Project Task management for IT Company

Project Details

Student Name: Fabio Vitalba

Student ID: 22374

Project Name: Project Task management for IT Company

Date: 18.02.2025

Index

1. Specification

1.1. Specification

A medium sized Software Development company (about 20-30 software developers of various seniority levels) wants to track their **project**'s milestones deadlines and wants to track their accuracy in task estimates, as well as the developer workload. For that we need to track projects that are uniquely identified by a code, and we also want to store a description, customer name, customer address, customer VAT registration no., a starting date, an ending date, and a status (active, complete, or on hold).

Each **project** has a list of **milestones** which need to be passed in order to reach the end of the project. The milestones are in sequential order and also contain a starting date and ending date. The starting date must be greater or equal to the project's starting date and the ending date must be smaller or equal to the project's ending date. Milestones are marked as critical if they are part of the minimum viable product and therefore are crucial for the project's completion. Each milestone in a project has a unique code, however, the same milestone code may be used on different projects.

Each **milestone** has a list of associated **tasks**. Each task is uniquely identified by an ID (even across projects) and has a technical estimate (in hours), the ID of the developer that estimated it, a description, a status (not started, authorized, in progress, test, completed) and a due date. Each task is assigned to a developer and for each task we want to track the number of hours spent by the developer(s). We also want to track the date the task was created, the date it was estimated and the date it was released (Status change from "not started" to "authorized"). Additionally, when a task is completed, we want to store the date of completion for the task.

Tasks always fall into one of the following three categories: **bug-fix-tasks**, **feature-tasks**, or **code-review-tasks**.
For **bug-fix tasks** we want to track the impact (software-interaction still possible, software-interaction requires work-arounds, software-interaction impossible). High impact bug-fixes must be assigned to mid- or senior-developers and must be resolved within 4h of the incident report. Medium impact bug-fixes have 48h for resolution.
For **feature-tasks** we want to track complexity (low, medium, high). High complexity tasks may only be assigned to senior developers. While junior developers should almost exclusively receive low complexity tasks to get acclimated with the projects.
For **code-review-tasks** we have a reviewer developer as well as a review result

(success, on hold, rejected). Code-review tasks cannot exceed an actual work effort of 4h. Each code-review task is reviewed by a single developer.

Whenever a task's due date is postponed, we want to log this (**Due Date changes**). Together with the previous and new due date, we want to track a reason and the task's unique code. We also want to track the hours worked until the due date was changed, as well as the current estimate. This will help track whether a specific task is being postponed without making progress. Tasks might also be postponed before having an estimate.

A **developer** is uniquely identified by their employee ID and has a name, email, hiring date, a list of skills, and a weekly capacity (in hours). Each developer has a list of assigned tasks. We also want to track the seniority of each developer (junior, mid, senior). Developers are not strictly bound to a project, but each project has a **lead developer** who is in charge for the bulk of estimates of that specific project and keeps track of the project's due date along with the project developer's load.

Lead developers are senior developers that have completed a series of special training in order to fulfill their leadership role.

Each time a developer works on a given task, we want to track the amount of time spent working on the task. For each of these **time logs** we would like to store the date, the task's ID, the developer's ID, and the amount of hours worked. A developer may work on the same task for multiple days. The developer may even be interrupted during their work, causing them to work on the same task, on the same day but for different periods. Only tasks that are not yet completed may be worked on.

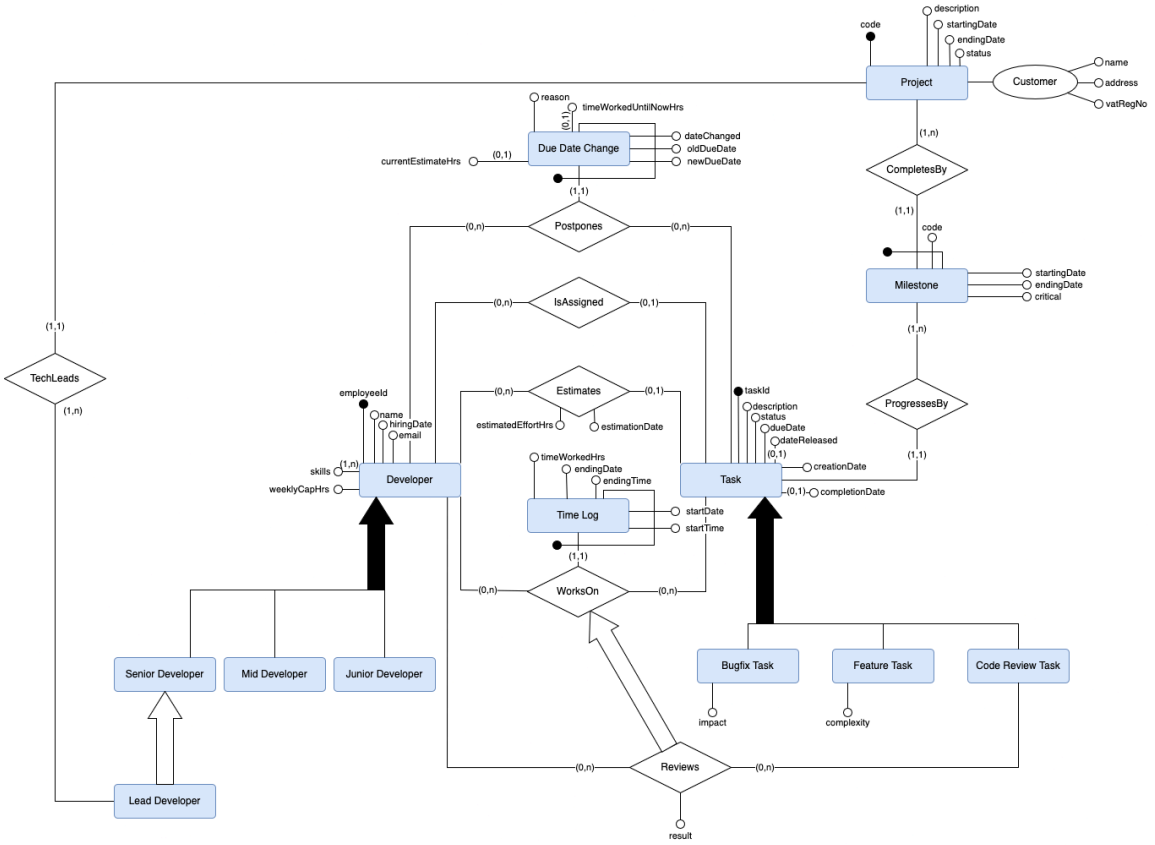
Working overtime should be avoided, therefore developers are only allowed to exceed their weekly capacity by 10%. Developers should also avoid working on more than 3 tasks at any one moment, as this leads to lots of overhead and reduces efficiency. Therefore, no more than 3 tasks in status “authorized” or “in progress” may be assigned to any developer at any time.

