COMP 421 – Project Specification

Tamir Bennatan – 2606014526
Stuart Mashaal – 260639962
Benjamin Taubenblatt 260626105
Pauline Chi – 260659960

Application Overview

Introduction

Companies who sell software as a service (SAAS) are often expected to provide their clients with product support, collect customer feedback, and consider new feature requests. As such, SAAS vendors often need internal systems to keep track of their support requests, split up large requests into actionable tasks, and set deadlines for deliverables.

In this project, we will build a ticket support system designed to help SAAS vendors with these tasks. Our design is inspired by a popular software development toolkit called "Jira". We also incorporate concepts from the agile development framework into our design.

Overview of the Agile Methodology

The agile approach to software development emphasises the importance of self-organized and flexible teams that can plan their own release schedules, and delivery frequently. Agile project management software like Jira supports agile teams by providing a framework in which large projects are split up into more digestible tasks, and each deliverable is given a timeframe.

Most agile teams split their work year into periods of two to four weeks, called "sprints." At the beginning of every sprint, teams get together to determine what work they plan to complete in the upcoming sprint. The basic unit of work is called a "ticket." Tickets are intended to be small, straightforward tasks that can be completed by a single developer. Each ticket is typically given a "point value," which serves as an estimate of how difficult or time consuming the task is, and how much business value it would add to complete the task. When a developer completes a ticket, the associated point value is added to the current sprint's point total. Each sprint has an objective point value to be attained by the end of the sprint - which helps teammates prioritize more valuable and actionable tasks.

Each team is designed to be cross-functional and self-sufficient. As such, a typical Agile team consists of people of many kinds of expertise. For simplicity, one can classify an individual's role in a team as either a *developer* or a *business owner*. Developers are responsible for taking on and completing tickets. Business owners are the individuals that are ultimately responsible for the business outcomes of a team's efforts. Business owners spend much of their time planning out sprints, making sure that developers have all the tools they need to be effective, and interacting with clients to gather requirements. They oftentimes serve the role of a project manager, though a developer may also be a business owner, if he/she is senior enough.

Use Case

Our application borrows these ideas from the Agile framework, but is tailored to vendors that provide large software solutions to clients. In this setting, it is especially important that requests made by clients are split into partitioned into small actionable tasks, as requests regarding large software projects are often too large to be approached by a single developer. Thus, we distinguish between "high level tickets" - which are tickets describing a large request made by a client - and "low level tickets," which result from splitting high level tickets into smaller disjoint tasks.

In our application, clients create high level tickets. It is then the task of a business owner to split up high level tickets into low level tickets, and developers' responsibility to complete the low level tickets. Low level tickets are given point values, and are associated which sprints - which have point objectives. When a low level ticket is completed by a developer, the business owner which created the ticket must verify that the task is complete before the points can be credited. This is to reflect the idea that the business owner interacts with clients most closely, and is thus most knowledgeable of what work is required to appease a client's request.

It is common for SAAS vendors to sell support as a service, as well as the software itself. As such, clients who pay for larger support contracts are entitled to more work hours of a team. To reflect this, our application will enforce a maximum point value each client company can request per sprint. It's reasonable to assume that companies that pay for larger support contracts buy a larger point-per-sprint budget.

Finally, our application will enable a SAAS vendor to analyze their past productivity. Each request made by a client is associated with a long-term project - which allows a vendor to investigate how much effort they've dedicated to each project set up with each of their clients. Each project is assigned one or more business owner, which are held accountable for the completion of the project and are the point of contact for the clients that started the project. Projects are assigned to teams, so that one can investigate each team's contribution to the completion of a project, as well as each individual employee's contributions.

Data Requirements

Here, we outline the data that must be stored by our application, and the relationships between them.

