

ID2207- Modern Methods in Software Engineering

Final project

Course Coordinator:

Mihhail Matskin

Course Assistant:

Shatha Jaradat

Óttar Guðmundsson and Örn Arnar Karlsson

Group 17

2017.16.1

1. A set of developed stories with time estimates.

	<i>Release planning 1</i>
<i>Login</i>	When the system starts the system should display an option to login. A registered user should be able to write in his credentials and login. <i>Time estimate: GUI 0.5 hours, logic 0.5 hours</i>
<i>Register</i>	Employees should be able to register into the system. <i>Time estimate: GUI 0.5 hours, logic 0.5 hours</i>
<i>HomePage</i>	After logging in the system should display overview of available actions each user can perform. <i>Time estimate: GUI 2 hours, logic 1 hours</i>
<i>View Events</i>	Users should be able to view events on their HomePage that are coming up and being worked on. <i>Time estimate: GUI 1 hours, logic 1 hours</i>
<i>Create EventRequest</i>	Customer Service and Senior Customer Service should be able to write down clients requests and submit them to the system to be worked on. <i>Time estimate: GUI 1 hours, logic 0.5 hours</i>
<i>SCS can view EventRequest</i>	Senior Customer Service should be able to view EventRequests and either reject or approve them and submit them on to the Financial Manager <i>Time estimate: GUI 1 hours, logic 0.5 hours</i>
<i>Financial feedback on EventRequest</i>	The Financial Manager should be able to write financial feedback on the event request and submit it further to the Administration Manager <i>Time estimate: GUI 0.5 hours, logic 0.5 hours</i>
<i>AM views event request</i>	The Administration Manager should be able to either reject or accept the event request. <i>Time estimate: GUI 0.5 hours, logic 0.5 hours</i>
<i>Create an Event for an EventRequest</i>	If an event request has been approved by all parties Senior Customer Service should be able to book a business meeting and create an event based on the event request. <i>Time estimate: GUI 0.5 hours, logic 0.5 hours</i>
<i>Create an OutsourceRequest</i>	The Production Manager and Service Department Managers should be able to send an outsource request to HR when he's finding staff for an event <i>Time estimate: GUI 1 hours, logic 1 hours</i>
<i>View</i>	HR should be able to view OutsourceRequests that are created and

<i>OutsourceRequest</i>	view the event that it is linked to. <i>Time estimate: GUI 0.5 hours, logic 0.5 hours</i>
<i>Respond to OutsourceRequest</i>	HR should have the option to either reject or accept an OutsourceRequest and send the decision back to the user who created the request. If he accepts he can add the details of the hire to the request. <i>Time estimate: GUI 0.5 hours, logic 0.5 hours</i>
<i>Create Task</i>	The Production Manager and Service Department Manager should be able to send tasks to subteams that are working on an event. Although, only to their corresponding subteams. <i>Time estimate: GUI 1 hours, logic 1/5 hours</i>
<i>Subteams view task</i>	The subteams should be able to ask for additional budget for a task they are working on. They can also accept the task which puts it into in progress. <i>Time estimate: GUI 1 hours, logic 0.5 hours</i>
<i>Respond to additional budget request on a task</i>	The Production Manager and Service Department Manager should be able to send tasks back to subteams that ask for additional budget with an adjusted budget. <i>Time estimate: GUI 1 hours, logic 0.5 hours</i>
<i>Closing a task</i>	Employees that are working on a task should be able to mark it finished when it is done. <i>Time estimate: GUI 1 hours, logic 0.5 hours</i>
<i>Create a FinancialRequest</i>	The Production Manager and Service Department Manager should have the option to send a FinancialRequest to the Financial Manager when subteams ask for additional budget on a task. <i>Time estimate: GUI 0.5 hours, logic 0.5 hours</i>
<i>View a FinancialRequest</i>	The Financial Manager should be able to view FinancialRequests created and view the corresponding task that it was created for. <i>Time estimate: GUI 0.5 hours, logic 0.5 hours</i>
<i>Respond to a FinancialRequest</i>	The Financial Manager should be able to reject, approve or approve the financial request. <i>Time estimate: GUI 1 hours, logic 0.5 hours</i>
<i>View an answered FinancialRequest</i>	The Production Manager and Service Department Manager should be able to see rejected or approved FinancialRequests that they have created. <i>Time estimate: GUI 0.5 hours, logic 0.5 hours</i>