1.2. Glossary

Term	Description	Synonyms	Connections
Project	Order given by a Customer to develop a software	Order, Job	Customer, Milestones, Lead Developer
Customer	Person ordering software	Client, Buyer	Project
Milestone	Checkpoint marking progress in a Project	Checkpoint, Deliverable, Progress	Project, Task

Term	Description	Synonyms	Connections
Task	Activity to be executed by a Developer	Job, Assignment, To-do	Milestone, Developer, Time Log, Due Date Change
(Software) Developer	Person who writes, and maintains software code	Programmer, Engineer	Task, Time Log, Due Date Change
Bugfix Task	Task to correct software defects	Defect fix task, Issue resolution, Correction	Developer, Task
Feature Task	Task to implement new software functionality	Functionality, Feature addition	Developer, Task
Code Review Task	Task to review software	Pull Request task, Code inspection task	Developer, Task
Time Log	Amount of time spent on a task	Work log, Timesheet, Activity log	Developer, Task
Due Date Change	Update of task deadline	Deadline update	Developer, Task
Junior Developer	Developer with little (0-3) years of experience	Trainee developer, Beginner developer	Developer
Mid Developer	Developer with moderate (3-6) years of experience	Intermediate developer	Developer
Senior Developer	Developer with many (6+) years of experience	Experienced developer	Developer, Lead Developer
Lead Developer	Developer in charge for the technical aspect of a Project	Technical lead, Lead software engineer	Senior Developer, Project
Estimate	Act of predicting the time effort of a task	Approximation, Effort prediction	Developer, Task
Postponing	Act of delaying task's due date to a later date	Delaying, Rescheduling	Developer, Task

1.3. ER Schema



1.4. External Constraints

- **[Project]:** The attribute startingDate must be before the attribute endingDate.
- **[Milestone]:** The attribute startingDate must be before the attribute endingDate.
- **[Milestone]:** If a [Milestone] is in relation with a [Project] through <CompletesBy>, then that milestone must have an attribute startingDate that is after or equal to the attribute startingDate of that [Project].
- **[Milestone]:** If a [Milestone] is in a relation with a [Project] through <CompletesBy>, then that milestone must have an attribute endingDate that is before or equal to the attribute endingDate of that [Project].
- **[Bug-fix Tasks]:** If the attribute impact is “High”, then the task must be assigned using <IsAssigned> to [Mid Developers] or [Senior Developers].
- **[Feature-Tasks]:** If the attribute complexity is “High” then the task may only be assigned using <IsAssigned> to [Senior Developers].
- **<IsAssigned>:** Each [Developer] may be assigned, through <IsAssigned>, at most 3 [Task]s whose status is either “Authorized” or “In progress”.

- **<WorksOn>:** A [Developer] that is in a relation with a [Task] through <WorksOn> must also be in a relation through <IsAssigned> with that same [Task].
- **<WorksOn>:** If a [Task] has status “Completed”, and has a relation with a [Time Log] through <WorksOn>, then that [Time Log] must have an ending date ahead or equal to the completion date of the task.
- **[Time Log]:** The attributes startingDate and startingTime must be before endingDate and endingTime. Note that startingTime could be earlier than endingTime, if endingDate is at least one day after startingDate.

1.5. Data Dictionaries

1.5.1. Data Dictionary Entities

Entity	**Description**	**Attributes**	**Identifiers**
Project	Software Order	code, description, startingDate, endingDate, status, customer(name, address, vatRegNo)	{ code }
Milestone	Project Progress Checkpoint	code, startingDate, endingDate, critical	{ code, Project }
Task	Project development activity	taskId, description, status, dueDate, dateReleased*, creationDate, completionDate*	{ taskId }
Bugfix Task	Activity related to software defects	taskId, description, status, dueDate, dateReleased*, creationDate, completionDate*, impact	{ taskId }

Entity	Description	Attributes	Identifiers
Feature Task	Activity for new functionalities	taskId, description, status, dueDate, dateReleased*, creationDate, completionDate*, complexity	{ taskId }
Code Review Task	Software verification activity	taskId, description, status, dueDate, dateReleased*, creationDate, completionDate*	{ taskId }
Developer	Person who writes and maintains software	employeeId, name, hiringDate, email, skills (1,n), weeklyCapHrs	{ employeeId }
Junior Developer	Developer with little experience	employeeId, name, hiringDate, email, skills (1,n), weeklyCapHrs	{ employeeId }
Mid Developer	Developer with moderate experience	employeeId, name, hiringDate, email, skills (1,n), weeklyCapHrs	{ employeeId }
Senior Developer	Developer with lots of experience	employeeId, name, hiringDate, email, skills (1,n), weeklyCapHrs	{ employeeId }
Lead Developer	Developer in charge for the technical aspect of a Project	employeeId, name, hiringDate, email, skills (1,n), weeklyCapHrs	{ employeeId }

