Software Testing - MovieBasen

København Erhvervsakademi – Software Team 9 Fall 2016

Elvis, Mert, Benyam and Casper

12/21/16

Software Development

Table of Contents

Introduction *- (Mert, Elvis)*

In this exam project we have decided to develop a movie web-based application similar to that of IMDB.com, where we got our inspiration. Our application name is MovieBasen where it will be possible for users to look movies up, add movies to your profile list and review/rate them[1].

The application is developed in ASP.Net MVC Framework with C#, and to help ourselves develop the system, we are going to use a software development process. With that said, our main focus in this project is software testing, so that is where we have put most of our effort into. That's why our aim in this project, is not to build an end product, but to build core functionality so we have enough materiel to do the different tests.
We will throughout the project document, evaluate and analyze the development, that way you can see the consideration we have done.

To see MoveBasen you can visit our site: http://moviebasen.azurewebsites.net/

[1] Same introduction is used for our database project

FURPS+ - *(Elvis, Mert)*

In this project, we wanted to use FURPS to find the specifications/requirements for the MovieBasen system.[2]

Functionality:
A user should be able to log into the system with their username and password. They will have the option to make one if they don't have any.
Inside the system, a user should be able to search for a specific movie by the name, or by genre. The user should also be able to rate the movie (with stars or points), review it and should be able to add the movie to his/hers 'watched list'.

An admin also has the same options as the normal user; furthermore, the admin should be able to make CRUD (*Create, Read, Update, and Delete*) functionality to movies, genres and actors. The admin should also be able to add actor and genre to the movie. The administrator will also be able to add other users to admin-roles. The administrator should also be able to delete users that is violating the system laws/agreements.

Usability:
We are planning to make the design clean and simple and to achieve that, we are going to use Mandel's golden rules. Mandel's golden rules is about 1. Place the user in control, 2. Reduce user's memory load, 3. Make the interface consistent. By doing this, the user is going to feel that he is in control when he is working with the system. In addition, we intend to make
understandable error messages, so that the user can get help, if he should fail action.

Our GUI will be a little different based on who is using it (user/admin) because the admin will have a few more possibilities and views. The GUI of the system will be designed; having in mind that it is going to be a web-based application and the interface is going to be in English.

Reliability:
The system should be able to run stable without any bigger problems and errors on runtime. The user will get an error message, if the user makes an illegal action.
If the system fails/shut down the restart time should be minimal, where the system should be up and running again.

Performance:
The application should have a fast response time, where several users should be able to use it at the same time without any problems. There shouldn't be any overloads in the system, or crashes.

- Response time from the system should happen within 4 sec.
- 200 concurrent users should be able to use the system per minute
- Recovery time from backup should be 30 min

Supportability:
The application should be easy to maintain and update if there is going to be changes. The system shall support use of the 3 most used browsers', Internet explorer, Firefox and Chrome.

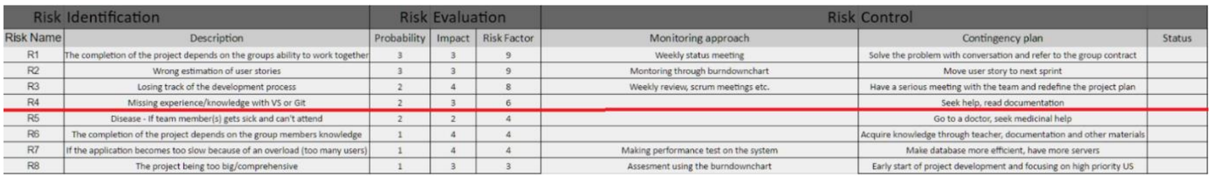
[2] Same FURPS+ is also used in our database report

Risk Management *- (Elvis, Mert)*

Risk management is an analysis you make in connection with a project. Each project contains risk and depending on how well you handle the various risks, the project may end up being a success or failure. There are two different methods that you can use to manage risks. There is the reactive and the proactive approach. When taking the reactive method in use, you take things as they come, which means that you solve the problems when you bump into them. When using the proactive method, you identify all the potential risk before starting the project
We have chosen to use the proactive approach, so we will identify all potential risks in the project by making a Risk Table, that way we can find out which risk is the most critical and requires monitoring.

We have chosen to use the numbers 1 - 5 to indicate the impact of a risk in our project and the probability. (*5= catastrophic, 4= critical, 3= high, 2 = medium og 1= low)*

We find risk exposure (RF) by multiplying the probability of that happening with the impact it has on my project. (RF = Probability * Impact).



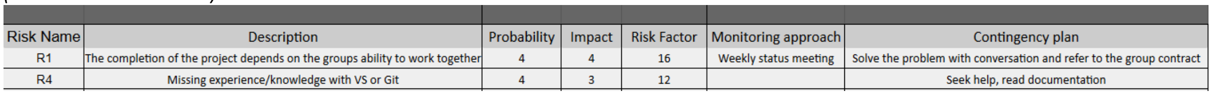
Risk Identification		Risk Evaluation			Risk Control		
Risk Name	Description	Probability	Impact	Risk Factor	Monitoring approach	Contingency plan	Status
R1	The completion of the project depends on the groups ability to work together	3	3	9	Weekly status meeting	Solve the problem with conversation and refer to the group contract	
R2	Wrong estimation of user stories	3	3	9	Montoring through burndownchart	Move user story to next sprint	
R3	Losing track of the development process	2	4	8	Weekly review, scrum meetings etc.	Have a serious meeting with the team and redefine the project plan	
R4	Missing experience/knowledge with VS or Git	2	3	6		Seek help, read documentation	
R5	Disease - If team member(s) gets sick and can't attend	2	2	4		Go to a doctor, seek medicinal help	
R6	The completion of the project depends on the group members knowledge	1	4	4		Acquire knowledge through teacher, documentation and other materials	
R7	If the application becomes too slow because of an overload (too many users)	1	4	4	Making performance test on the system	Make database more efficient, have more servers	
R8	The project being too big/comprehensive	1	3	3	Assesment using the burndownchart	Early start of project development and focusing on high priority US	

(Zoom in to see it better)

We have put the "cut off line" here, since we do not have the resources to monitor all risks, therefore will focus on the risks that is above the line. All risks that are below the line is low prioritized.

Risks updated
Later on, we found out that some of the probabilities and impact needed to be higher than we initially thought, so we updated the risk table after sprint 1.