	Release planning 2
<i>View all employees</i>	HR can view a list of all employees that are registered in the system and invalidate accounts. <i>Time estimate: GUI 1 hours, logic 0.5 hours</i>
<i>Create subteams</i>	Users with given authority, such as the Production Manager and Service Department Manager, can create subteams and assign employees to them. <i>Time estimate: GUI 0.5 hours, logic 1.5 hours</i>
<i>See available dates for first business meeting</i>	Gather all scheduled meetings for employees that should attend to the first business meeting with the client and mark the dates that are unavailable so there won't be a conflict. <i>Time estimate: GUI 1.5 hours, logic 1 hours</i>
<i>Assign tasks to individuals</i>	Allow a sub team leader to assign an accepted tasks to individuals in his team to coordinate tasks. <i>Time estimate: GUI 0.5 hours, logic 2 hours</i>
<i>View old event requests</i>	Users should be able to view EventRequests that have been confirmed and assigned a business meeting. <i>Time estimate: GUI 0.5 hours, logic 0.5 hours</i>
<i>Delete Requests</i>	Users with certain authority should be able to delete requests, such as outsource and financial, that have been assigned to certain events or tasks. <i>Time estimate: GUI 1 hours, logic 1 hours</i>
<i>Notifications</i>	Add a notification system to the event home page so users can instantly see task and requests that have been assigned to them. <i>Time estimate: GUI 2.5 hours, logic 3 hours</i>
<i>Add images to event</i>	Allow users to add images tagged under their subteam with description to events <i>Time estimate: GUI 1.5 hours, logic 1.5 hours</i>
<i>User Profile Page</i>	Users should have a home page that other users can view, there they can fill in basic contact information and add a picture of themselves. <i>Time estimate: GUI 3 hours, logic 2 hours</i>

2. A set of selected stories for the first release.
Explain your selection by considering importance and risk factors.

<i>User story</i>	<i>Value</i>	<i>Risk</i>
<i>Login</i>	Medium	Medium
<i>Register</i>	Medium	Medium
<i>HomePage</i>	Medium	High
<i>View Events</i>	High	Medium
<i>Create EventRequest</i>	High	High
<i>SCS views EventRequest</i>	Medium	Medium
<i>Financial feedback on EventRequest</i>	High	Medium
<i>AM views event request</i>	Medium	Medium
<i>Create an Event for an EventRequest</i>	High	High
<i>Create an OutsourceRequest</i>	High	Medium
<i>View OutsourceRequest</i>	Medium	Medium
<i>Respond to OutsourceRequest</i>	High	Medium
<i>Create Task</i>	High	High
<i>Subteams view task</i>	Medium	Medium

<i>Respond to additional budget request on a task</i>	High	Medium
<i>Closing a task</i>	Medium	Medium
<i>Create a FinancialRequest</i>	High	High
<i>View a FinancialRequest</i>	Medium	High
<i>Respond to a FinancialRequest</i>	Medium	High
<i>View an answered FinancialRequest</i>	Medium	High
<i>View all employees</i>	Low	Medium
<i>Create subteams</i>	Medium	Low
<i>See available dates for first business meeting</i>	Low	Low
<i>Assign task to individuals</i>	High	Low
<i>View old EventRequest</i>	Low	High
<i>Delete Requests</i>	Low	Low
<i>Notification</i>	Medium	Low
<i>Add image to event</i>	Low	Medium
<i>User Profile Page</i>	Low	Low

Total types of user stories

	High value	Medium value	Low value
<i>High risk</i>	4	4	1
<i>Medium risk</i>	5	7	2
<i>Low risk</i>	1	2	3

3. Iteration plan that is a list of which stories you implement in which iteration.

Iteration 1

Create EventRequest	SCS can manipulate EventRequest
Financial feedback on EventRequest	AM can view EventRequest
Create an Event for an EventRequest	

Iteration 2

Create OutsourceRequest	View OutsourceRequest
Respond to OutsourceRequest	HomePage
View Events	

Iteration 3

Create Task	Subteams can view Task
Respond to additional budget request on a Task	Closing a Task
Login	

Iteration 4

Create a FinancialRequest	View a FinancialRequest
Respond to a FinancialRequest	View an answered FinancialRequest
Register	

4. The metaphor.

The event planning system can be described in the metaphor:

The human body

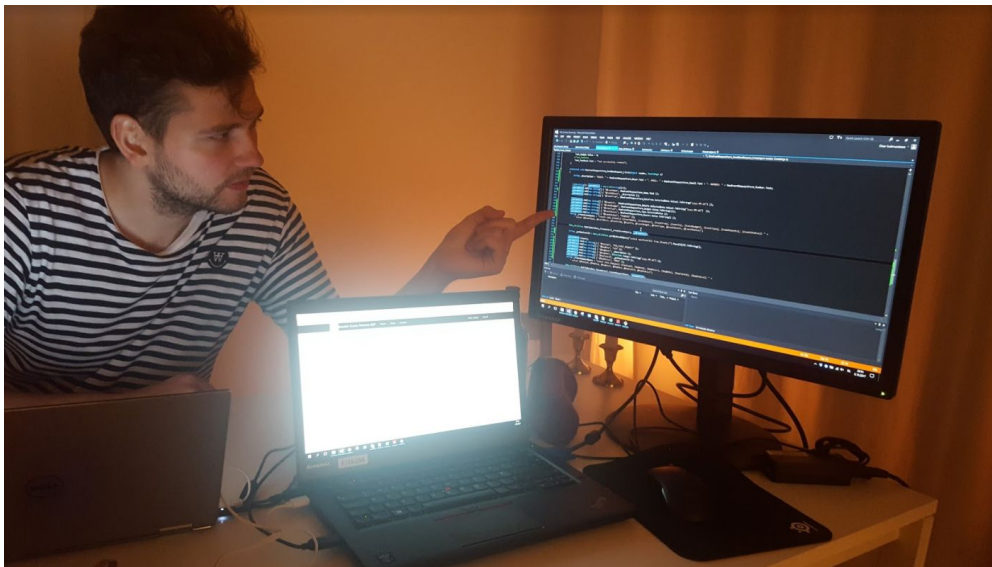
<i>Metaphor</i>	<i>System</i>
Food	The client's request
The act of eating	Taking down the client's requests and submitting them to the system.
Mouth	Customer service receiving the client's request and
The stomach	Senior Customer Service, who has power to reject or accept the request.
Blood	Product Manager and Service Department Manager who distribute the tasks according to the client's request
Nerve signals to the brain	OutsourceRequest and FinancialRequest
Brain	Financial Manager and HR who receive requests from other users.

5. Description of your test-driven pair programming process and applied refactoring.

Also describe how well you managed to estimate what should be done in each iteration. Write the truth! In order to pass you must show that you can draw conclusions of your mistakes, not that you have done everything perfectly.

Pair programming

Pair programming worked both efficiently for us but also slowed us down at some points in the development. The setup for the pair programming was simple. We had an extra monitor connected to our computers so every time we needed to develop a code that needed two eyes it was easy to take control of it so both of us could see what was happening.



We split the user stores and code into two parts - control schema and interface/ux design. Örn would take care of the backend and the flow of events while Óttar would take care of the latter. Designing the database, tables and foreign keys was done with pair programming so both of us would agree on the data structure and how tables could be synchronized together. While doing the interface design, Örn often sat through the development so we could get a second opinion if the buttons and textboxes were placed correctly. By doing that, we could agree on the final look so both of us could navigate through it.

Pair programming was also applied to the control schema and flow of events so Óttar could figure out what would happen when he called certain methods with given parameters.

We were slowed down in two cases, one of them was when certain buttons that should handle edge cases in our system were supposed to talk to the control schema. It seemed that we had misunderstood each other about when and who should update status of tasks as well when it should be closed or finished. So when we talked about it in real time, Óttar had a hard time understanding the logic that Örn was trying to explain. In hindsight, it would have been better that Örn and Óttar would just agree on what inputs he would like to accept in the control and trust him to take care of it rather than detailing the complex logic since he had already created the helper classes used for the process. If pair programming would have been used the whole time, this wouldn't have been the case but then we would be way behind schedule. The latter case was when we applied the pair programming to a certain part of the user interface when dealing with the Telerik library and its special components. Óttar had a great experience on the library but Örn had only read a little bit about it. So when Óttar was implementing some tokens in the AJAX engine to synchronize everything, Örn constantly had to ask Óttar about why he was doing what he was doing. By explaining every step, we lost valuable time.

