipulation of timesheets for Retrans. For the system representation, there are five types of employees: Grunts, Grunt+s, Elites, Prophets, and CFO. Grunts log hours by clocking in and out in real time while at work. Sick and vacation time do not apply to this class of workers. Grunt+s log hours by filling out a timesheet table daily or in bulk at the end of the week. Also, Grunt+s can request sick and vacation time. Elites review, submit, deny, or edit timesheets based on what the employees enter. In addition to timesheet manipulation, the Elites can request or deny sick and vacation time. The Elite of each department sends the timesheets to the CFO, who usually exports the data to Excel or a payroll application. Furthermore, the CFO can request an audit of an employee’s time sheet actions to validate the accuracy of a timesheet. Finally, the Prophets are system administrators who maintain the application and have nearly universal access rights to facilitate repairs and updates.

Nonfunctional Requirements

**Usability**

-Does not require expertise to use. It is simple to use and actions are clearly labeled.

-Starts with a basic username and password login. Then moves to a simple time-in/time-out screen. Higher level users have a more advanced screen that would take some training to use.

-User would be provided with some documentation for how to navigate the system

**Reliability**

-The systems must be reliable and robust. A glitch in the system would mess up how someone is paid and render the system useless.

-It would only be acceptable to reset the system if no data is lost in the process.

-The system cannot afford to lose any data.

-A high enough ranking user should be able to directly modify the times in the event of an exception.

-There are no safety requirements of the system

-System should only be accessed by authorized users (employees) and user profiles should only be accessed by the appropriate person.

**Performance**

-System should respond immediately to the user

-System should process user actions immediately

-System should be able to handle the high amount of users clocking in at the same time every morning

-*How large is a typical data store for comparable systems?*

-Users should not experience latency. Users should be guaranteed that their hours are being logged immediately.

**Supportability**

-System may eventually support alternate time entry devices (i.e. Card swipe)

-The payroll department maintains the system

**Implementation**

-System should be able to run on all company computers

-*Are constraints imposed by the maintenance team?*

*-Are constraints imposed by the testing team?*

**Interface**

-System should run on existing company computers

-Data should be stored as a .csv or similar file type stored on a central server.

-*What standards in use by the client should be supported by the system*

**Operation**

-Payroll manages the system

**Packaging**

-IT would install the system

-There would be one installation per computer

-Installation should happen ASAP since it is required for employees

**Legal**

-System would be licensed property of the company

-*Are any liability issues associated with system failure*

*-Are any royalties or licensing fees incurred by using specific algorithms or components*