(Zoom in to see it better)



Risk Name	Description	Probability	Impact	Risk Factor	Monitoring approach	Contingency plan
R1	The completion of the project depends on the groups ability to work together	4	4	16	Weekly status meeting	Solve the problem with conversation and refer to the group contract
R4	Missing experience/knowledge with VS or Git	4	3	12		Seek help, read documentation

The changed risk was R1 and R4. In R1 we changed the probability and impact to 4, when some of our team members didn't deliver to the deadlines and we generally had hard time working together. We realized that this risk was underestimated and therefore needed to change.
In R4 we changed the risk probability to 4, because we constantly had problems with Git *(f.eks our merge conflicts and sometime the connection to the database was lost after a push).* And therefore, we needed to give it more attention.

User Stories *- (All)*

User Stories are short simple descriptions of a feature told from the perspective of the person who wants the new capacity/requirements. User Stories are often written on cards or sticky notes, and arranged up on walls or tables to facilitate planning and discussion. When doing user stories it is important to switch focus from writing about the functions, to discussing them. In fact, these discussions/reflections are much more important than what is written in the text. It is also important to prioritize and estimate the User Stories, and the more experience you have, the more the estimations are accurate.

We have prioritized our User Stories: **1** priority, **2** priority, **3** priority, because that way we are sure to have implemented the most important/critical parts of the system first. We have also put our User stores in Time Boxes 1, 2, 4, 8, 12 hours, etc.

Administrator:

US 01 - As an administrator, I want to be able to create/delete/update the movie so that it is in the system *(Total 8hr)*. Prioritet 1

- Task 1 - Make CRUD*(Create, Read, Update and Delete)* functionality to movie*(2*hr*)*.
- Task 2 - Upload and modify the movies cover image (*6*hr)

US 02 - As an administrator, I want to be able to create/delete/update genres to the movie, so that you can see which genre the movie is *(Total 8*hr*)*. Prioritet 1

- Task 1 - Make CRUD*(Create, Read, Update and Delete)* functionality to genre *(2hr)*.
- Task 2 - Assign genres to a movie (*6*hr)

US 03 - As an administrator, I want to be able to create/delete/update actors to the movie, so you can see which actors is in the movie *(Total 8hr)*. Prioritet 2

- Task 1 - Make CRUD*(Create, Read, Update and Delete)* functionality to actor *(2hr)*.
- Task 2 - Assign actors to a movie (*6*hr)

US 04 - As an administrator, I want to be able to delete a user that is in the system, so he can't use login anymore. *(Total 6hr)*. Prioritet 3

- Task 1 - Make delete functionality to remove a user*(6hr)*.

US 05 - As a administrator I want to be able to register others in administrator roles so that they can administrate the system. *(Total 8hr)*. Prioritet 3

- Task 1 - Expand existing user account *(4hr)*.
- Task 2 - Upload and modify the administrator’s profile picture *(4hr)*.

User:

US 06 - As a user I want to be able to register myself in order to use the system.
(Total 8hr). Prioritet 1

- Task 1 - Expand existing user account *(4hr)*.
- Task 2 - Upload and modify the user's profile picture *(4hr)*

US 07 - As a user I want to be able to look under genres in order to find the movie i want to watch *(Total 16hr)*. Prioritet 2

- Task 1 - Make a page where all the movie are shown from the selected genre*(16hr).*

US 08 - As a user I want to be able to search for a specific movie in order to find the movie i want to add/watch. *(Total 8hr)*. Prioritet 2

- Task 1 - Make a search bar where it finds movies via name *(8hr).*

US 09 - As a user I want to be able to add the movie to ‘My List’ so that you can see all the movies you have watched, reviewed and rated. *(Total 8hr)*. Prioritet 2

- Task 1 - Assign a movie to a user account *(4hr).*
- Task 2 - Make a page, where he can see all the movie he has added as ‘watched’ *(4hr).*

US 10 - As a user I want to be able to rate the movie to express my opinion. *(Total 8hr)*. Prioritet 3

- Task 1 - Make CRUD*(Create, Read, Update and Delete)* functionality to rating*(2hr)*.
- Task 2 - Assign your rating to a movie *(6hr).*

US 11 - As a user i want to be able to review the movie so that other people can see my opinion of the movie *(Total 8 hr)*. Prioritet 3
• Task 1 - Make CRUD*(Create, Read, Update and Delete)* functionality to review *(2hr)*. • Task 2 - Assign review to a movie and a user *(6hr).*

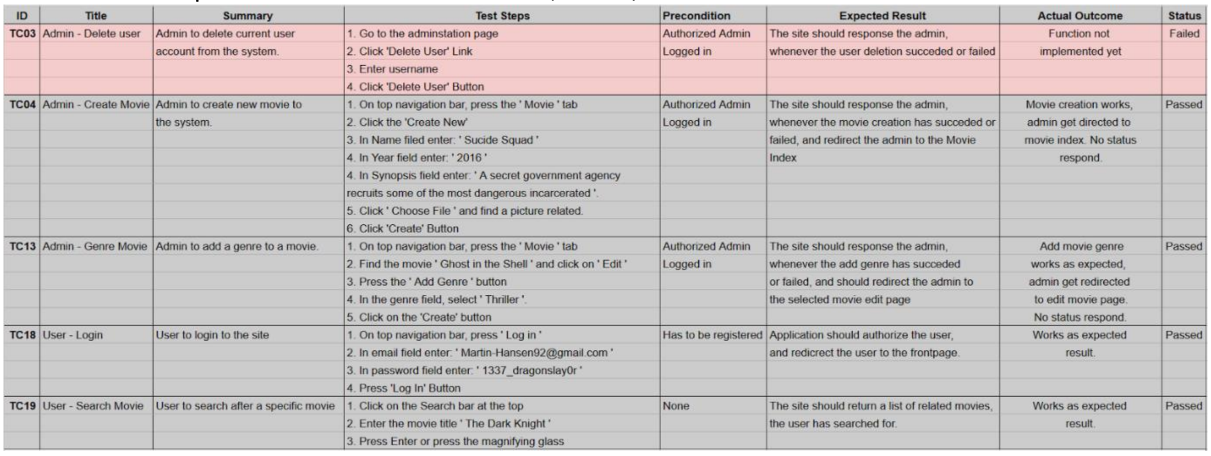
All:

US 12 - As a user and administrator I want to be able to Login / logoff in order to use the system *(Total 2hr)*. Prioritet 2
• Task 1 - Create login functionality (*2hr)*

Test Cases *- (Casper)*

A test case is a way to determine and verify if a particular feature or functionality works as intended on the system. It can also help us to validate whether our system is free of bugs and works as required by the end user.
We formed the test cases based on the user stories we made earlier, so that all the possible test scenarios were covered.