Entity	Description	Attributes	Identifiers
Time Log	Time spent on a task	startDate, startTime, endingDate, endingTime, timeWorkedHrs	{ startDate, startTime, Developer, Task }
Due Date Change	Task deadline delay	dateChanged, oldDueDate, newDueDate, reason, timeWorkedUntilNowHrs*, currentEstimateHrs*	{ dateChanged, oldDueDate, newDueDate, Developer, Task }

1.5.2. Data Dictionary Relationships

Relationship	Description	Components	Attributes	Identifiers
CompletesBy	A Project is complete when all its Milestones are completed	Project, Milestone		
ProgressesBy	A Milestone is complete when all its Tasks are completed	Milestone, Task		
IsAssigned	A Task belongs to a Developer	Task, Developer		
Estimates	A Developer predicts the time effort of a task	Task, Developer	estimatedEffortHrs, estimationDate	
WorksOn	A Developer works on a Task	Task, Developer, Time Log		
Reviews	A Developer verifies a (piece of) software	Code Review Task, Developer	result	
Postpones	A Developer reschedules the due date of a Task	Task, Developer, Due Date Change		
TechLeads	A Developer decides/leads on technical aspects of the Project	Lead Developer, Project		

1.6. Table of Volumes

Note that in the following table of volumes we assume the following:

- The company has about 20-30 developers, so we work with the highest value.
- We assume that only 1/3 is of senior level and of the remaining 2/3 about 1/3 will be of junior level. The rest are mid level developers.
- The company will complete an average of 15 projects a year. And we assume that we are able to archive data after 10 years, so we will have an average of 150 Projects in the database.
- Each Project has an average of 4 Milestones to complete.
- Each Milestone consists of an average of 50 Tasks.
- Tasks will be evenly distributed between Bugfix-, Feature-, and Code-Review-Tasks.
- There is an average of 3 Lead developers.
- Postponing of Tasks occurs only on 1/3 of tasks.
- Some Tasks might not be worked on, and some Tasks might be worked on on different dates. On average we assume that we will have 2x more Time Logs than Tasks.
- The number of Reviews is closer to the number of Code Review Tasks. Almost 1:1, but some will require more reviews, and some Code Review Tasks are not reviewed.

Concept	**Construct**	**Volume**
Project	Entity	150
Milestone	Entity	600
Task	Entity	30.000
Bugfix Task	Entity	10.000
Feature Task	Entity	10.000
Code Review Task	Entity	10.000
Developer	Entity	30
Junior Developer	Entity	7
Mid Developer	Entity	13
Senior Developer	Entity	10
Lead Developer	Entity	3
Time Log	Entity	60.000
Due Date Change	Entity	10.000

Concept	Construct	Volume
CompletesBy	Relationship	600
ProgressesBy	Relationship	30.000
IsAssigned	Relationship	30.000
Estimates	Relationship	30.000
WorksOn	Relationship	60.000
Reviews	Relationship	10.000
Postpones	Relationship	10.000
TechLeads	Relationship	150

1.7. Table of Operations

1.7.1. Operations

1. Assign a Task to a Developer
2. A Developer gives a time Estimate for a Task
3. Find all overdue Tasks in a Project
4. Postpone a Task
5. Assign a period of time as worked time to a Task of a Developer
6. Find all Tasks without estimate in a Project
7. Find all assigned, workable (in status authorized or in progress) Tasks for the current week for all Developers
8. Calculate the amount of hours required to complete the Project (using estimates of uncompleted Tasks)

1.7.2. Table of Operations

Note that in the following table of operations we assume the following:

- Assuming 15 Projects a year with an average of 4 Milestones and 50 Tasks per Milestone we have 3.000 Tasks a year. This results in 250 Tasks being created each month and as such about an average of 8 (rounded up to 10) Tasks are assigned and estimated per day.
- Project update meetings are done once per week.
- We have about 1.000 Tasks postponed a year and that leads to an average of 20 Tasks a week being postponed.
- We have about 6.000 Time Logs a year and so an average of 16 Logs a day (rounded up to 20).