Use Case Model

|  |  |  |
| --- | --- | --- |
|  |  |  |
| *Use case name* |  | ClockIn/Out |
|  |  |  |
| *Participating actors* |  | Initiated by Grunt |
|  |  |  |
| *Flow of events* |  | 1. The Grunt activates the "Clock In/Out function of his/her terminal. |
|  |  | 2. Program responds by stamping the time on the Grunt's "card." |
|  |  |  |
| *Entry condition* |  | The Grunt is logged into TimeSheet |
|  |  |  |
| *Exit condition* |  | The Grunt has received acknowledgment of clock in/out success via "greyed out" interface. |
|  |  |  |
| *Quality requirements* |  | The interface remains greyed for one minute to prevent the Grunt from accidentally clocking in/out twice. |
|  |  |  |
|  |  |  |
| *Use case name* |  | RequestSick/Vaca |
|  |  |  |
| *Participating actors* |  | Initiated by Grunt+ |
|  |  | Communicates with Elite |
|  |  |  |
| *Flow of events* |  | 1. The Grunt+ activates the "Request Time Off" function of his/her terminal. |
|  |  | 2. The Grunt+ fills out and submits request form. |
|  |  | 3. Program sends alert and text file containing the Grunt+'s request information to the Elite. |
|  |  |  |
| *Entry condition* |  | The Grunt+ is logged into TimeSheet |
|  |  |  |
| *Exit condition* |  | The Elite has been sent notification of Grunt+'s RTO |
|  |  |  |
| *Quality requirements* |  |  |
|  |  |  |
|  |  |  |
| *Use case name* |  | ReviewTimesheets |
|  |  |  |
| *Participating actors* |  | Initiated by Elite |
|  |  |  |
| *Flow of events* |  | 1. After the Elite is logged in, a list of Grunt/+s under his department appears on his screen. |
|  |  | 2. The Elite clicks on each Grunt/+'s name to have his or her timesheet data displayed for review. |
|  |  | 3. The Elite reviews the hours and comments from the Grunt/+s, determing if there are any problems with the hours. |
|  |  |  |
| *Entry condition* |  | The Elite is logged into TimeSheet |
|  |  |  |
| *Exit condition* |  | The Elite has reviewed the Grunt/+'s timesheet and is ready to approve or deny the hours. |
|  |  |  |
| *Quality Requirements* |  |  |
|  |  |  |
|  |  |  |
| *Use case name* |  | ApproveTimesheets |
|  |  |  |
| *Participating actors* |  | Initiated by Elite |
|  |  |  |
| *Flow of events* |  | 1. See ReviewTimesheets. At the end of ReviewTimesheets, the Elite is ready to make a decision regarding the Grunt/+'s hours. |
|  |  | 2. The Elite determines that there are no problems with the hours recorded and clicks the "Approve Hours" button. |
|  |  |  |
| *Entry condition* |  | The Elite has reviewed the Grunt/+'s timesheet info. |
|  |  |  |
| *Exit condition* |  | The Elite has approved the Grunt/+'s hours. |
|  |  |  |
| *Quality requirements* |  |  |
|  |  |  |
|  |  |  |
| *Use case name* |  | DenyTimesheets |
|  |  |  |
| *Participating actors* |  | Initiated by Elite |
|  |  |  |
| *Flow of events* |  | 1. See ReviewTimesheets. At the end of ReviewTimesheets, the Elite is ready to make a decision regarding the Grunt/+'s hours. |
|  |  | 2. The Elite determines that there are issues with the hours recorded by the Grunt/+ and hits the "Deny Hours" button. |
|  |  | 3. A notification is sent to the Grunt/+ to inform him or her that the hours have been denied. The Grunt/+ must discuss the issue with the Elite. |
|  |  |  |
| *Entry condition* |  | The Elite is logged into Timesheet |
|  |  |  |
| *Exit condition* |  | A notification has been sent to the Grunt/+ indicating that there is an issue with his timesheet info. |
|  |  |  |
| *Quality requirements* |  |  |
|  |  |  |
|  |  |  |
| *Use case name* |  | ApproveVacaSick |
|  |  |  |
| *Participating actors* |  | Elite recieves request from Grunt+ |
|  |  | Grunt+ receives notification from Elite |
|  |  |  |
| *Flow of events* |  | 1. Elite reviews text file received from Grunt+. |
|  |  | 2. Elite manually adjusts Grunt+'s Timesheet. |
|  |  | 3. Program sends notification to Grunt+. |
|  |  |  |
| *Entry condition* |  | Grunt+ requests vacation or sick time. |
|  |  |  |
| *Exit condition* |  | Notification of Elite's actions sent to Grunt+. |
|  |  |  |
| *Quality requirements* |  |  |
|  |  |  |
|  |  |  |
| *Use case name* |  | DoesntClock |
|  |  |  |
| *Participating actors* |  | Negligent Grunt |
|  |  | Initiated by Elite |
|  |  |  |
| *Flow of events* |  | 1. See ReviewTimesheets. At the end of ReviewTimesheets, the Elite is ready to make a decision regarding the Grunt's hours. |
|  |  | 2. Elite manually adjusts Grunt's Timesheet. |
|  |  | 3. Program sends notification to Grunt. |
|  |  |  |
| *Entry condition* |  | Elite is logged into Timesheet. |
|  |  |  |
| *Exit condition* |  | Notification of Elite's actions sent to Grunt. |
|  |  |  |
| *Quality requirements* |  |  |
|  |  |  |
|  |  |  |
| *Use case name* |  | SendNotification |
|  |  |  |
| *Participating actors* |  | Initiated by Elite |
|  |  | Received by Grunt |
|  |  |  |
| *Flow of events* |  | 1. See DoesntClock or ApproveVacaSick. Elite has manually adjusted the Grunt's timesheet. |
|  |  | 2. Notification is sent to Grunt. |
|  |  |  |
| *Entry condition* |  | Elite manually adjusts time sheet. |
|  |  |  |
| *Exit condition* |  | Grunt receives notification. |
|  |  |  |
| *Quality requirements* |  |  |
|  |  |  |
|  |  |  |
| *Use Case name* |  | Modify User |
|  |  |  |
| *Participating actors* |  | Prophet, CFO, Grunt or Grunt+ |
|  |  |  |
| *Flow of Events* |  | 1. Prophet or CFO is notified of change of User's information |
|  |  | 2. The Prophet or CFO logs into Timesheet. |
|  |  | 3. The Prophet or CFO clicks on the Grunt's name and clicks the Edit button. |
|  |  | 4. The Prophet or CFO is presented with an Edit Profile form where they make the changes. |