Entity Sets

- **Tickets**: Tickets describe units of work that need to be completed. For each ticket, need to store:
    a. Date created
    b. Status (Todo, In progress, Done)
    c. Description
- **Low-level Tickets**: These are tickets that describe small tasks that are ready to be completed by a developer. In addition to the information regarding stored for a generic ticket must be stored for a low-level ticket, one has the option to store a point estimate to a low-level ticket, which determines how many points the ticket is worth. Low level tickets must be created by partitioning exactly one high-level ticket.
- **High-level Ticket:** When a client requests a support issue, feature request, or any other task, a high-level ticket is created. Before these tasks are ready to be assigned to developers, a business owner must first split the high-level ticket into one or more low-level tickets. No additional information must be stored for high-level tickets - beyond what is stored for generic tickets.
- **Sprint:** A sprint captures the point goals and completed work of a particular team in a fixed period of time. Sprints are only uniquely identified with relation to the team that creates the sprint Each sprint must store:
    a. Sprint number - a counter of the number of sprints each team has schedules. For each new sprint added to the database by a particular team, this counter is incremented.
    b. Sprint start date
    c. Sprint end date
    d. Point goal
- **People:** A SAAS vendor needs to keep track of its own employees, as well as people in customer companies with which the vendor interacts. Thus, for each person in the system, we need to keep track of:
    a. Name
    b. Email
    c. Phone-number
- **Client**: This is a customer which interacts with the vendor. We need to store the same information as a generic person for each client.
- **Employee:** This is a person that works for the SAAS vendor. In addition to the information recorded for a generic person, we need to also store the title of each employee.

- **Business Owner:** This is a special type of employee, which is held accountable for delivering business values on certain projects. We need to store the same information for a business owner as for a generic employee.
- **Developer:** This is an employee that is responsible for taking on and completing low-level tickets. We need to store the same information for a developer s for a generic employee.
- **Teams:** The SAAS vendors employee's are split up into teams, and so we must keep track of the different teams within the company. For each team, we need to keep track of the team's name.
- **Client Companies:** We need to keep track of the companies to which the SAAS vendor sells software to. This is especially important, since we assume that the contracts negotiated by the vendor are such that each client company has a quota for the number of points tickets they can request to be completed per sprint. This quota will be implemented by setting a maximum number of points achieved per sprint through tickets created by each company - called the "points-per-sprint quota". Thus, for each company we must store:
    a. Name
    b. Points per sprint quota
- **Project:** We would like to keep track of the long-term projects initiated by client companies. For each project, we need to store:
    a. Project name
    b. Description
    c. Date project was initiated

Relationships Between Entity Sets

Tickets come in two varieties: high-level and low-level tickets.

When client makes a feature request, creates a support case, or asks the vendor company to complete any other task, that client creates a high level ticket. These high level tickets are then partitioned into low-level tickets by a business owner. A high level ticket can be partitioned into one or more low-level tickets. Even a high level ticket is transformed into exactly one low-level ticket, clients can only create high-level tickets. Further, business owners are the only employees of the SAAS vendor that can partition high-level tickets into low-level tickets. Each low-level ticket *must* be derived from exactly one high-level ticket.

When a high level ticket is created by a client, it must be created regarding to exactly one existing project stored in the database. This means that if a task is requested by a client twice regarding two different projects, two seperate high-level tickets must be created - one for each project. This is to ensure that the work done for each client-initialized project can be investigated.

Each project is initialized by exactly one client company. For each client stored in the database, we must keep track of the company for which that client works. It is assumed that each client works for exactly one client company.

Clients can be stakeholders in one or more project - though it is not a requirement that every client be a stakeholder in a project. If a client is a stakeholder of a project, we can record the role that client plays in the project (e.g "project lead", "feature requester", etc).

Each project can be associated with one or more business owners which are to be held accountable for the completion of that project. It is not necessary for a project to have a business owner held accountable for it (as when a project is first initialized, the vendor may need time to assign a business owner to it.)

Each person whose information is stored in the database is either a client or an employee. One cannot be both a client and an employee.

Employees are further categorized as business owners and developers. It is permitted for an employee to be both developer and a business owner, though he/she must be at least one of the two.

Low-level tickets can be assigned to a developper only by that developper.

An employee can optionally be assigned to one or more projects. This is to help record all the different business functions each employee is involved in. When an employee is assigned to a project, the role of that employee can be recorded (e.g. "project lead", "customer point of contact," etc).

Each employee can be a team member of at most one team. It is possible for an employee to not be a member of any team (for example - the CEO of the vendor is likely not a member of any of the operational teams).

Each sprint is scheduled by exactly one team - the sprint encapsulates the work that team plans to complete in the upcoming tie period. A sprint can only be uniquely identified by looking at the team which scheduled the sprint, and looking at how many sprints that team has scheduled before the one in question (which can be seen by looking at the sprint number - stored for each sprint in the database.)

Low-level tickets can be scheduled for sprints - meaning that the team who scheduled said sprint aims to complete that low-level ticket during the sprint. Low level tickets may be assigned to multiple sprints - as if a team does not complete a ticket in one sprint it can be carried over to another sprint. High-level tickets may not be assigned to sprints.