Refactoring

The refactoring part really shined in every iteration since we improved our code a little bit in by either making it more readable or improving in performance wise. The biggest refactoring cases that are noteworthy might be the AJAX engine and multi functional buttons. Since we developed our code in ASP.NET, we started letting every action perform a postback to re-render the interface. When we were happy with the final looks and saw that the flow of events was correct we wrote the interface again with an AJAX engine to make it the page feel more alive and be more responsive. But by doing so, a lot of code in the backend had to be changed since some controls were hidden when the AJAX engine was trying to communicate with them.

The control schema also went through some refactoring in the midst of the development since we found out that there were some Tasks that needed to know who should receive next. Finally, we decided to simplify the user interface in the end by only having one button in certain forms that would change its text and methods called depending on the current state of the request/task. It looked cleaner to have one button instead of 3 or 4 buttons that would either be invisible or disabled. But that needed a lot of refactoring by combining some functions and logic and updating some parameters of the buttons.

Finally we made the code more human readable at parts e.g. creating a variable that was used in similar functions and using it as the parameter rather than referencing the object many times.

```
protected void ViewTask_Accept_Click(object sender, EventArgs e)
{
    Decimal _decInput = Convert.ToDecimal(ViewTask_ExtraBudget.Value);
    string _commentText = ViewTask_ExtraComment.Text;

    RadioButton _sender = (RadioButton)sender;
    if (_sender.CommandName == "FINANCIAL")
    {
        Insert financial request
        Data_Uilities.ModifyDataBase_Parameters(_createRequestQuery, _parameters);
        TaskControl.SubmitTask(Convert.ToInt32(_sender.Value), _decInput, _commentText);
    }
    if (_sender.CommandName == "CLOSE")
        TaskControl.SubmitTask(Convert.ToInt32(_sender.Value), _decInput, _commentText);
    if (_sender.CommandName == "PENDING")
        TaskControl.SubmitTask(Convert.ToInt32(_sender.Value), _decInput, _commentText);
    InterfaceByUser();
}
```

Estimation of iteration

Let's be honest here - we completely underestimated our planning of iteration and in what order they should be done. But in order to describe why we did so is a story that we learned from and will hopefully serve as an example in the near future.

First of all, the first iteration was a total success! We worked really well together when designing the database and building events and the views based on the event status and the request order. The events were complete, so viewing them was as simple as adding a double-click on our datagrid. Thus we also added View Events to that iteration because it took us such a short time. Since we were using ASP.NET the login and register was actually easier to implement than we thought before, so our initial iteration looked like this.

Iteration 1

Create EventRequest	SCS can manipulate EventRequest
Financial feedback on EventRequest	AM can view EventRequest
Create an Event for an EventRequest	Login
Register	View Events

As one can see, we were actually way ahead - for the time being.

For the next version we were going to create the Outsource request, but we had to ask ourselves if the Outsource request should be related to financial request, that is, would it make sense that the HR could ask for a financial request if he was hiring a really expensive photographer for example? We thought so but we didn't implement it yet. But by thinking so, we thought it would make more sense to create a financial request first because that might sync with both outsource request and the task request. After agreeing on that we suddenly realized that creating a financial request regarding task was actually more complicated than creating a financial request regarding the outsource request. Thus we changed our plan completely and switched most of the outsource request user stories in Iteration 2 with the user stories in iteration 3 and moved the financial requests to iteration 3. Thus our iteration looked like this

Iteration 2

Create Task	Subteams can view Task
Respond to additional budget request on a Task	HomePage

Now we were on schedule with the amount of users stories finished for each iteration but not in the correct order. Note that Closing a Task was incomplete since we still had to finish the Financial Request if the task had extra budget. Instead, we created the homepage so we could see who was responsible for all types of requests and tasks which helped us greatly. It was easier to see the data in the interface rather than looking constantly at the data. We even went a little further in the first iteration by adding simple functions like viewing the event from requests and task. This was not a user story which we thought of in the beginning but we thought it would be nice for the system. For the next iteration we needed to complete the financial request which was just as we planned on doing.

Iteration 3

Create a FinancialRequest	View a FinancialRequest
Respond to a FinancialRequest	View an answered FinancialRequest

Finally, when the Financial Request was finished we could successfully close tasks that were related to financial tasks and needed permission for extra budget. Adding the Outsource request included most of the functionality developed for the Financial Request so we could easily use the code Örn wrote to make it sync between responsibilities of users in the system. Thus our final iteration ended like this (not including tweaking little bugs around and finalizing the interface).