|  |  | 5. The Prophet or CFO clicks the "Save Changes" button. |
|  |  | 6. The Grunt is sent a notification that changes were made to his or her profile. |
|  |  |  |
| *Entry Condition* |  | User's personal information changes |
|  |  |  |
| *Exit Condition* |  | Changes are made to the User's profile |
|  |  |  |
| *Quality Requirements* |  |  |
|  |  |  |
|  |  |  |
| *Use Case Name* |  | Add User |
|  |  |  |
| *Participating Actors* |  | Elite, Grunt/+ |
|  |  |  |
| *Flow of Events* |  | 1. The Grunt/+ supplies HR with their information |
|  |  | 2. The Elite logs into Timesheet. |
|  |  | 3. The Elite clicks the "Add New Member" button. |
|  |  | 4. The Elite fills out the profile information for the new Grunt/+. |
|  |  | 5. The Elite clicks the "Save New Member" button. |
|  |  | 6. The Grunt/+ is notified that their profile has been created, and they are supplied with a username/password to log into the system. |
|  |  |  |
| *Entry Condition* |  | User supplies personal information |
|  |  |  |
| *Exit Condition* |  | Elite adds the profile to the system and Grunt/+ is given username/password. |
|  |  |  |
| *Quality Requirements* |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| *Use Case Name* |  | Remove User |
|  |  |  |
| *Participating Actors* |  | Elite |
|  |  |  |
| *Flow of Events* |  | 1. The Elite is notified of the Grunt/+'s departure from the company. |
|  |  | 2. The Elite logs into Timesheet. |
|  |  | 3. The Elite clicks on the Grunt/+'s name and clicks the "Delete Member" button. |
|  |  |  |
| *Entry Condition* |  | Grunt/+ leaves the company |
|  |  |  |
| *Exit Condition* |  | Elite removes Grunt/+'s profile from the system |
|  |  |  |
| *Quality Requirements* |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| *Use Case Name* |  | Edit Password |
|  |  |  |
| *Participating Actors* |  | Prophet, Grunt/+ |
|  |  |  |
| *Flow of Events* |  | 1. Prophet is notified of changes to Grunt/+'s information |
|  |  | 2. Prophet logs into Timesheet. |
|  |  | 3. Prophet clicks on the Grunt/+'s name and clicks the "Edit" button. |
|  |  | 4. Prophet edits the Grunt/+'s password. |
|  |  | 5. The Grunt/+ is sent a notification of the password change and what it has become. |
|  |  |  |
| *Entry Condition* |  | Grunt/+'s personal information changes |
|  |  |  |
| *Exit Condition* |  | Grunt/+'s profile is updated with new password that is sent to him/her. |
|  |  |  |
| *Quality Requirements* |  |  |
|  |  |  |
|  |  |  |
| *Use Case Name* |  | Issue Global Hours |
|  |  |  |
| *Participating Actors* |  | Initiated by CFO |
|  |  | Received by all |
|  |  |  |
| *Flow of Events* |  | 1. CFO logs into Timesheet. |
|  |  | 2. CFO clicks the "Create Holiday" button. |
|  |  | 3. CFO fills out the "Holiday" form with the date and hours that the employees are to receive. |
|  |  | 4. CFO clicks the "Save Holiday" button. |
|  |  |  |
| *Entry Condition* |  | CFO announces paid day of leave. |
|  |  |  |
| *Exit Condition* |  | Employees take day off and receive pay. |
|  |  |  |
| *Quality Requirements* |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| *Use Case Name* |  | Export Data |
|  |  |  |
| *Participating Actors* |  | Initiated by CFO |
|  |  |  |
| *Flow of Events* |  | 1. CFO logs into Timesheet. |
|  |  | 2. CFO notices that all Grunt/+ hours have been reported for the current week. |
|  |  | 3. CFO clicks the "Export to Excel" button at the bottom of his page. |
|  |  |  |
| *Entry Condition* |  | CFO desires to assess hourly information in more detail. |
|  |  |  |
| *Exit Condition* |  | CFO copies data into a spreadsheet. |
|  |  |  |
| *Quality Requirements* |  |  |
|  |  |  |
|  |  |  |
| *Use Case Name* |  | AdministerHolidayPay |
|  |  |  |
| *Participating Actors* |  | Initiated by CFO |
|  |  |  |
| *Flow of Events* |  | 1. CFO logs into Timesheet. |
|  |  | 2. CFO reviews each employee's Timesheet information to determine if they should receive Holiday pay. |
|  |  | 3. CFO clicks "Approve Holiday Pay" button for each Grunt+. |
|  |  |  |
| *Entry Condition* |  | CFO reviews employee time sheets |
|  |  |  |
| *Exit Condition* |  | CFO gives holiday pay |
|  |  |  |
| *Quality Requirements* |  |  |
|  |  |  |
|  |  |  |
| *Use Case Name* |  | Log Hours (Grunt+) |
|  |  |  |
| *Participating Actors* |  | Initiated by Grunt+ |
|  |  |  |
| *Flow of Events* |  | 1. Grunt+ logs into Timesheet. |
|  |  | 2. Grunt+ is presented with a weekly Timesheet view where hours can be input. |
|  |  | 3. Grunt+ enters TimeIn, LunchOut, LunchIn, TimeOut, and Comments. |
|  |  |  |
| *Entry Condition* |  | Grunt+ needs to input hours. |
|  |  |  |
| *Exit Condition* |  | If all hours have been input, then they will be available to be viewed by the appropriate Elite based on department. |
|  |  |  |
| *Quality Requirements* |  |  |
|  |  |  |
|  |  |  |
| *Use Case Name* |  | Access Audit |
|  |  |  |
| *Participating Actors* |  | Initiated by CFO or Elite |
|  |  |  |
| *Flow of Events* |  | 1. CFO or Elite logs into Timesheet. |
|  |  | 2. CFO or Elite clicks on the "Auditing Info" button. |
|  |  | 3. CFO is presented with the auditing information for all employees of the system, while an Elite is presented with those pertaining to his dept. |
|  |  | 4. CFO or Elite checks the auditing information against the information on the timesheet. |
|  |  |  |
| *Entry Condition* |  | The CFO or Elite would like to validate timesheet information. |
|  |  |  |
| *Exit Condition* |  | The CFO or Elite has been given the time of events that he wished to validate. |
|  |  |  |
| *Quality Requirements* |  | The auditing system must give an accurate account of the events occurring within the system. |
|  |  |  |

