Tool Support for Inspections

Alexander Egorov

Kongens Lyngby, 2013

IMM-MSc-2013-95

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Building 303B, Matematiktorvet, DK-2800 Kongens Lyngby, Denmark
Tel.: +45 45253031, Fax: + 45 45881399
reception@compute.dtu.dk
www.compute.dtu.dk IMM-MSc-2013-95

Summary

Nowadays software systems play an important role in a human life, since they are present almost in every aspect of it. Software systems surround us and they are everywhere: from tiny little devices such as thermometers and sensors to airplanes and production factories. Unfortunately, these software systems are not completely free from defects. Defect can cause a small embarrassment or a large scale disaster. However there are techniques that can help software developers to find defects, and formal inspection is one of those techniques. The technique can be applied to any development document at any stage of the development process.

This thesis is dealing with a modified version of formal inspection process that suits better for academic purposes. The inspection process is considered to be rather complex, it involves many interactions and paper work, which distracts participants from defect finding. In order to make the process more transparent an online support tool should be created. Due to software system introduction to the inspection process, the process was modified in order to fit better to the new conditions. Background analysis of previous works was conducted and their drawbacks were analyzed, concluding that a new tool should be implemented using a different approach.

The support tool received name "FIT" that stands for "Formal Inspection Tool". Since the tool is a web based application it is possible to make a clear separation between a front-end and back-end components. Bootstrap framework was user to produce the front-end component, Ruby on Rails web application framework was used in the back-end component. The application was deployed on Heroku that is PaaS provider. FIT is a free open-source software product, which sources are worldwide publicly available through GitHub.

The security issues of a web application represent a great concern, thus the application was analyzed on possible vulnerabilities and security breaches. The analysis showed that FIT is not a subject to most popular attacks.

Preface

This thesis was prepared at the Institute for Applied Mathematics, Technical University of Denmark in order to acquire the MSc degree in Computer Science. This thesis was prepared between 5th of February and 30th of August under the supervision of Associate Professor Dr. Harald Störrle. This thesis is worth 35 ECTS credit points.

The thesis deals with subject of formal inspections. The outcome of the project is a fully functional web application that is used in the process of formal inspections within academic environment. Preliminary research, design analysis, prototyping, detailed design and implementation, testing and deployment of this application is performed and described in this thesis. Chapter 1 describes theoretical background for the application. Chapter 2 presents the analysis of the problem with regards to the web application. Chapter 3 focuses on the application design and describes change decisions that was made during the project. The implementation of the application is presented in Chapter 4. The results of the implementation and testing results are described in Chapter 5. Conclusion section focuses on possible follow-up on the application. The content that is delivered with the thesis is described in Appendix A. Appendix B describes terms and abbreviations used in the thesis. Appendices C and D contain accordingly lists of figures and tables that were presented in this thesis.

Lyngby, 30 August 2013

[illegible]

Alexander Egorov

Acknowledgements

I would like to thank Harald Störrle for his great supervision, not to mention that it was a pleasure for me to work with him during the project.

I am glad to give exceptional thanks to Andrii Sereda, Anton Makarov, and Vlad Acretoaie for providing a shoulder to lean on during difficult moments and giving useful advices when it was needed.

I would like to say special thanks to Attila Sukosd, who helped me with security analysis.

And of course I cannot express the amount of acknowledgement to my parents, who always motivate me to push forward.

Contents

1 Background

Very often software is shipped with defects. A software defect is a deficiency in a software product that causes it to preform unexpectedly, while from user perspective a defect is a cause for the software not to meet user's expectation (McDonald, Musson, & Smith, 2007). An example of user's perspective on software defects is presented in Table 1.

User Expectation	Software defect
The software will help in accomplishing a task.	Desired functionality is missing.
The software will respond quickly.	The software responds too slowly from the user's perspective.
The software is secure from hackers.	Hackers are able to exploit software vulnerabilities and attack the system.
A fatal error message is shown in case of inappropriate actions or system malfunction.	No error is presented, and the software freezes.

Table 1: Software defects from the user's perspective (McDonald, Musson, & Smith, 2007)

Software defects can cause user's dissatisfaction and may even lead to project failure if the amount of defects is too high. It is a well-known fact that the cost of error elimination in software development increases as project progress. Clearly to reduce overall cost of the project, it is vital to make an attempt of finding and fixing errors as early as possible.

There are several techniques designed for finding defects and improving software quality during software development process. Formal inspection (often referred as Fagan inspection) is one of those techniques. During 02264 Requirement Engineering course the author had a possibility to participate in two formal inspections.