Iteration 4

Create OutsourceRequest	View OutsourceRequest
Respond to OutsourceRequest	Closing a Task

So as you, dear reader, can see is that by being able to change plans in the middle of development actually helped us reuse code After fixing it in the middle of the development, we clearly saw that this was a better way and we had a good reason to support it. We quickly adapted to this pattern which helped us deliver our final version of the event system. On time.

We left the rest of the user stories for the release planning 2 but we greatly believe that they will add a lot of personality, feeling and functionality to our system.

6. The source code

(including source code for Test Cases), and a readme.txt file that explains what is needed to compile and run the program.

All the source code can be found in the zip file as well as how to run the program in the readme.txt file.

7. Write acceptance tests for two of the user stories you implement.

You must choose stories that take input and give output through the user interface. Report this task by submitting a short description of the acceptance tests (including how they are started). The chosen acceptance test should address the main functionality of your system.

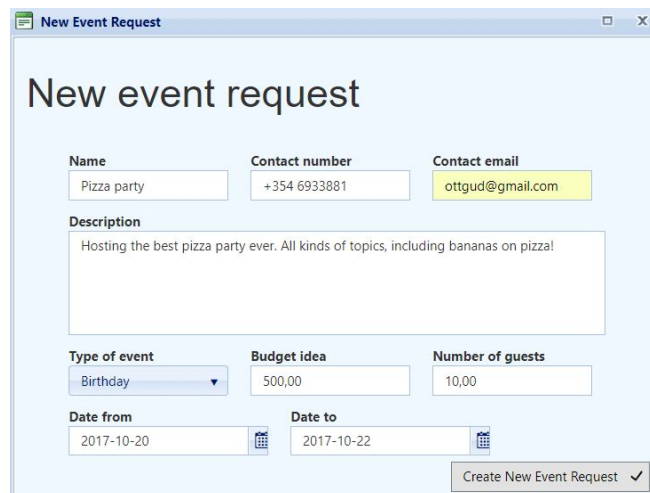
1. Book an Event

This test starts when either a *Senior Customer Service* or *Customer Service* is logged into the system. The user can click the “New Event Request” button down to the right, only visible to those users. He clicks that button.

A client called?

+ New Event Request

After doing so, a new form opens in a modal window for the user.



The screenshot shows a modal window titled "New Event Request". Inside, there is a form with the following fields:

- Name:** Text input with "Pizza party" entered.
- Contact number:** Text input with "+354 6933881" entered.
- Contact email:** Text input with "ottgud@gmail.com" entered.
- Description:** Text area with "Hosting the best pizza party ever. All kinds of topics, including bananas on pizza!" entered.
- Type of event:** Dropdown menu with "Birthday" selected.
- Budget idea:** Text input with "500,00" entered.
- Number of guests:** Text input with "10,00" entered.
- Date from:** Date picker with "2017-10-20" selected.
- Date to:** Date picker with "2017-10-22" selected.

At the bottom right of the form is a button labeled "Create New Event Request" with a checkmark icon.

The user should now fill in the desired information and click the “Create New Event Request” button. By doing so, the input clears and the user will be prompted with a “New Event Request Created” underneath the button. Now you the *Senior Customer Service* should be able to see a new event request at the bottom of the page.

Requests

Pizza party | EVENT | 2017.10.11

By clicking this button, an event request form is displayed where the *Senior Customer Service* has the option to reject the request or forward it. Note that the *Senior Customer Service* cannot fill in the information regarding financial information.

The user now clicks the “Accept and Forward” button and the request will now be sent to the *Financial Manager* of the system. He now sees the request at the bottom of the page and opens it by clicking the request. Then, a window opens where he can fill in the financial information. He only has the option of forwarding it to the *Administration Manager*. The *Financial Manager* fills in the feedback form and clicks the button.

When the *Administration Manager* has logged in, he can see his request at the bottom of the page and opens it by clicking it. He has the option of either rejecting it or accepting it, based on the financial data about the event request. He chooses to accept by clicking the “Accept and Forward” button.

Now the request will go back to Senior Customer Service where he can pick a date and time for the first business meeting to take place. The user should now click the request at the bottom of the page to open up the event.

Book a meeting!

Now the Senior Customer Service selects time and date and books the meeting by clicking the Book Meeting button. The request will now be closed and the event booking is complete. Now all users in the system can see event on the event home page.