Glossary

**accuracy** – a quantitative measure of the magnitude of error

**add user** – initiated by an Elite, a new employee is entered into the system and assigned a username, password, and employee classification.

**analysis** – an activity during which developers ensure that the system requirements are *correct*, *complete*, *consistent*, and unambiguous. Analysis produces the *analysis model*.

**analysis model** – a model of the system that aims to be *correct*, *complete*, *consistent*, and unambiguous. The analysis model consists of the *functional model*, *the analysis object model*, and the *dynamic model*.

**analysis object model** – the object produced during analysis. The analysis object model describes the application domain concepts that the system manipulates and the user-visible interfaces of the system.

**approve timesheet** – timesheet is reviewed and approved by an Elite

**associations** – a relationship between two or more classes denoting the possible links between instances of the classes. An association has a name and can have multiplicity and role information attached to each of its ends.

**attributes** – a named property of a class defining a range of the values an object can contain.

**availability** – the degree to which a system or component is operational and accessible when required for use.

**Chief Financial Officer (CFO)** – an employee who ultimately receives all of the timesheets for analysis and payment of the employees.

**completeness** – the property of a model describing whether or not all relevant phenomena are modeled. A model is incomplete if one or more relevant phenomena do not have a corresponding concept in the model.

**consistency** – the property of a model that indicates whether or not it contradicts itself. A model is inconsistent if it provides several incompatible views of the system.

**constraint** – a rule attached to a UML element restricting its semantics.

Constraints can be depicted by a note containing natural language text or an expression in a formal language (e.g., OCL).

**contracts** – a set of *constraints* on a class or component allowing the caller and callee to share the same assumptions about the class or component.

**correctness** – the property of a model that indicates if it accurately represents the system that the client needs and that the developers intend to build.

**current system** – current state of the proposed system.

**deny timesheet** – timesheet is reviewed and denied by an Elite. A notification is then sent to the employee stating that he/she needs to meet with his/her supervisor to fix the discrepancies.

**dependability** – the property of a computer system that describes how reliably it performs.

**design goals** – a quality that the system should optimize. Design goals are often inferred from *non-functional requirements* and are used to guide design decisions.

**dynamic model** – describes the components of the system that have interesting behavior.

**Elite** – a salaried employee (usually a manager of Grunt/+) who approves or denies the timesheets of the employees in his department.

**entry condition** – a condition that needs to be satisfied before a use case is initiated.

**exit condition** – a condition that is satisfied after the completion of a use case.

**export** – all employee hourly info will be exported into a Microsoft Excel file. This action is only available to the CFO.

**functional model** – describes the functionality of the system from the user’s point of view.

**functional requirements** – an area of functionality the system must support. The functional requirements describe the interactions between the actors and the system independent of the realization of the system.