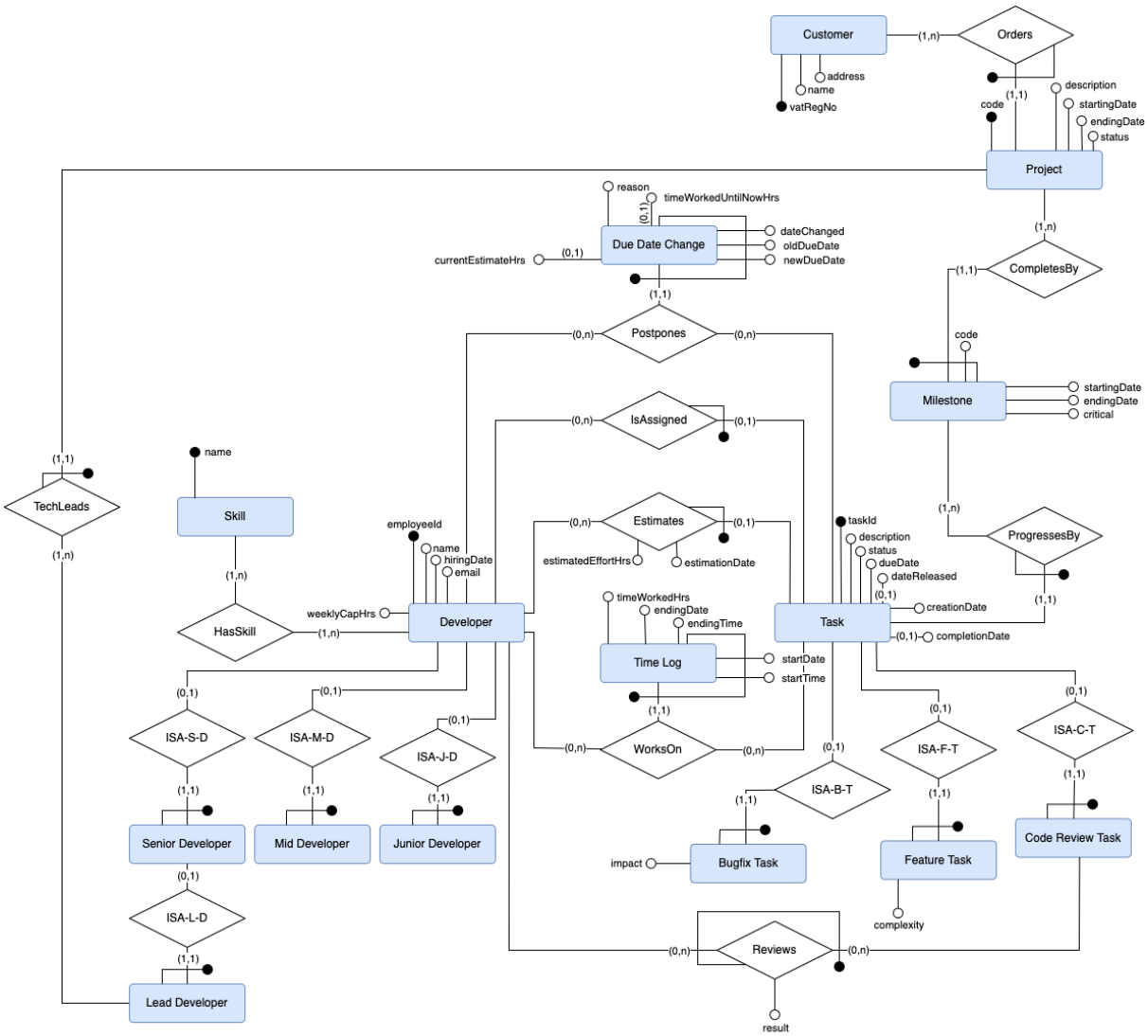
Operation	Type	Frequency
1	Interactive	10 / day
2	Interactive	10 / day
3	Interactive	1 / week
4	Interactive	20 / week
5	Interactive	20 / day
6	Interactive	1 / week
7	Interactive	1 / week
8	Interactive	1 / week

2. Restructured Specification

2.1. ER Schema

Note that in the following schema we maintain the redundancies of:

- **[Time Log]:** Between the attributes startDate, startTime, endingDate and endingTime and the attribute timeWorkedHrs. This redundancy remains to avoid having to calculate the hours worked each time the instances are retrieved.
- **[Due Date Change]:** Between the attribute timeWorkedUntilNowHrs and the relationship <WorksOn> and entity [Time Log]. In order to avoid having to calculate the timeWorkedUntilNowHrs each time the instances are retrieved.



2.2. External Constraints

- **[Project]:** The attribute startingDate must be before the attribute endingDate.
- **[Milestone]:** The attribute startingDate must be before the attribute endingDate.
- **[Milestone]:** If a [Milestone] is in relation with a [Project] through <CompletesBy>, then that milestone must have an attribute startingDate that is after or equal to the attribute startingDate of that [Project].
- **[Milestone]:** If a [Milestone] is in a relation with a [Project] through <CompletesBy>, then that milestone must have an attribute endingDate that is before or equal to the attribute endingDate of that [Project].
- **[Bug-fix Tasks]:** If the attribute impact is “High”, then the task must be assigned using <IsAssigned> to [Mid Developers] or [Senior Developers].
- **[Feature-Tasks]:** If the attribute complexity is “High” then the task may only be assigned using <IsAssigned> to [Senior Developers].
- **<IsAssigned>:** Each [Developer] may be assigned, through <IsAssigned>, at most 3 [Task]s whose status is either “Authorized” or “In Progress”.
- **<WorksOn>:** A [Developer] that is in a relation with a [Task] through <WorksOn> must also be in a relation through <IsAssigned> with that same [Task].
- **<WorksOn>:** If a [Task] has status “Completed”, and has a relation with a [Time Log] through <WorksOn>, then that [Time Log] must have an ending date ahead or equal to the completion date of the task.
- **[Time Log]:** The attributes startingDate and startingTime must be before endingDate and endingTime. Note that startingTime could be earlier than endingTime, if endingDate is at least one day after startingDate.
- **<Reviews>:** For each instance I of the schema, for each instance (Developer:d, CodeReviewTask:crt) of <Reviews> in I, let t be the instance of Task such that (CodeReviewTask:crt,Task:t) is an instance of ISA-C-T in I (note that t always exists and is unique). Then we have that (Developer:d,Task:t) is an instance of <WorksOn> in I.
- **[Task]:** Each instance of [Task] participates in one of the following: <ISA-B-T>, <ISA-F-T>, or <ISA-C-T>, but never in more than one of them.
- **[Developer]:** Each instance of [Developer] participates in one of the following: <ISA-S-D>, <ISA-M-D>, or <ISA-J-D>, but never in more than one of them.