1.1. *Formal Inspection*

The software inspection process was created for purposes of both improving software quality and increasing productivity by M. Fagan in 1972. An inspection is a formal, efficient and economical method of locating errors in both design and code (Fagan M. E., 1976). The development process with two inspections is demonstrated on Figure 1.

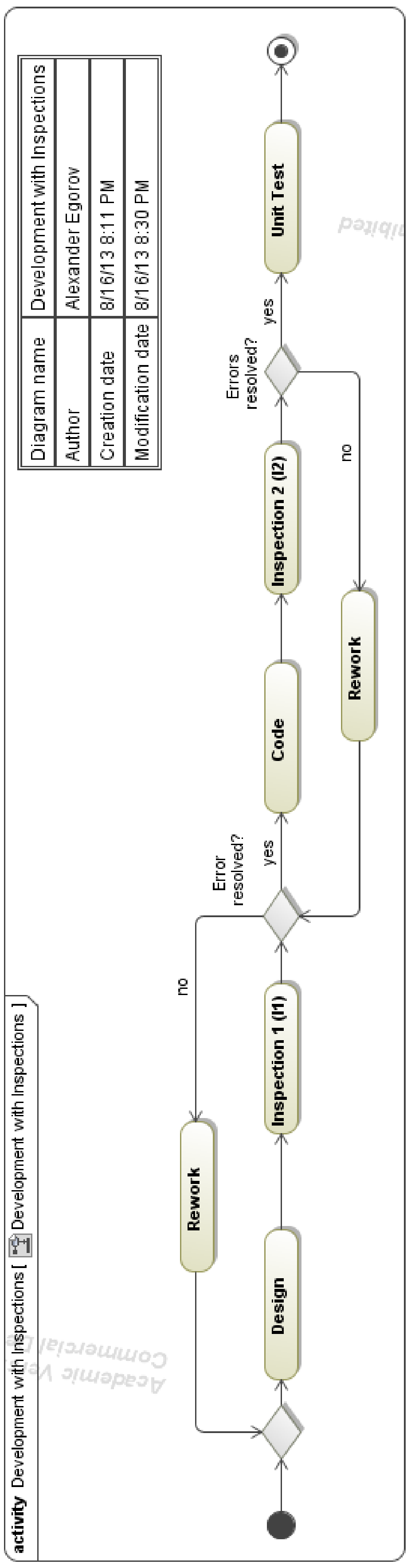


Figure 1: Development with Fagan Inspections

Inspection Roles

The inspection team consists of inspection participants that are usually programmers who are chosen form the project involved in the inspection. There are several roles to be played by inspection participants during inspection process. The roles are presented in Table 2.

Role	Description
Moderator	The moderator is the key person for a successful inspection. It is not necessary for a person playing the moderator role to be a technical expert on the project being inspected, but he must be qualified programmer. It is strongly recommended that the moderator should not originate from the project that is subject to inspection, therefore on the grounds of maintaining objectivity and increasing integrity of the inspection the moderator should be picked from an unrelated project (Fagan M. E., 1976). Managing the inspection team and guiding the inspection direction are the major responsibilities of the moderator, in consideration of that the person playing the role must have a leadership qualities such as personal sensitivity, tact and passion. In order to produce synergistic effect he should utilize the strength of team members. The moderator is responsible for scheduling inspection stages, reporting on the results of the inspection and follow-up on rework. Briefly trained moderator produce the best results (Fagan M. E., 1976).
Designer (Author)	The inspection participant is responsible for producing the program design.
Coder/Implementer (Author)	The Coder is implementing the design in code.
Tester	The programmer responsible for writing and/or executing test cases or otherwise testing the product of designer and coder.

Table 2: Inspection Roles

Inspection Process

The inspection process consists of five stages: `Overview`, `Preparation`, `Inspection`, `Rework` and `Follow-up` that are demonstrated in Figure 2.

The first stage called `Overview`. The whole inspection team participates in this stage of the inspection process. During the first inspection (I_1) the designer describes the overall idea behind the inspected design and provides the rest of the team with appropriate documentation.

The next stage – `Preparation` - is performed individually. Inspection participants read up the documents provided by the author on the previous stage in order to understand the design, its intent and logic. The most discernible errors are found during the preparation stage.