**global hours** -- hours administered and approved by a CFO that are automatically given to the employee. These hours override any entered by an employee during that given time slow.

**Grunt** – an hourly employee who simply clocks in and out in real time when at work. Sick and vacation days do not apply to this type of employee. His hours are logged by clicking a button when he logs into the timesheet application at work.

**Grunt+** - an hourly employee who must submit a timesheet at the end of every week. This type of employee logs his hours (time in, lunch out, lunch in, time out) for every day of the week by entering it on the timesheet page after he logs into the system. Sick and vacation time apply for this type of employee.

**implementation** – an activity during which developers translate the object model into code.

**interface** – a point of interaction between components

**legal** – Constraint concerned with licensing, regulation, and certification issues.

**log hours** – available to Grunt+, the employee will enter his time in, time out, lunch in, and lunch out into a table. This table will then be submitted as the employee’s timesheet.

**modify user** – initiated by and Elite, an employees information can be adjusted in the event of change. Elite can change all info except for an employees password, which can only be changed by the CFO.

**non-functional requirements** – a user-visible *constraint*on the system. Nonfunctional requirements describe user-visible aspects of the system that are not directly related with the functionality of the system.

**object design model** – a detailed model representing the application and solution objects that make up the system. The object design model includes detailed class specifications, *contracts*, *types*, *signatures*, and *visibilities* for public operations.

**object model** – describes the structure of a system in terms of *objects*, *attributes, associations*, and *operations*. During *requirements* and *analysis*, the object model starts as the *analysis object model* and describes the application concepts relevant to the system. During system design, the object model is refined into the *system design object model* and includes descriptions of the subsystem interfaces. During object design, the object model is refined into the *object design model* and includes detailed descriptions of solution objects.

**objects** – an instance of a class. An object has an identity and stores attribute values.

**operations** – an atomic piece of behavior that is provided by a class. A calling object triggers the execution of an operation by sending a message to the object on which the operation should be executed.

**packaging** – constraint on the actual delivery of the system.

**performance** – any quantifiable attribute of a system, such as *response time*, *throughput*, *availability*, and *accuracy*.

**Prophet** – a system administrator of the system. A Prophet has almost universal access rights and can change many properties of the system.

**reliability** – ability of a system or component to perform its required functions under stated conditions for a specified period of time. Reliability requirements include, for example, an acceptable mean time to failure, the ability to detect specified faults, or to withstand specified security attacks. Reliability includes *dependability*, *robustness*, and *safety*.

**remove user** - initiated by an Elite, an employee is taken out of the system and prevented from entering.

**requirements** – a function that the system must have. or a user-viable constraint on the system.

**response time** – an attribute of the system denoting how quickly it can react to a user input.

**robustness** – The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environment conditions.

**safety** – a measure of the absence of catastrophic consequences to the environment

**scenario** – instance of a use case. A scenario represents a concrete sequence of interaction between one or more actors of the system.

**signatures** – given an operation, the *tuple* made up of the types of its parameters and the type of the return values. Operation signatures are specified during object design.

**subsystem decomposition** – the division of the system into subsystems. Each subsystem is described in terms of its services during system design and its API during object design.

**supportability** – the ease of changing the system after deployment.

**system design object model** – a high level description of the system, including *design goals*, *subsystem decomposition*, hardware/software platform, persistent storage strategy, global control flow, access control policy, and boundary condition strategies. The system design model represents the strategic decisions made by the architecture team that allow subsystem teams to work concurrently and cooperate effectively.

**throughput** – an attribute of the system denoting how much work it can accomplish within a specified amount of time.

**timesheet** – the application that serves to facilitate hour logging by allowing the employees to fill out and submit timesheets electronically.

**tuple** – an ordered set of values. Common uses for the tuple are representing a set of value attributes in a relational database and representing an access right.

**type** – describes the legal values the attribute or variable can take.

**usability** – ease with which the user can learn to operate, prepare inputs for, and interpret outputs of a system or component.

**use case**­ – a general sequence of interactions between one or more actors of the system.

**use case diagram** – UML notation used during requirements elicitation and analysis to represent the functionality of the system. A use case describes a function of the system in terms of a sequence of interactions between an actor and the system. A use case also includes *entry conditions* that must be true before executing the use case and the *exit conditions* that are true at the completion of the use case.

**visibilities** – specifies whether or not the attribute can be accessed by other classes.

**work breakdown structure** – a hierarchical decomposition of project work into tasks. Leaves represent tasks that are assigned to participants. Aggregates represent the work associated with a work product.