2.3. Data Dictionaries

2.3.1. Data Dictionary Entities

Entity	**Description**	**Attributes**	**Identifiers**
Project	Software Order	code, description, startingDate, endingDate, status,	{ code }
Customer	Entity or Person ordering Software Project	vatRegNo, name, address	{ vatRegNo }
Milestone	Project Progress Checkpoint	code, startingDate, endingDate, critical	{ code, Project }
Task	Project development activity	taskId, description, status, dueDate, dateReleased*, creationDate, completionDate*	{ taskId }
Bugfix Task	Activity related to software defects	taskId, description, status, dueDate, dateReleased*, creationDate, completionDate*, impact	{ taskId }
Feature Task	Activity for new functionalities	taskId, description, status, dueDate, dateReleased*, creationDate, completionDate*, complexity	{ taskId }
Code Review Task	Software verification activity	taskId, description, status, dueDate, dateReleased*, creationDate, completionDate*	{ taskId }

Entity	Description	Attributes	Identifiers
Developer	Person who writes and maintains software	employeeId, name, hiringDate, email, weeklyCapHrs	{ employeeId }
Junior Developer	Developer with little experience	employeeId, name, hiringDate, email, weeklyCapHrs	{ employeeId }
Mid Developer	Developer with moderate experience	employeeId, name, hiringDate, email, weeklyCapHrs	{ employeeId }
Senior Developer	Developer with lots of experience	employeeId, name, hiringDate, email, weeklyCapHrs	{ employeeId }
Lead Developer	Developer in charge for the technical aspect of a Project	employeeId, name, hiringDate, email, weeklyCapHrs	{ employeeId }
Time Log	Time spent on a task	startDate, startTime, endingDate, endingTime, timeWorkedHrs	{ startDate, startTime, Developer, Task }
Due Date Change	Task deadline delay	dateChanged, oldDueDate, newDueDate, reason, timeWorkedUntilNowHrs*, currentEstimateHrs*	{ dateChanged, oldDueDate, newDueDate, Developer, Task }
Skill	Competence acquired by a Developer	name	{ name }

2.3.2. Data Dictionary Relationships

Relationship	Description	Components	Attributes	Identifiers
Orders	A Customers gives the order for a software Project	Customer, Project		{ Project }

Relationship	Description	Components	Attributes	Identifiers
CompletesBy	A Project is complete when all its Milestones are completed	Project, Milestone		{ Milestone }
ProgressesBy	A Milestone is complete when all its Tasks are completed	Milestone, Task		{ Task }
IsAssigned	A Task belongs to a Developer	Task, Developer		{ Task }
Estimates	A Developer predicts the time effort of a task	Task, Developer	estimatedEffortHrs, estimationDate	{ Task }
WorksOn	A Developer works on a Task	Task, Developer, Time Log		{ Developer, Task, Time Log }
Reviews	A Developer verifies a (piece of) software	Code Review Task, Developer	result	{ Developer, Code Review Task }
Postpones	A Developer reschedules the due date of a Task	Task, Developer, Due Date Change		{ Developer, Task, Due Date Change }
TechLeads	A Developer decides/leads on technical aspects of the Project	Lead Developer, Project		{ Project }
HasSkill	A developer posses a Skill	Developer, Skill		{ Developer, Skill }
ISA-B-T	Bugfix Task is a Task	Bugfix Task, Task		{ Bugfix Task }
ISA-F-T	Feature Task is a Task	Feature Task, Task		{ Feature Task }
ISA-C-T	Code Review Task is a Task	Code Review Task, Task		{ Code Review Task }
ISA-S-D	Senior Developer is a Developer	Senior Developer, Developer		{ Senior Developer }
ISA-M-D	Mid Developer is a Developer	Mid Developer, Developer		{ Mid Developer }
ISA-J-D	Junior Developer is a Developer	Junior Developer, Developer		{ Junior Developer }

Relationship	Description	Components	Attributes	Identifiers
ISA-L-D	Lead Developer is a Senior Developer	Lead Developer, Senior Developer		{ Lead Developer }

2.4. Table of Volumes

Note that in the following table of volumes we make the same assumptions as in the Table of Volumes presented in 1.6.

Additionally, we make the following assumptions:

- Most of the projects are ordered by new customers. However, about a third of projects are ordered by customers who have previously ordered a project.
- We track about 25 unique skills and one skill might be associated with more than one developer.
- For each developer we keep track of an average of 5 Skills.

Concept	Construct	Volume
Customer	Entity	100
Project	Entity	150
Milestone	Entity	600
Task	Entity	30.000
Bugfix Task	Entity	10.000
Feature Task	Entity	10.000
Code Review Task	Entity	10.000
Developer	Entity	30
Skill	Entity	25
Junior Developer	Entity	7
Mid Developer	Entity	13
Senior Developer	Entity	10
Lead Developer	Entity	3
Time Log	Entity	60.000
Due Date Change	Entity	10.000
Orders	Relationship	150
CompletesBy	Relationship	600

Concept	Construct	Volume
ProgressesBy	Relationship	30.000
IsAssigned	Relationship	30.000
Estimates	Relationship	30.000
WorksOn	Relationship	60.000
Reviews	Relationship	10.000
Postpones	Relationship	10.000
HasSkill	Relationship	150
TechLeads	Relationship	150
ISA-B-T	Relationship	10.000
ISA-F-T	Relationship	10.000
ISA-C-T	Relationship	10.000
ISA-S-D	Relationship	10
ISA-M-D	Relationship	13
ISA-J-D	Relationship	7
ISA-L-D	Relationship	3

2.5. Table of Operations

1.7.1. Operations

1. Assign a Task to a Developer
2. A Developer gives a time Estimate for a Task
3. Find all overdue Tasks in a Project
4. Postpone a Task
5. Assign a period of time as worked time to a Task of a Developer
6. Find all Tasks without estimate in a Project
7. Find all assigned, workable (in status authorized or in progress) Tasks for the current week for all Developers
8. Calculate the amount of hours required to complete the Project (using estimates of uncompleted Tasks)

1.7.2. Table of Operations

Note that in the following table of operations we make the same assumptions as in the Table of Operations presented in 1.7.2.