29	Oktober	DESCR: fdfsfsafa. EMAIL: ottw. NUMBER: 455	2017-10-05 00:00:00
5	Lisa 18	Big spoiled brat hosting a shitty party	2017-10-11 00:00:00
36	Pizza party	DESCR: Hosting the best pizza party ever. All kinds of topics, including bananas on pizza! EMAIL: ottgud@gmail.com. NUMBER: +354 6933881	2017-10-20 00:00:00
		DESCR: Lefelet. EMAIL:	

2. Accept a Task with additional required budget

This test starts when either *Production Manager* or *Service Department Manager* is logged into the system For this this, we will be demonstrating the former one. *Production Manager* opens up an event by double-clicking an event on the home page. That will open a modal window of the event.

The screenshot shows a modal window titled "Event View". It contains the following sections:

- Event Details:**
 - Name: House and chairs 2019
 - Description: The most interesting summit so far
 - Date Start: 2018-05-28 00:00:00
 - Date End: 2018-05-31 00:00:00
 - Financial Comment: No sofas please
 - Budget: 4433276
 - Guests: 9000
 - Type of event: CONFERENCE
- Create a task:**
 - Name: (text input)
 - Description: (text input)
 - Assign a task to sub team: (dropdown menu)
 - Budget for task: (text input)
 - Do before: (calendar icon)
 - Create Task: (button)
- Create an outsource request:**
 - Outsource Name: (text input)
 - Outsource for: (dropdown menu)
 - Description: (text input)
 - Create Task: (button)

The user can now fill in information about a required task by filling in the information in the Create a task section. He fills in the information and selects a subteam. To create the task, he clicks the "Create Task" button. This task will now be displayed for the subteam. We will log in as a *Audio Specialist* to demonstrate.

When the *Audio Specialist* now logs into the system, he can see a new task assigned to him at the bottom left section.

Tasks

Get DJ mix | 2017.10.20 | PENDING

He clicks the task to open up a modal window with the info about it.

Task

Due date
Do before: 2017-10-20 (

Name	Description	Created by
Get DJ mix	Only Rock & Roll	JACK (2017-10-11 00:00:00)

Budget 1000 **Needed budget (if needed)** 50 000,00

Reason for needed budget
Rock n roll tracks are very expensive

Send task accept ✓

View event ➡

The *Audio Specialist* writes the needed budget for the task and comments on why he needs it. After doing so he can forward the task for acceptance by clicking the “Send task accept” button. The *Production Manager* now sees a task at the bottom left corner of his screen that is a pending financial request.

Tasks

Get DJ mix | 2017.10.20 | PENDING FINANCIAL
REQUEST

He clicks it to open it to see what the task will cost and the reason for it. He can edit the comment if he'd like to and create a financial request for the task. He creates a new financial request by clicking the “Send financial request” button.

Due date
Do before: 2017-10-20 (

Name	Description	Created by
Get DJ mix	Only Rock & Roll	JACK (2017-10-11 00:00:00)

Budget 1000 **Needed budget (if needed)** 50 000,00

Reason for needed budget
Rock n roll tracks are very expensive.
I guess he is right - he knows his stuff

Send financial request ✓

The *Financial Manager* now sees a new financial request at the bottom of his page.

Requests

Get DJ mix | FINANCE | 2017.10.11

He opens it up by clicking it. By doing so a financial request form opens.

Financial request

Regarding <i>House and chairs 2019</i>	View event
Name <i>Get DJ mix</i>	Department <i>AUDIO</i>
Description <i>Rock n roll tracks are very expensive. I guess he is right - he knows his stuff! Only Rock & Roll</i>	
Needed budget 50 000,00	
Reject	Accept

The *Financial Manager* now has the option to either reject it or approve it. He chooses to approve the extra budget by clicking the “Accept” button. The task will now be sent back to *Audio specialist* and he will know that he is working on it since it says that it is In progress.

Tasks

Get DJ mix | 2017.10.20 | IN PROGRESS

8. Report of daily stand-up meetings! (2 or 3 is okay)

We had standup meetings most days, even though we did not plan on meeting each other or do anything over the day. It helped us keep track of the project status and what we needed to do in the upcoming days.