Here is an example of some of our test cases: *(zoom in)*



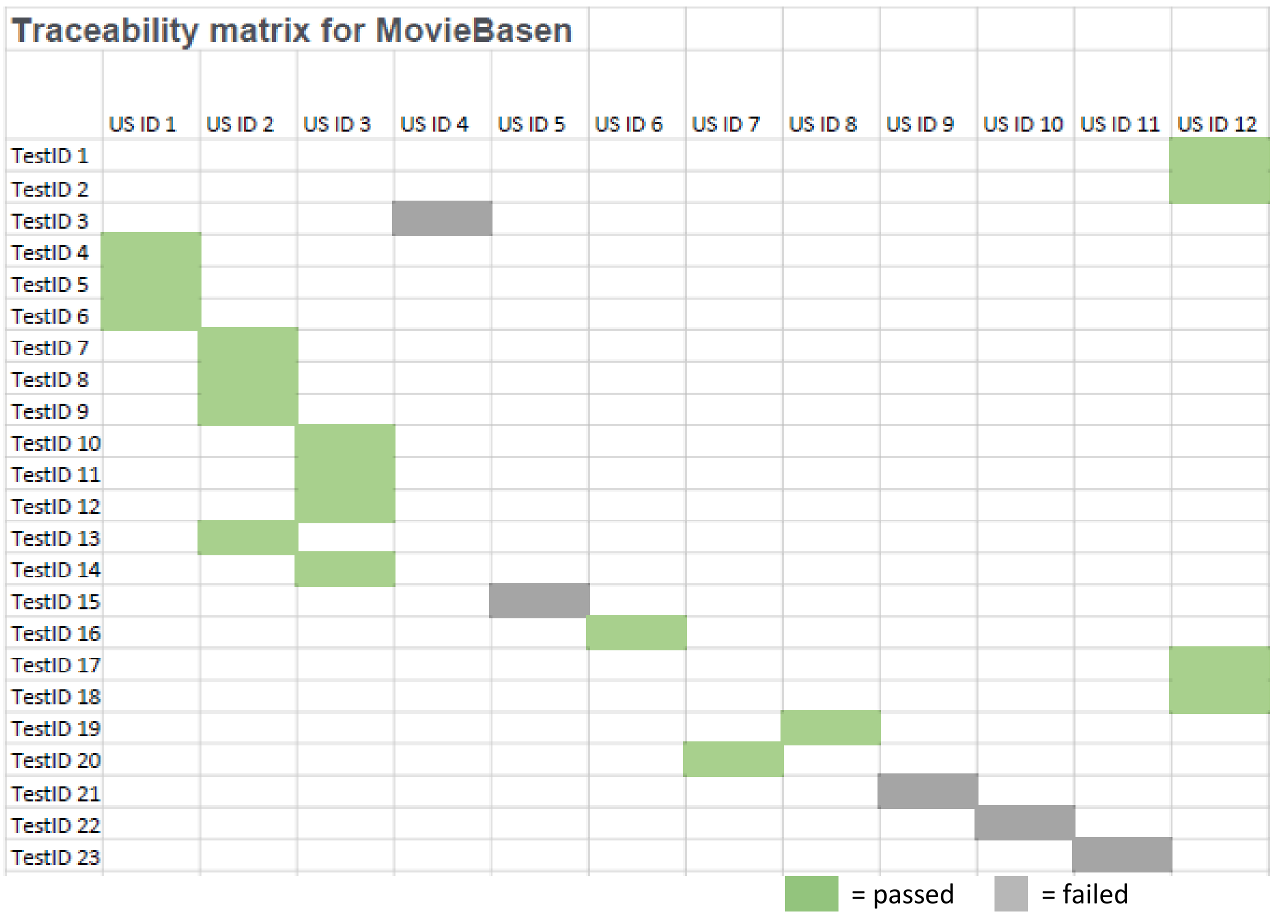
ID	Title	Summary	Test Steps	Precondition	Expected Result	Actual Outcome	Status
TC03	Admin - Delete user	Admin to delete current user	1. Go to the adminstation page	Authorized Admin	The site should response the admin,	Function not	Failed
		account from the system.	2. Click 'Delete User' Link	Logged in	whenever the user deletion succeded or failed	implemented yet	
			3. Enter username				
			4. Click 'Delete User' Button				
TC04	Admin - Create Movie	Admin to create new movie to	1. On top navigation bar, press the ' Movie ' tab	Authorized Admin	The site should response the admin,	Movie creation works,	Passed
		the system.	2. Click the 'Create New'	Logged in	whenever the movie creation has succeded or	admin get directed to	
			3. In Name filed enter: ' Sucide Squad '		failed, and redirect the admin to the Movie	movie index. No status	
			4. In Year field enter: ' 2016 '		Index	respond.	
			4. In Synopsis field enter: ' A secret government agency				
			recruits some of the most dangerous incarcerated '.				
			5. Click ' Choose File ' and find a picture related.				
			6. Click 'Create' Button				
TC13	Admin - Genre Movie	Admin to add a genre to a movie.	1. On top navigation bar, press the ' Movie ' tab	Authorized Admin	The site should response the admin,	Add movie genre	Passed
			2. Find the movie ' Ghost in the Shell ' and click on ' Edit '	Logged in	whenever the add genre has succeded	works as expected,	
			3. Press the ' Add Genre ' button		or failed, and should redirect the admin to	admin get redirected	
			4. In the genre field, select ' Thriller '.		the selected movie edit page	to edit movie page.	
			5. Click on the 'Create' button			No status respond.	
TC18	User - Login	User to login to the site	1. On top navigation bar, press ' Log in '	Has to be registered	Application should authorize the user,	Works as expected	Passed
			2. In email field enter: ' Martin-Hansen92@gmail.com '		and redicrect the user to the frontpage.	result.	
			3. In password field enter: ' 1337_dragonslay0r '				
			4. Press 'Log In' Button				
TC19	User - Search Movie	User to search after a specific movie	1. Click on the Search bar at the top	None	The site should return a list of related movies,	Works as expected	Passed
			2. Enter the movie title ' The Dark Knight '		the user has searched for.	result.	
			3. Press Enter or press the magnifying glass				

The full list of our test cases can be seen in the attached appendix.pdf "Test Cases (page 12-13)"

Examination of a test case;
In one of our test case *(TC18)* we see when a user logs in our system. We can see that the test case precondition is that the user has to be registered in the system. And when he tries to sign in we'll expect the system should authorize and redirect the user to the home page. After we went through the test steps, the actual final outcome was as expected, so therefore the status for the test case is passed.