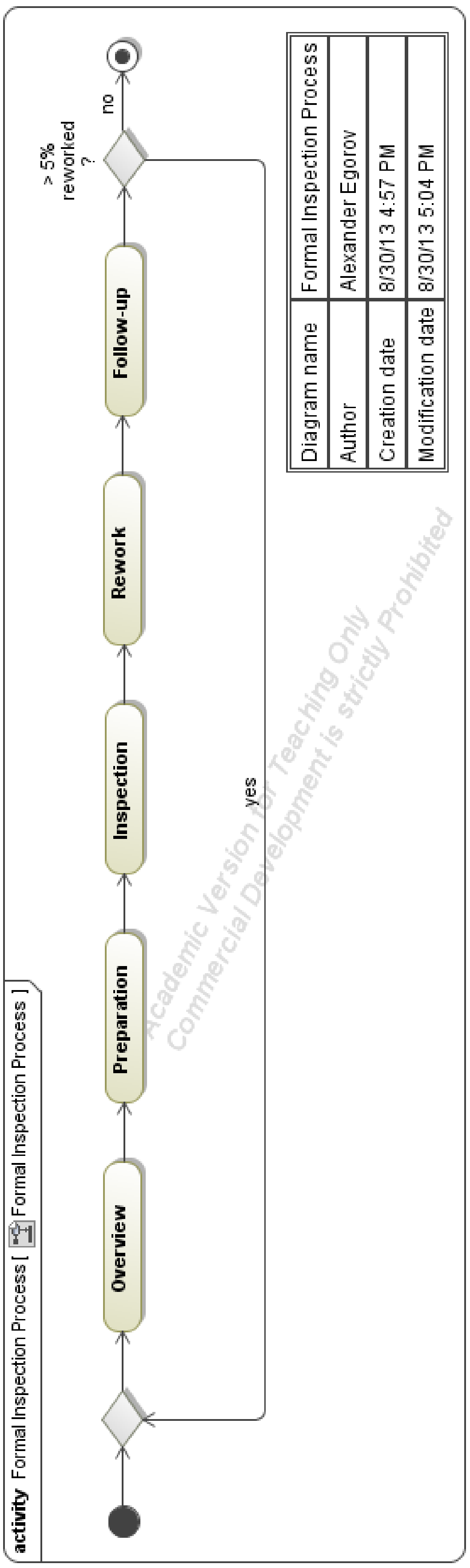

activity Formal Inspection Process [Formal Inspection Process]
Overview
Preparation
Inspection
Rework
Follow-up
> 5% reworked ?
no
yes

Diagram name	Formal Inspection Process
Author	Alexander Egorov
Creation date	8/30/13 4:57 PM
Modification date	8/30/13 5:04 PM

Figure 2: Inspection Process

The first stage called `Overview`. The whole inspection team participates in this stage of the inspection process. During the first inspection (I_1) the designer describes the overall idea behind the inspected design and provides the rest of the team with appropriate documentation.

The next stage – `Preparation` - is performed individually. Inspection participants read up the documents provided by the author on the previous stage in order to understand the design, its intent and logic. The most discernible errors are found during the preparation stage.

On the `Inspection` stage the whole team is gathered. The moderator chooses a "reader" (typically the coder) that will describe how he will implement the design. The reader explains his vision on the design described by the designer. The objective of the Inspection stage is to find errors. The error finding is actually done during the "readers" discourse (Fagan M. E., 1976). Fagan recommend limiting the inspection meetings to two hours, due to the fading in error detection efficiency. (Fagan M. E., 1986). The moderator should write a report no later than one day after the stage conclusion where he states all errors or problems.

During the `Rework` stage all errors or problems mentioned at the inspection report should be resolved by the designer or coder.

It is crucial that at the `Follow-up` stage every issue is addressed. The moderator is responsible for ensuring all issues discovered during the inspection are resolved. Reinspection should be carried out in case of more than five percent of the materials has been reworked.

Process operations	**Rate of progress[1], loc/hr.[2]**		**Objectives of the operation**
	Design (I_1)	Code (I_2)	
Overview	500	not necessary	Communication, education
Preparation	100	125	Education
Inspection	130	150	Find errors
Rework	20 hrs./K.NCSS[3]	16 hrs./K.NCSS	Rework and fix error found by inspection
Follow-up	-	-	Ensure that all errors, problems and concerns have been resolved

Table 3: Inspection process and rate of progress (Fagan M. E., 1976)

[1] According to M. Fagan the exemplified rates are conservative and applied to systems programming, while comparable rates for applications programming are much higher

[2] Lines of code per hour

[3] 1000 Non-Commentary Source Statements

Inspection Benefits

Purging error from the project results in the immediate improvement in productivity (Fagan M. E., 1976). Observation of sample of software developers proposed that experience gained from inspections led to reduction in defects inserted in the design and code of systems created later during the same project[4].

Studies conducted by M. E. Fagan showed that the cost of rework on errors as a fraction of overall project cost is significantly higher that for the projects were inspections were not used in comparison with the projects were inspections were conducted (Fagan M. E., 1976).