Operation	Type	Frequency
1	Interactive	10 / day
2	Interactive	10 / day
3	Interactive	1 / week
4	Interactive	20 / week
5	Interactive	20 / day
6	Interactive	1 / week
7	Interactive	1 / week
8	Interactive	1 / week

3. Cost Evaluation

3.1. Access table for Operation 1 (Assign a Task to a Developer)

Concept	Construct	Accesses	Type
Developer	Entity	1	R
IsAssigned	Relationship	1	W
Task	Entity	1	R

3.2. Access table for Operation 2 (A Developer gives a time Estimate for a Task)

Concept	Construct	Accesses	Type
Task	Entity	1	R
Estimates	Relationship	1	W
Developer	Entity	1	R

3.3. Access table for Operation 3 (Find all overdue Tasks in a Project)

Concept	Construct	Accesses	Type
Project	Entity	1	R
CompletesBy	Relationship	4	R
Milestone	Entity	4	R
ProgressesBy	Relationship	200	R
Task	Entity	200	R

3.4. Access table for Operation 4 (Postpone a Task)

Concept	Construct	Accesses	Type
Task	Entity	1	R
Postpones	Relationship	1	W
Due Date Change	Entity	1	W
Developer	Entity	1	R

3.5. Access table for Operation 5 (Assign a period of time as worked time to a Task of a Developer)

Concept	Construct	Accesses	Type
Task	Entity	1	R
WorksOn	Relationship	1	W
Time Log	Entity	1	W
Developer	Entity	1	R

3.6. Access table for Operation 6 (Find all Tasks without estimate in a Project)

Concept	Construct	Accesses	Type
Project	Entity	1	R
CompletesBy	Relationship	4	R
Milestone	Entity	4	R
ProgressesBy	Relationship	200	R
Task	Entity	200	R
Estimates	Relationship	200	R

3.7. Access table for Operation 7 (Find all assigned, workable (in status authorized or in progress) Tasks for the current week for all Developers)

Concept	Construct	Accesses	Type
Developer	Entity	30	R
IsAssigned	Relationship	150	R
Task	Entity	150	R

3.8. Access table for Operation 8 (Calculate the amount of hours required to complete the Project (using estimates of uncompleted Tasks))

Concept	Construct	Accesses	Type
Project	Entity	1	R
CompletesBy	Relationship	4	R
Milestone	Entity	4	R
ProgressesBy	Relationship	200	R
Task	Entity	200	R

Concept	Construct	Accesses	Type
Estimates	Relationship	200	R

4. Direct Translation to the Relational Model

Customer(<u>vatRegNo</u>, name, address)
- **inclusion:** Customer(vatRegNo) ⊆ Orders(customer)

Project(<u>code</u>, description, startingDate, endingDate, status)
- **inclusion:** Project(code) ⊆ Milestone(project)

Milestone(<u>project</u>, <u>code</u>, startingDate, endingDate, critical)
- **foreign key:** Milestone(project) ⊆ Project(code)
- **inclusion:** Milestone(project, code) ⊆ ProgressesBy(project, milestone)

Task(<u>taskId</u>, description, status, dueDate, dateReleased*, creationDate, completionDate*)
- **foreign key:** Task(taskId) ⊆ ProgressesBy(task)
- **completeness:** Task(taskId) ⊆ BugfixTask(taskId) ∪ FeatureTask(taskId) ∪ CodeReviewTask(taskId)

BugfixTask(<u>taskId</u>, impact)
- **foreign key:** BugfixTask(taskId) ⊆ Task(taskId)
- **disjointness:** BugfixTask(taskId) ∩ FeatureTask(taskId) = ∅
- **disjointness:** BugfixTask(taskId) ∩ CodeReviewTask(taskId) = ∅

FeatureTask(<u>taskId</u>, complexity)
- **foreign key:** FeatureTask(taskId) ⊆ Task(taskId)
- **disjointness:** FeatureTask(taskId) ∩ CodeReviewTask(taskId) = ∅

CodeReviewTask(<u>taskId</u>)
- **foreign key:** CodeReviewTask(taskId) ⊆ Task(taskId)

Developer(<u>employeeId</u>, name, hiringDate, email, weeklyCapHrs)
- **completeness:** Developer(employeeId) ⊆ JuniorDeveloper(employeeId) ∪ MidDeveloper(employeeId) ∪ SeniorDeveloper(employeeId)
- **inclusion:** Developer(employeeId) ⊆ HasSkill(developer)

JuniorDeveloper(<u>employeeId</u>)
- **foreign key:** JuniorDeveloper(employeeId) ⊆ Developer(employeeId)
- **disjointness:** JuniorDeveloper(employeeId) ∩ MidDeveloper(employeeId) = ∅
- **disjointness:** JuniorDeveloper(employeeId) ∩ SeniorDeveloper(employeeId) = ∅

MidDeveloper(employeeId)
foreign key: MidDeveloper(employeeId) ⊆ Developer(employeeId)
disjointness: MidDeveloper(employeeId) ∩ SeniorDeveloper(employeeId) = ∅

SeniorDeveloper(employeeId)
foreign key: SeniorDeveloper(employeeId) ⊆ Developer(employeeId)

LeadDeveloper(employeeId)
foreign key: LeadDeveloper(employeeId) ⊆ SeniorDeveloper(employeeId)
inclusion: LeadDeveloper(employeeId) ⊆ TechLeads(leadDeveloper)