Traceability Matrix *- (Casper, Elvis)*

A traceability matrix is usually in the form of a table used to find the link/relation between the requirements *(in our case user stories)* and test cases.
By making one you'll generally get a better overview of your functional requirements and which user stories you'll need to test, since in most projects you don't have time to test everything *(so f.eks you can say that you want to test all the user stories, which have a priority of 1).* You'll also be able to easier track the test cases of the requirements that have been changed *(by doing so, you can change those test cases)*.



Traceability matrix for MovieBasen

	US ID 1	US ID 2	US ID 3	US ID 4	US ID 5	US ID 6	US ID 7	US ID 8	US ID 9	US ID 10	US ID 11	US ID 12
TestID 1												🟩
TestID 2												🟩
TestID 3				▨								
TestID 4	🟩											
TestID 5	🟩											
TestID 6	🟩											
TestID 7		🟩										
TestID 8		🟩										
TestID 9		🟩										
TestID 10			🟩									
TestID 11			🟩									
TestID 12			🟩									
TestID 13		🟩										
TestID 14			🟩									
TestID 15					▨							
TestID 16						🟩						
TestID 17												🟩
TestID 18												🟩
TestID 19								🟩				
TestID 20							🟩					
TestID 21									▨			
TestID 22										▨		
TestID 23											▨	

🟩 = passed ▨ = failed

SCRUM - *(Elvis)*

Scrum is a well-known agile development method that you use to build systems. We chose to use it in this project, because it is very flexible and good at showing us how things are going with the big picture. Moreover, it is an agile development method we all have experience working with.

We have put all our user stories/tasks in an online program called Trello. In that way, it was easier to maintain and have access to the Product Backlog and Sprint Backlog.

Each sprint we planned was going to be different, because when we started the project, we still had classes so our schedule was chaotic. Moreover, people also got jobs to attain to and other stuff that made it impossible for us to work consistently each sprint.

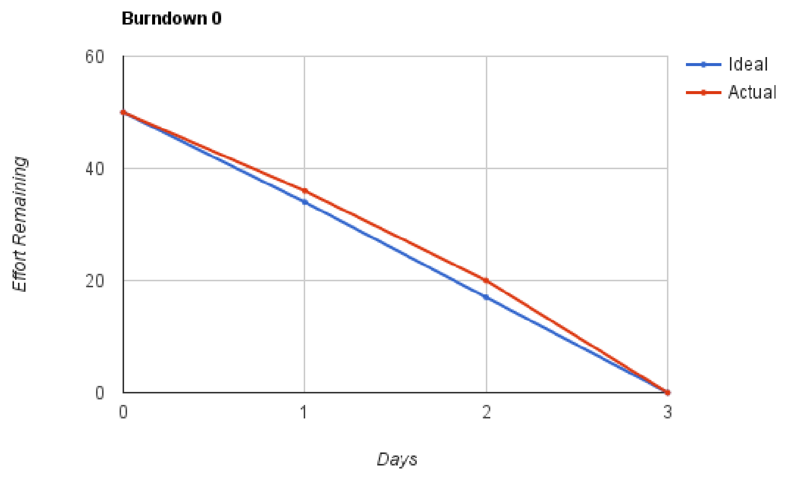
Before we began planning the first sprint, we found out that we would work **50 hours** a week each sprint, that way we know how many user stories we can put in each Sprint.

Sprint 0

We have planned to complete
Plan:

- Project formulering *(6t)*.
- FURPS+ (8t).
- User Stories *(16t)*.
- Test Cases (10t).
- Traceability Matrix (2t).
- Risk Management *(8t)*.

Evaluation
In the first sprint we planned, that we were going to work **50 hours** and it would last for two weeks. Many of the tasks we needed to do were basic things that we needed to figure out, before we could start coding. Overall, the first sprint went well and we managed to make everything on time.



Sprint 1

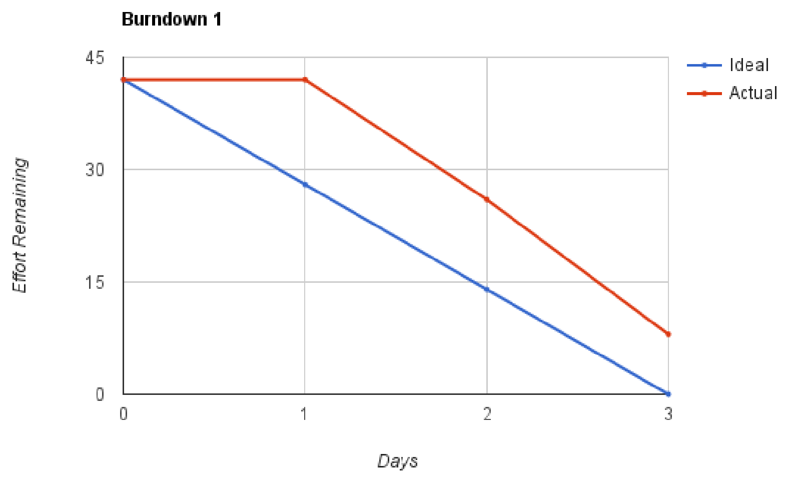
We have planned to complete

Plan:

- US 01 *(8t)*
- US 02 *(8t)*
- US 03 *(8t)*
- US 06 *(8t)*
- Test Case Actual Outcome - for the completed user stories (2t).
- Unit Test (8t)

Evaluation

This sprint was one week long and we planned to work **42 hours**. We started by connecting the database to an online server (Azure), which we thought was going to take 10 min, but it did not and instead end up taking much more time. In addition, we didn't include it as a task, so because of that we ended being behind schedule *(Ideal Work line)* through the whole sprint. Moreover, we had some problems with Git, and we also had some conflicts in the group, which we needed to fix. Because of these problems, we realized that some of our risks were greatly underestimated and therefore need to change.
In this sprint the only thing we did not finish, was US 6.