Functional Requirements

A SAAS vendor will use our application to not only keep track of its business' tasks and performance, but also to execute queries that will help it analyze what parts of the business are less productive than others.

Thus, the actions that will be performed on the database are broadly categorized into actions that add new records to the database, and queries that provide answers to business questions a vendor might have.

Adding New Entities

Specifications of tasks that a vendor must complete are recorded as tickets. As discussed, tickets are categorized into high-level tickets and low-level tickets.

Each client should be able to create one or more high-level ticket. Except for clients, no other person in the system can add a high-level ticket to the database.

Low-level tickets may be added to the database only if they are derived from a high-level ticket by a business owner employee. No other type of employee may partition high-level tickets to create low-level tickets, and low-level tickets cannot be created except by splitting up exactly one high-level ticket.

Each team needs to add a new sprint entity to the database at the end of the previous sprint. A sprint cannot be added if it is not associated with exactly one team. The database should allow for an employee's listed team to change. The database should also allow for the creation of new teams.

One should be able to add information about new people - both clients and employees - to the database. In the case of employees, one should be able to update the information as to who manages whom - as employees may be promoted to more senior positions and manage more more people, or new employees may need to be assigned a manager.

New client companies can be added to the database. Further, each client company can create records for new projects. We should be able to update the information about what role each client stakeholder has in each project (if any), and what business owner is to be held accountable for the completion of each project (if any.)

Permitted Queries

One should be able to calculate the total number of points achieved, grouped by any combination of sprint, team, developer, project, and high-level ticket. One should also be able to extract information about the historical performance of teams and individuals, such as the number of points each team has over-delivered/under-delivered for all past sprints - relative to each sprint's point goal.

By investigating the status of the tickets in a sprint, we should also be able to calculate the number of points completed in sprint, and the number of points left to go (scheduled but not completed tickets). We should be able to compute the amount of time a ticket has been active, and the number of sprints each low-level ticket was scheduled for (if any).

We should also be able to deduce the amount of time it took for each low-level ticket to be completed (since the time it was derived from a high-level ticket), the total amount of developers' time spent working on each ticket (approximated by the time between when the ticket was assigned to a developer and the time it was completed). We should also be able to compute how much time each developer has spent working on all the tickets he/she was assigned to, across all sprints.

To ensure that no company exceeds their points-per-sprint quota, we should be able to calculate the total point value of all the tickets scheduled by a company on each of the teams' current sprints.

Relational Mapping

- Person(pid, name, email, phone_number)
- Employee(pid, team, title)
- Client(pid, client_company_id) (*client_company_id* references *Client_company*)
    - To ensure participation constraint, i.e every client must be part of a client company, the field *client_company_id* must be NOT NULL
- Business_owner(pid)
- Developer(pid)
- Managers(manager_id, managee_id) (*manager_id* references *Employee*(pid), *managee_id* references *Employee*(pid))
- Client_company(client_company_id, name, points_per_sprint)
- Project(proj_id, name, description, date_created, client_company_id) (*client_company_id* references *Client_company*)
    - To ensure participation constraint, i.e every project must be initialized by a company, the field *client_company_id* must be NOT NULL
- Project_stakeholders(pid, proj_id, role) (*pid* references *Client*, *proj_id* references *Project*)
- High_ticket(tid, proj_id, requesting_client, urgency, date_created, status, description) (*proj_id* references *Project, requesting_client* references *Client(pid)*)

    - To ensure participation constraint, i.e every high-level ticket is made as part of a project, the field *proj_id* must be NOT NULL.
- Low_ticket(tid, source_high_ticket, source_busi_owner, urgency, point_estimate, date_created, status, description) (*source_high_ticket* references *High_ticket*(tid), *source_busi_owner* references *Business_owner*(pid))
- Ticket_assignment(tid, dev_id, date_assigned) (*tid* references *Low_ticket*, *dev_id* references *Developer*(pid))
- Project_assignment(proj_id, pid, role) (*proj_id* references *Project*, *pid* references *Employee*)
- Project_poc(pid, proj_id) (*pid* references *Business_owner*, *proj_id* references *Project)*
    - Note: "Project_poc" represents each project's "point of contact" - i.e the business owner held accountable for the success of said project.
- Sprint(team_name, sprint_number, start_date, end_date, point_goal) (*team_name* references *Team(name)*)
- Ticket_scheduling(tid, sprint_number, team_name, date_scheduled) (*tid* references *Low_ticket, sprint_number* references *Sprint*, *team_name* references *Sprint*)