TimeLog(developer, task, startDate, startTime, endingDate, endingTime, timeWorkedHrs)
foreign key: TimeLog(developer) ⊆ Developer(employeeId)
foreign key: TimeLog(task) ⊆ Task(taskId)

DueDateChange(developer, task, dateChanged, oldDueDate, newDueDate, reason, timeWorkedUntilNowHrs*,currentEstimateHrs*)
foreign key: DueDateChange(developer) ⊆ Developer(employeeId)
foreign key: DueDateChange(task) ⊆ Task(taskId)

Skill(name)
inclusion: Skill(name) ⊆ HasSkill(skill)

Orders(project, customer)
foreign key: Orders(project) ⊆ Project(code)
foreign key: Orders(customer) ⊆ Customer(vatRegNo)

ProgressesBy(task, project, milestone)
foreign key: ProgressesBy(task) ⊆ Task(taskId)
foreign key: ProgressesBy(project, milestone) ⊆ Milestone(project, code)

IsAssigned(task, developer)
foreign key: IsAssigned(task) ⊆ Task(taskId)
foreign key: IsAssigned(developer) ⊆ Developer(employeeId)

Estimates(task, developer, estimatedEffortHrs, estimationDate)
foreign key: Estimates(task) ⊆ Task(taskId)
foreign key: Estimates(developer) ⊆ Developer(employeeId)

Reviews(developer, codeReviewTask, result)
foreign key: Reviews(developer) ⊆ Developer(employeeId)

foreign key: Reviews(codeReviewTask) ⊆ CodeReviewTask(taskId)

HasSkill(developer, skill)
foreign key: HasSkill(developer) ⊆ Developer(employeeId)
foreign key: HasSkill(skill) ⊆ Skill(name)

TechLeads(project, leadDeveloper)
foreign key: TechLeads(project) ⊆ Project(code)
foreign key: TechLeads(leadDeveloper) ⊆ LeadDeveloper(employeeId)

External constraints

- **Project:** For each tuple (*code, desc, startDate, endDate, status*) in Project, *startDate* ≤ *endDate*.
- **Milestone:** For each tuple (*proj, code, startDate, endDate, crit*) in Milestone, *startDate* ≤ *endDate*.
- **Milestone:** For each tuple (*pCode, mCode, mStartDate, mEndDate, mCrit*) in Milestone, there exists a tuple (*pCode, pDesc, pStartDate, pEndDate, pStatus*) in Project such that *pStartDate* ≤ *mStartDate* and *pEndDate* ≥ *mEndDate*.
- **BugFixTask:** For each tuple (*t, d*) in IsAssigned, if there exists a tuple (*t, impact*) where *impact* = “High”, then there exists a tuple (*d*) in either MidDeveloper or SeniorDeveloper.
- **FeatureTask:** For each tuple (*t, sd*) in IsAssigned, if there exists a tuple (*t, complexity*) in FeatureTask where *complexity* = “High”, then there exists a tuple (*sd*) in SeniorDeveloper.
- **IsAssigned:** For every Developer *d*, there may only be at most 3 tuples (t_1, *d*), (t_2, *d*), and (t_3, *d*) in IsAssigned where t_1, t_2, and t_3 each correspond to a tuple in Task, where each tuple in Task has an attribute *status* equal to “Authorized” or equal to “In progress”.
- **TimeLog:** For each tuple (*d, t, sd, st, ed, et, twh*) in TimeLog there exists a tuple (*t, d*) in IsAssigned.
- **TimeLog:** For each tuple (*d, t, sd, st, endingDate, et, twh*) in TimeLog there exists a tuple (*t, desc, status, dd, dr, cd, completionDate, pro, mil, dev*) in Task such that *completionDate* ≥ *endingDate*.
- **TimeLog:** For each tuple (*d, t, startDate, startTime, endingDate, endingTime, twh*) in TimeLog, *startDate* ≤ *endingDate*. If *endingDate* = *startDate*, then *startTime* ≤ *endingTime*.
- **Reviews:** For each tuple (*d, crt, r*) in Reviews, there exists a tuple (*d, crt, sd, st, ed, et, twh*) in TimeLog.

5. Restructured Relational Model

Note that for the restructuring we consider the following reasons:

- With operations 3, 6, and 8 we always need to access the Tasks related to a Milestone, related to a Project. Therefore we want to reduce the number of accessed objects by merging ProgressesBy into Task.
- In operations 1, and 7 we want to know which developer a Task is assigned to. In order to reduce the number of accessed objects, we merge IsAssigned into Task, even if this leads to null values in Task.

Customer(vatRegNo, name, address)
inclusion: Customer(vatRegNo) ⊆ Orders(customer)

Project(code, description, startingDate, endingDate, status)
inclusion: Project(code) ⊆ Milestone(project)

Milestone(project, code, startingDate, endingDate, critical)
foreign key: Milestone(project) ⊆ Project(code)
inclusion: Milestone(project, code) ⊆ Task(project, milestone)

Task(taskId, description, status, dueDate, dateReleased*, creationDate, completionDate*, project, milestone, assignedDeveloper*)
foreign key: Task(project, milestone) ⊆ Milestone(project, code)
foreign key: Task(assignedDeveloper) ⊆ Developer(employeeId)
completeness: Task(taskId) ⊆ BugfixTask(taskId) ∪ FeatureTask(taskId) ∪ CodeReviewTask(taskId)

BugfixTask(taskId, impact)
foreign key: BugfixTask(taskId) ⊆ Task(taskId)
disjointness: BugfixTask(taskId) ∩ FeatureTask(taskId) = ∅
disjointness: BugfixTask(taskId) ∩ CodeReviewTask(taskId) = ∅