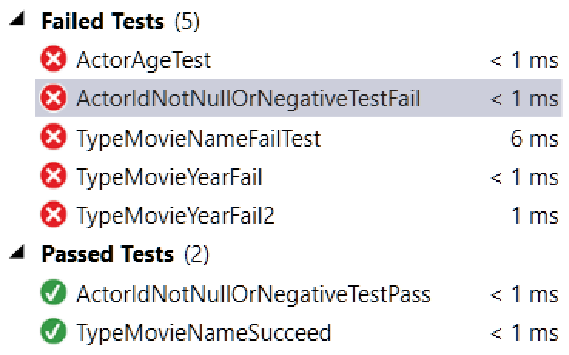


Unit Test - *(Mert)*

Unit test in software testing is the part where you focus on the single unit, within the module. The purpose of the unit test is to ensure that the unit meets its specification, before integrating with other units. Usually the programmer codes the unit tests and often he can use tools/frameworks that is supported; such as Junit, FlexUnit and many more. In our case, we used the unit test framework in Visual Studio.

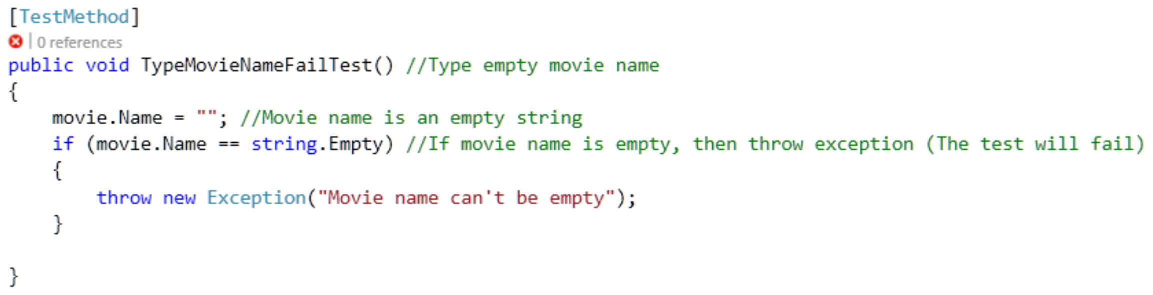
Our intention with making unit test for our project was not to test everything or make unit test for every function *(unit)*, but to test the most core and important functions. That way we will get fewer errors.

The unit tests we coded, was for creating a movie and an actor in the MovieBasen and this is the result of the different unit test.



```
◢ Failed Tests (5)
    ✗ ActorAgeTest                          < 1 ms
    ✗ ActorIdNotNullOrNegativeTestFail      < 1 ms
    ✗ TypeMovieNameFailTest                   6 ms
    ✗ TypeMovieYearFail                     < 1 ms
    ✗ TypeMovieYearFail2                      1 ms
◢ Passed Tests (2)
    ✓ ActorIdNotNullOrNegativeTestPass      < 1 ms
    ✓ TypeMovieNameSucceed                  < 1 ms
```

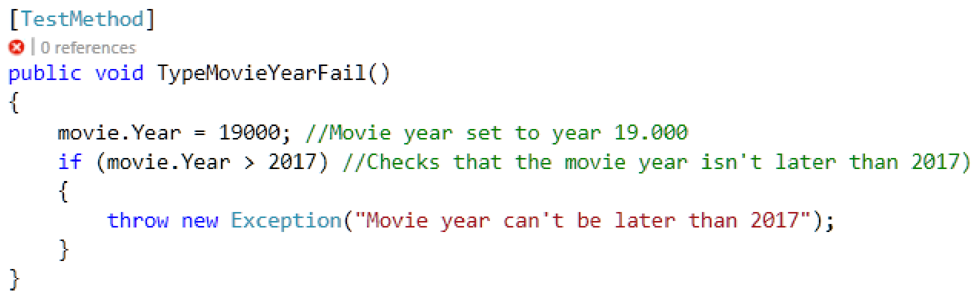
Our first unit test method;



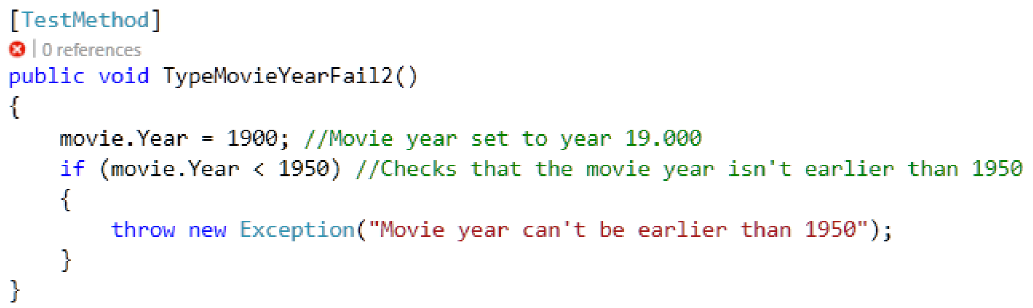
```
[TestMethod]
✗ | 0 references
public void TypeMovieNameFailTest() //Type empty movie name
{
    movie.Name = ""; //Movie name is an empty string
    if (movie.Name == string.Empty) //If movie name is empty, then throw exception (The test will fail)
    {
        throw new Exception("Movie name can't be empty");
    }

}
```

The test above, tests when you type in a movie name, for instance if you type an empty name when creating the movie, you will expect an error or a warning.
We have set the movie name to be an empty string (""), if the condition for the movie name is true, then we will throw an exception and the test will fail.

Two of the other unit tests we made was to test that the movie year couldn't be later than 2017 or earlier than 1950.