Date	2017.10.4
What did you accomplish since the last meeting?	<p>Örn Installed SQL development environment, got the code from Github and ran the basic version of the project. It runs!</p> <p>Óttar Created the data tables, declared the connection string and inserted the dummy data. All done through the in built visual studio components.</p>
What are you working on until the next meeting?	<p>Örn Read about the Telerik API, see if I can create a button and something more by using the code. Start by creating a simple C# class that is static and can be accessed in the code through the namespace functionality of Visual studio.</p> <p>Óttar See if the connection string works when creating Tasks and Requests in the tables created as well as updating values. Changing controls from regular ASP controls to TELERIK and see if the bin files and assemblies are correctly loaded with the skins.</p>
What is getting in your way or keeping you from doing your job?	<p>Örn Other projects in other courses.</p> <p>Óttar The same. We will start full force next Friday when we have both turned in our other projects.</p>

Date	2017.10.7
What did you accomplish since the last meeting?	<p><i>Örn</i> A lot - nearly finished with the EventControl class and we can now run the event through all users in the system that should see it in the correct order.</p> <p><i>Óttar</i> Created the forms for requests and the Event itself. All buttons and all data that should be in each form now works and loads properly.</p>
What are you working on until the next meeting?	<p><i>Örn</i> Just have to set the status of the Event as 'Accepted' so it will be displayed in the Event table on the front page. Then I'm heading on to the Task Controller so users can assign tasks to subteams.</p> <p><i>Óttar</i> Fixing the CSS code in the forms to make it look a little bit more authentic and nice. Then I have to start working on the task form in each event.</p>
What is getting in your way or keeping you from doing your job?	<p><i>Örn</i> Figuring out where to save who is responsible for the task. We must always keep track of who created it, who it belongs to and who has access to it at certain points. Maybe we will need an extra column in the database so I might have to modify it first.</p> <p><i>Óttar</i> There is a little bit of problem related to the requestTaskId column in the Request database since it refers to tasks or events in some cases. I'm databinding it by its type but that can sometimes be problematic since my code is not clear enough. Got to fix that so correct name loads on requests if it is related to a task or an event.</p>

Date	2017.10.11
What did you accomplish since the last meeting?	<p><i>Örn</i> Finished all of the control schemes and finished the outsource request control so it works as it is supposed to. Also added a new function to set the tasks as rejected when they had extra budget needed and the financial manager rejected them.</p> <p><i>Óttar</i> Created a simple log in buttons in the log in site so we could test our systems faster by switching users without typing name and password. The Outsource request is also finished and does not display correctly in the event itself.</p>
What are you working on until the next meeting?	<p><i>Örn</i> Just need to fix the Tasks so when the one who is responsible has marked it as finished it should be finished and not be displayed anymore. Then start on writing the final report and evaluation on pair programming on refactoring.</p> <p><i>Óttar</i> Fixing the final look and debugging the rest of system to make sure that the first deployment does take care of all the assignments. Also make sure that the interface really updates using the AJAX function when needed.</p>
What is getting in your way or keeping you from doing your job?	<p><i>Örn</i> Waiting for Óttar for pushing his last change of the code to GitHub so we can test out Task complete function.</p> <p><i>Óttar</i> Some problems with the AJAX engine seem to occur when the control is not rendered in C# and a user that has no access to the control tries to update it. Need to switch control visibility from [Control].Display = false; to [Control].Style.Add("display","none"); so it does not interfere with the engine. Later on, if this was a real project for a real company, the functions that accesses those controls would be bound by different functions</p>

9. A comparison between this approach and the object oriented analysis and design approach that you followed in the previous three assignments (your feedback).

We both agree that we like and dislike different parts of both methods. In the beginning, we liked the idea of getting faster into the coding after developing the user stories and minimal planning, compared to object oriented analysis. But we found out in the middle of the development that some things were not planned out as much as we wanted to, like the database and how we would create objects for the client from what data. Restrictions on how we did things were less than using the design approach and gave our imagination more responsibility when we needed to find solutions on the spot to make the code sync and work correctly. Sometimes these solutions were brilliant, but sometimes they simply “worked” but the code was nothing to be really proud of. The mini stand up meetings also helped us keep track of the current status of the project, what was needed at each time and what we wanted to accomplish in every session.

On the other hand, the object oriented analysis might have been a more suitable to begin with since we could start off with planning relations between classes and databases to make sure beforehand that everything worked out the way we would like to. That method requires a lot of time and we could never have finished this project using that method. Some parts and solutions were not thought of at the planning stage, which only came when we were developing such as the button to open up the event when you were viewing the related request or task. By planning too much we felt that our creativity and simplicity got toned down because we would otherwise be following a strict pattern.

We would like to find a more suitable method which can be a mix of both, such as excessive planning in the beginning to map most of the things out and then dive head on into the code.