FeatureTask(taskId, complexity)
foreign key: FeatureTask(taskId) ⊆ Task(taskId)
disjointness: FeatureTask(taskId) ∩ CodeReviewTask(taskId) = ∅

CodeReviewTask(taskId)
foreign key: CodeReviewTask(taskId) ⊆ Task(taskId)

Developer(employeeId, name, hiringDate, email, weeklyCapHrs)

completeness: Developer(employeeId) ⊆ JuniorDeveloper(employeeId) ∪ MidDeveloper(employeeId) ∪ SeniorDeveloper(employeeId)
inclusion: Developer(employeeId) ⊆ HasSkill(developer)

JuniorDeveloper(employeeId)
foreign key: JuniorDeveloper(employeeId) ⊆ Developer(employeeId)
disjointness: JuniorDeveloper(employeeId) ∩ MidDeveloper(employeeId) = ∅
disjointness: JuniorDeveloper(employeeId) ∩ SeniorDeveloper(employeeId) = ∅

MidDeveloper(employeeId)
foreign key: MidDeveloper(employeeId) ⊆ Developer(employeeId)
disjointness: MidDeveloper(employeeId) ∩ SeniorDeveloper(employeeId) = ∅

SeniorDeveloper(employeeId)
foreign key: SeniorDeveloper(employeeId) ⊆ Developer(employeeId)

LeadDeveloper(employeeId)
foreign key: LeadDeveloper(employeeId) ⊆ SeniorDeveloper(employeeId)
inclusion: LeadDeveloper(employeeId) ⊆ TechLeads(leadDeveloper)

TimeLog(developer, task, startDate, startTime, endingDate, endingTime, timeWorkedHrs)
foreign key: TimeLog(developer) ⊆ Developer(employeeId)
foreign key: TimeLog(task) ⊆ Task(taskId)

DueDateChange(developer, task, dateChanged, oldDueDate, newDueDate, reason, timeWorkedUntilNowHrs*,currentEstimateHrs*)
foreign key: DueDateChange(developer) ⊆ Developer(employeeId)
foreign key: DueDateChange(task) ⊆ Task(taskId)

Skill(name)
inclusion: Skill(name) ⊆ HasSkill(skill)

Orders(project, customer)
foreign key: Orders(project) ⊆ Project(code)
foreign key: Orders(customer) ⊆ Customer(vatRegNo)

Estimates(task, developer, estimatedEffortHrs, estimationDate)
foreign key: Estimates(task) ⊆ Task(taskId)
foreign key: Estimates(developer) ⊆ Developer(employeeId)

Reviews(<u>developer</u>, <u>codeReviewTask</u>, result)
foreign key: Reviews(developer) ⊆ Developer(employeeId)
foreign key: Reviews(codeReviewTask) ⊆ CodeReviewTask(taskId)

HasSkill(<u>developer</u>, <u>skill</u>)
foreign key: HasSkill(developer) ⊆ Developer(employeeId)
foreign key: HasSkill(skill) ⊆ Skill(name)

TechLeads(<u>project</u>, leadDeveloper)
foreign key: TechLeads(project) ⊆ Project(code)
foreign key: TechLeads(leadDeveloper) ⊆ LeadDeveloper(employeeId)

External constraints

- **Project:** For each tuple $(c, d, startDate, endDate, s)$ in Project, $startDate \leq endDate$.
- **Milestone:** For each tuple $(p, code, startDate, endDate, cr)$ in Milestone, $startDate \leq endDate$.
- **Milestone:** For each tuple $(pCode, mCode, mStartDate, mEndDate, mCrit)$ in Milestone, there must exist a tuple $(pCode, pDesc, pStartDate, pEndDate, pStatus)$ in Project such that $pStartDate \leq mStartDate$ and $pEndDate \geq mEndDate$.
- **BugFixTask:** For each tuple $(t, impact)$ in BugFixTask, where $impact$ = "High", if there exists a tuple $(t, d, s, dd, dr, crd, cod, p, m, ad)$ in Task, then a tuple of (ad) in either MidDeveloper or SeniorDeveloper must exist.
- **FeatureTask:** For each tuple $(t, complexity)$ in FeatureTask where $complexity$ = "High", if there exists a tuple $(t, d, s, dd, dr, crd, cod, p, m, ad)$ in Task, then a tuple (ad) in SeniorDeveloper must exist.
- **Task:** For every Developer ad, there may only be at most 3 tuples $(t_1, d_1, s_1, dd_1, dr_1, crd_1, cod_1, p_1, m_1, ad)$, $(t_2, d_2, s_2, dd_2, dr_2, crd_2, cod_2, p_2, m_2, ad)$, and $(t_3, d_3, s_3, dd_3, dr_3, crd_3, cod_3, p_3, m_3, ad)$ in Task where where each attribute s_1, s_2, and s_3 are equal to "Authorized" or equal to "In progress".
- **TimeLog:** For each tuple $(d, t, sd, st, ed, et, twh)$ in TimeLog there must exist a tuple $(t, de, s, dd, dr, crd, cod, p, m, d)$ in Task.
- **TimeLog:** For each tuple $(d, t, sd, st, endingDate, et, twh)$ in TimeLog there must exist a tuple $(t, desc, status, dd, dr, cd, completionDate, pro, mil, dev)$ in Task such that $completionDate \geq endingDate$.
- **TimeLog:** For each tuple $(d, t, startDate, startTime, endingDate, endingTime, twh)$ in TimeLog, $startDate \leq endingDate$. If $endingDate = startDate$, then $startTime \leq endingTime$.
- **Reviews:** For each tuple (d, crt, r) in Reviews, there must exist a tuple $(d, crt, sd, st, ed, et, twh)$ in TimeLog.