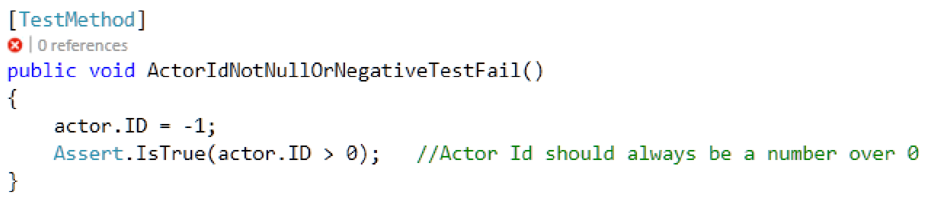
```
[TestMethod]
| 0 references
public void TypeMovieYearFail()
{
    movie.Year = 19000; //Movie year set to year 19.000
    if (movie.Year > 2017) //Checks that the movie year isn't later than 2017)
    {
        throw new Exception("Movie year can't be later than 2017");
    }
}
```



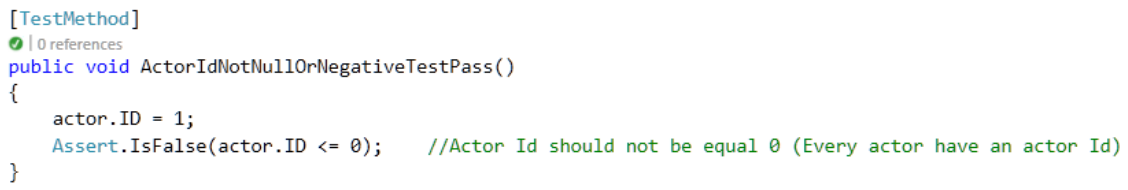
```
[TestMethod]
| 0 references
public void TypeMovieYearFail2()
{
    movie.Year = 1900; //Movie year set to year 19.000
    if (movie.Year < 1950) //Checks that the movie year isn't earlier than 1950
    {
        throw new Exception("Movie year can't be earlier than 1950");
    }
}
```

In one of the if statement we check if the number is bigger than 2017, in the other we check if it's lower than 1950, if any of them should fail it will throw an exception.

In these two unit test methods we check the actor id, and that it should not be 0 or under (negative number).



```
[TestMethod]
| 0 references
public void ActorIdNotNullOrNegativeTestFail()
{
    actor.ID = -1;
    Assert.IsTrue(actor.ID > 0);    //Actor Id should always be a number over 0
}
```



```
[TestMethod]
| 0 references
public void ActorIdNotNullOrNegativeTestPass()
{
    actor.ID = 1;
    Assert.IsFalse(actor.ID <= 0);    //Actor Id should not be equal 0 (Every actor have an actor Id)
}
```

With the first method, we have set the actor id to be -1, and the test result is that it will fail, because an actor will never have a negative or a 0 id. The second test method will pass, because it is okay that the actor id is any number over 0.

These are some of the unit test methods we have made, if you want to see more you can find them in the MovieBasenTestProject.

Sprint 2

We have planned to complete

Plan:

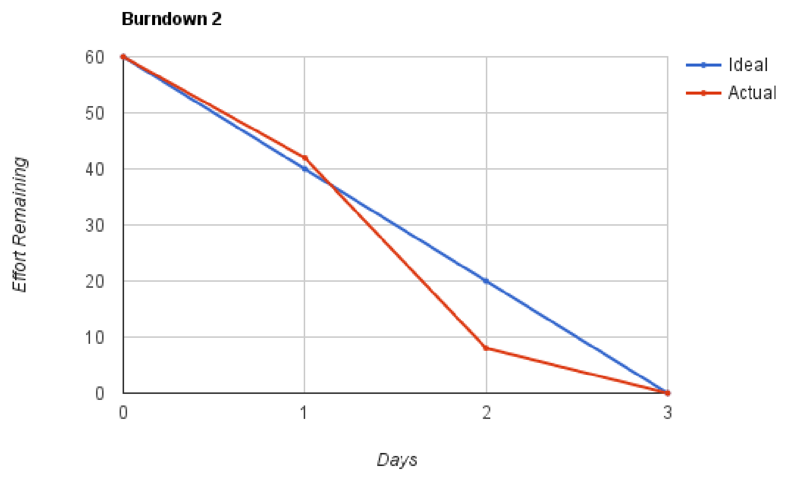
- Update Risk Management Table (2t)
- Finish US 06 *(4t)*
- US 08 *(8t)*
- US 12 *(2t)*
- Test Case Actual Outcome - for the completed user stories (2t).
- Integration test (10t)
- Usability Test (8t)
- Performance test (16t)
- Acceptance Test (8t)

Evaluation

In our last sprint we concluded that it would be 1 week long and we planned to work **60 hours**. Since we didn't complete US 6 from last sprint, we estimated that it would take 4 more hours to finish it. We also had underestimated some of our risk, so we needed to update the risk table.

This week we didn't have any more classes and because of that we had more time to work together as a group, which I think made, us work more productively and therefore we managed to finish everything in sprint 2 on time.

As you can see, we have chosen not to do more sprints and therefore agreed that this would be our last one, and because of that, we would not code any more functionalities/US. We did that, because we thought that we had more than enough material to do all the tests that was required. We also needed time to write a report to this project.

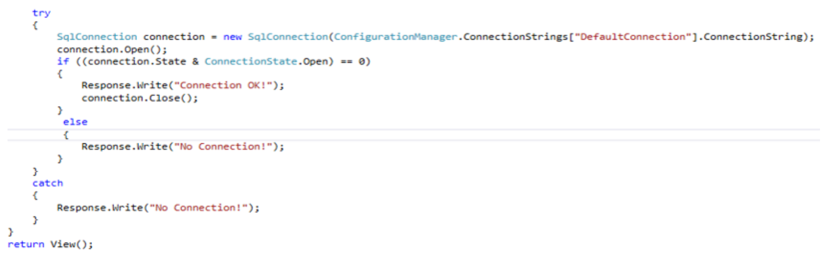


Integration Test *- (Benyam)*

Integration test is implemented after you have done unit testing, it is done in the architecture level in the V-model. There is Top-down and Bottom-up approach; we have chosen to do a mix of them both, because in some cases we made a bottom-up like in the testing of the connection string.

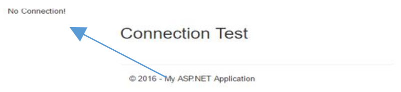
Before we deployed to the Server we had to make sure that our new database that was on the server side could be connected from our application with a connection string. Our development tool had a local database that it was connected to from the start. We thought about how we could test the connection to the server from our framework, the solution was to create a class that would test a connection from a string that we injected into the program.

The method is a simple one where it takes a connection string and opens it.



```
    try
    {
        SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString);
        connection.Open();
        if ((connection.State & ConnectionState.Open) == 0)
        {
            Response.Write("Connection OK!");
            connection.Close();
        }
         else
         {
            Response.Write("No Connection!");
        }
    }
    catch
    {
        Response.Write("No Connection!");
    }
}
return View();
```

It checks to see if it can connect and write a response to the view if there is a connection or not. *(See image below)*



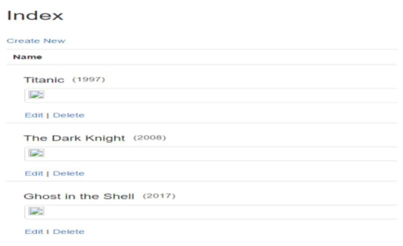
No Connection!

Connection Test

© 2016 - My ASP.NET Application

After we had done unit test for a part of the Movie and Actor, it was time to integrate them so that we knew they would work. We used GitHub to merge the classes to find out if our classes that we had made, could work together and implement the requirement we had done.

An unexpected problem that we found out, when we deployed our application on the azure server, was that it didn’t create the external folder for the images that we had created in our project. The reason why it didn't take the folder, was that azure only takes the frameworks folder that it creates when you start the project. This came as a surprise so we had to read the documentation in the Azure and Asp.net framework, to solve the problem.
So the problem was when we ran our application from the server http://moviebasen.azurewebsites.net/ the images we had was not there although all our functionality and data was working just fine.



Index

Create New

Name

Titanic (1997)

Edit | Delete

The Dark Knight (2008)

Edit | Delete

Ghost in the Shell (2017)

Edit | Delete

The solution we found out after many searches on the internet was that we had to push/publish the Image folder from our development tool to azure separately.

How could we avoid this mistake?
We think this comes with experience, working with server and reading the documentation for the development and the server environment.

Usability Testing *- (Mert, Elvis)*

Usability testing is a black box testing technique where the functionality is in focus, without diving into the code. We chose to make a hallway usability test, which means that we will find random persons and have them test it.
One of our main purposes with using usability testing is to find out how user friendly our app/web application is and if they feel comfortable with it doing different tasks. It is also our intention to make sure that the GUI is understandable for our users and check if the flow and layout appears naturally for our testers/users.

Tasks list:

1. Register to Movie Basen.
   a. Login.
2. Create movie in Movie Basen.
   . Fill in the required fields.
3. Create a new actor in Movie Basen.
   . Fill in the required fields.
4. Create a new Genre in the Movie Basen.
   . Fill in the required fields.
5. Add an actor to an existing movie.
6. Add a genre to an existing movie.
7. Change the movie picture in an existing movie.
8. Delete a movie from the list in Movie Basen.
9. Search after an existing movie.

While doing the usability test with our testers we have noticed:
Task 1 *(Register movie)*: It was easy for our testers to register and insert the information's that were needed.

Task 2 *(Create movie)*: The flow of finding the Movie page, inserting the data and then create the "movie" didn't take much time. However after inserting the movie, some of the testers remarked that there weren't any notifications when the movie was created. Another thing that the testers noticed was that the movie appeared last in the list.

Task 3 *(Create actor)*: Create a new actor in the MovieBasen was pretty easy, and it felt natural for our testers.
Task 4 *(Create genre)*: The response was positive, and that it felt natural creating genre.

Task 5 *(Add actor)*: Our testers couldn't find the add actor button, because it was placed in the details page of the movie *(note; when they were looking for it, they looked in the edit movie page).* Afterwards the testers remarked that they thought it should be in the edit movie page.

Task 6 *(Add genre)*: Our testers also concluded that the add genre button was placed wrong, just as the add actor *(because it was placed the same place)*.

Task 7 *(Change movie picture)*: Our testers found it easy to change the movie picture, but one of them found it bothersome going through all the steps, which were; find the picture on the internet, download it, and then add it to the movie *(this was also the case of task 2, when you create the movie, you also create a picture)*.

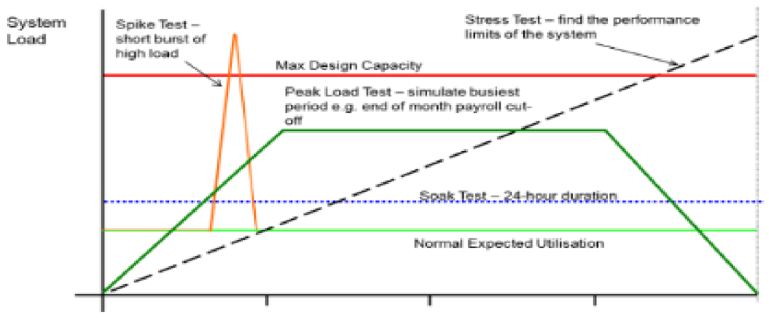
Task 8 *(Delete movie)*: Our testers found it easy to delete a movie, but most of them wanted some kind of notification when the deletion was completed.

Task 9 *(Search movie)*: Searching after a movie worked as expected, all the testers found it easy and natural and liked the placement of the search bar.

Changes made after usability test
After we got responses from our testers we changed some of the GUI, in order to make the app simpler, user friendly and with a better flow.
We moved the add actor and add genre functions from the movie details page to the movie edit page.
Some of the things we didn't do, due to lack of time was; To make a notification for every time you create and delete a movie. Second in task 2 and 7 when you add/change a picture in the movie, instead of going through all the step, we could make an API which connects to IMDB *(that contains all the pictures)* and therefore finds the picture, by writing the name of the movie.

Performance Test - *(Casper)*

Performance testing is non-functional testing, where you test the responsiveness and usage speed in an application (website, database, software).


System Load
Spike Test – short burst of high load
Stress Test – find the performance limits of the system
Max Design Capacity
Peak Load Test – simulate busiest period e.g. end of month payroll cut-off
Soak Test – 24-hour duration
Normal Expected Utilisation

Performance testing tool
We will be using the open source *Apache Jmeter* for testing the performance of our system. Where we will be running, the 4 different tests, such as Spike, Load, Stress and Soak test.
During our tests, we had some problems starting with it, because of the Jmeter that we didn't have much experience with.
After being consulted with the teachers and learning a bit more about the testing software, we figured how to use it out.

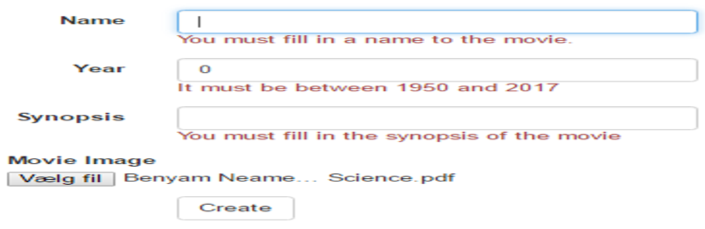
Look at appendix 2 and 3 for the rest of the test results and explanations

Aceptance Test *- (Benyam)*

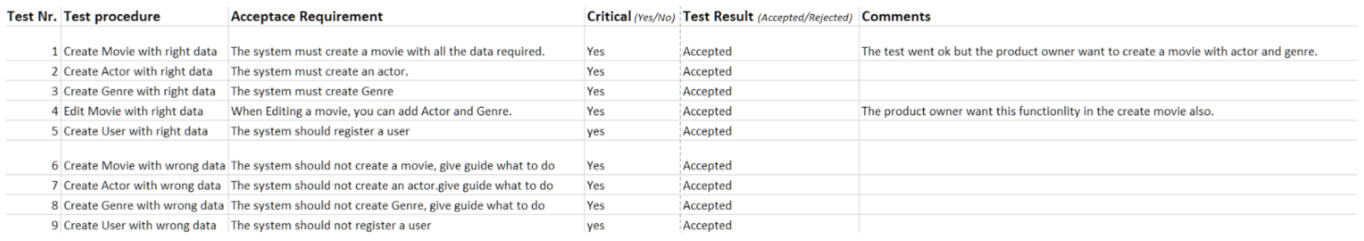
Acceptance test is one of the last test you do in the V-Model, usually it's the product owner that have the say in the outcome of the result, but since we did not have any, we chose one from the team to act as one. This can complicate things because it is easy to oversee or find a shortcut to get the desired result.

We made acceptance test for our CRUD Movie, Actor, Genre, the question was straight forwarded have we fulfil the requirement? Can the product owner execute the task in a manner that is easy to understand and fast?

We divide each acceptance test in two sections one with right data input, and the second with wrong and third with no data inputted. We excepted the system to function correct and if there was typed unexpected data (or wrong data) we had validation that would hinder and guide the user which steps he/she should do.



Name	|
	You must fill in a name to the movie.
Year	0
	It must be between 1950 and 2017
Synopsis	
	You must fill in the synopsis of the movie
Movie Image	Vælg fil Benyam Neame... Science.pdf



Test Nr.	Test procedure	Acceptace Requirement	Critical *(Yes/No)*	Test Result *(Accepted/Rejected)*	Comments
1	Create Movie with right data	The system must create a movie with all the data required.	Yes	Accepted	The test went ok but the product owner want to create a movie with actor and genre.
2	Create Actor with right data	The system must create an actor.	Yes	Accepted	
3	Create Genre with right data	The system must create Genre	Yes	Accepted	
4	Edit Movie with right data	When Editing a movie, you can add Actor and Genre.	Yes	Accepted	The product owner want this functionlity in the create movie also.
5	Create User with right data	The system should register a user	yes	Accepted	
6	Create Movie with wrong data	The system should not create a movie, give guide what to do	Yes	Accepted	
7	Create Actor with wrong data	The system should not create an actor.give guide what to do	Yes	Accepted	
8	Create Genre with wrong data	The system should not create Genre, give guide what to do	Yes	Accepted	
9	Create User with wrong data	The system should not register a user	yes	Accepted	

Overall the acceptance test went as expected, and we could confirm that the requirements we tested (acceptance requirements) was met.

Git - Version Control *- (Mert, Elvis)*

Version control tools like Git, manage to keep track of every modification in the code/program, which is made by different users.

In this project, we used Git extension and Github desktop as version control tools, where we uploaded the project on a git repository. We all cloned the project from the git repository, which was put online, and made our own branches.

We have put our commits in a Git Commit-Table, where you can see the progress made to the program.

-------------------- Look at appendix 1 for the Git Commit-Tabel and link to Github --------------------

Our experience using GIT

The things we found out during merging the different branches with master, was that some of us got merge conflicts, so that we had to roll-back to an earlier commit and fix it before we could merge it with the master branch.
We sometimes also forgot to pull before pushing the changes we have made. Which resulted in some more errors when pushing to the branch.

Even though we had some problems with the database and the program itself after merging, we managed to get it to work, by pulling and working on the newest version.

Reflection - *(All)*

There have occurred different challenges during this project, where we had to find a solution. One of the things that took a lot of time in the start, was to set up the git-repository and all have a connection to it.

The overall requirement we had from the start for this project was to build a product with less or none defects and error that could occur in constructing an application. The software testing course have guided us to check and balance our workflow so that we could identify the errors and defects beforehand. From unit testing to acceptance testing it all had a profound mindset with tools and ways for us to make our programming flow to go smoothly and educational.

The unit test helped us to identify the errors that could happen in the smallest component before we started to integrate to a fully functional model/class. We could easily test the different field/data that our model was expecting and see if we had the expected behavior and in a desired way before we integrated it to the system.

The integration test was a critical part of our test because we had to integrate our application to database, server and our classes together. We learned there is different tools and ways to do the testing but the most educational was that we learned it's a critical part in testing.

Performance test we learned this is a good practice to implement before releasing the application so that we could know and have an overview of how our application would behave when using it.

These are some of the tests and test methods we implemented for our project, unfortunately we can't go through all of them here in our reflection but in overall, they have helped us to progress and think differently before we start to program an application for the next time.

Bibliography - *(All)*

Book:

- SOFTWARE TESTING, An ISTQB–BCS Certified Tester Foundation Guide Third edition

Web-Pages:

- http://jmeter.apache.org/usermanual/build-web-test-plan.html
- https://www.raywenderlich.com/101306/unit-testing-tutorial-mocking-objects
- http://istqbexamcertification.com/what-is-integration-testing/
- https://www.usability.gov/how-to-and-tools/methods/running-usability-tests.html
- http://istqbexamcertification.com/what-is-acceptance-testing/
- https://www.tutorialspoint.com/software_testing_dictionary/test_case.htm
- https://www.mindtools.com/pages/article/newTMC_07.htm