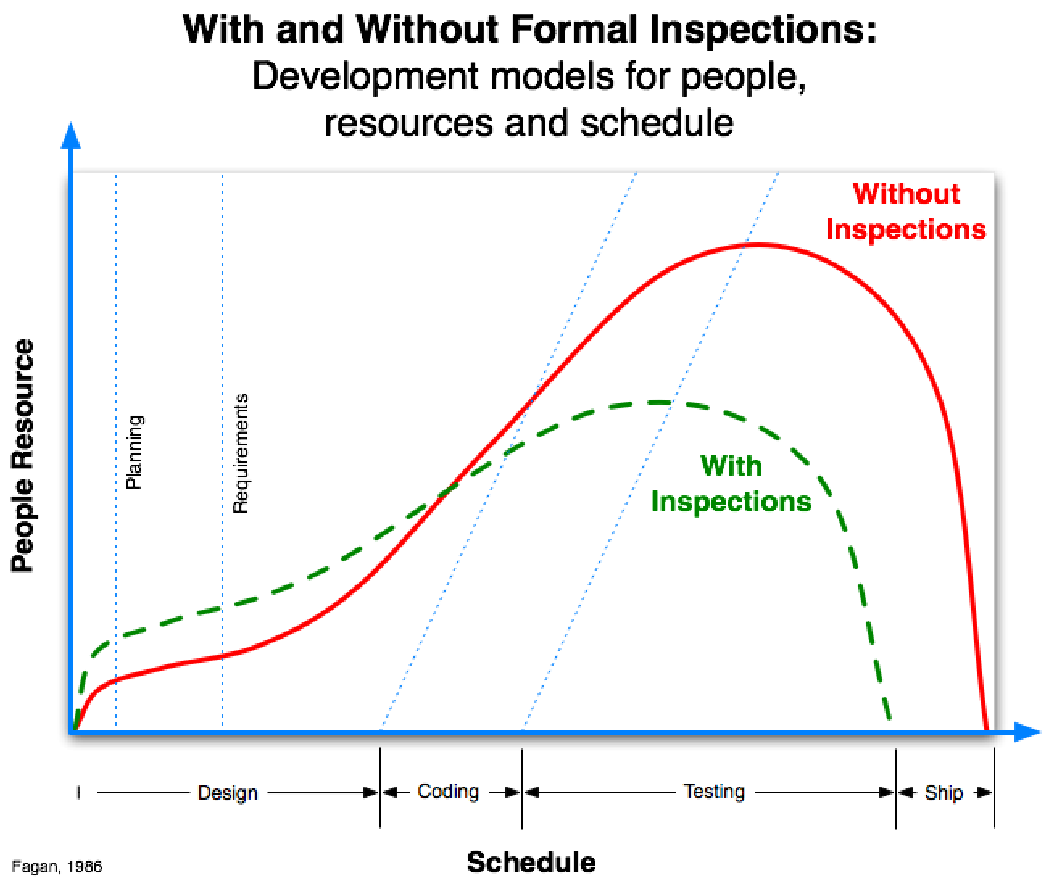


Figure 3: Software Development Process (Fagan M. E., 1986)

The main benefits from conducting inspections are defect reduction, defect prevention and cost improvement (cost of inspections is included in the project cost) (Fagan M. E., 1986). Fagan estimates all design and code inspections cost to be around 15 percent of the project cost.

1.2. *Fagan-style Inspection*

An adaptation of Fagan inspections for academic usage was proposed by associate professor H. Störrle. Störrle introduced new terminology to the inspection process and modified rules of the inspection. The Fagan-style inspection is supposed to be

[4] IBM Technical Newsletter GN20-3814, Base Publication GG20-2000-0, Aug. 15, 1978

conducted on a project-oriented course where a project is conducted by a group of students. Each group participates in an inspection process presenting their project to other group.

An `artifact` is a piece of documentation, code, model, diagram or any file that is intended for the inspection. A `campaign` is a group of individual inspections that are conducted simultaneously for all groups participating in the course.

The rules of inspection changed in the way suitable for an academic purpose. There are five (six) possible roles that can be taken during the inspection process: `author`, `moderator`, `inspector`, `scribe`, `admin`, and `supervisor`. An inspection team should consist of three to seven members one of which must be the moderator and another one the author (Störrle, QA3 Inspection Process Guide, 2012).

An author is a person that represents a group whose project is subject to the inspection. Obviously an author should originate from the same group that produced artifacts that are inspected. The moderator role is kept almost intact and well coincide with Fagan's descriptions. Another role introduced is an inspector, who is responsible for providing a feedback on a given artifact. The scribe takes notes during the inspection meeting; typically this role is carried on by the moderator. The moderator and the inspectors should originate from different group in order to reduce ambiguity and provide an independent unbiased point of view on the artifact being inspected. Figure 4 shows the inspection process of Fagan-style inspection.

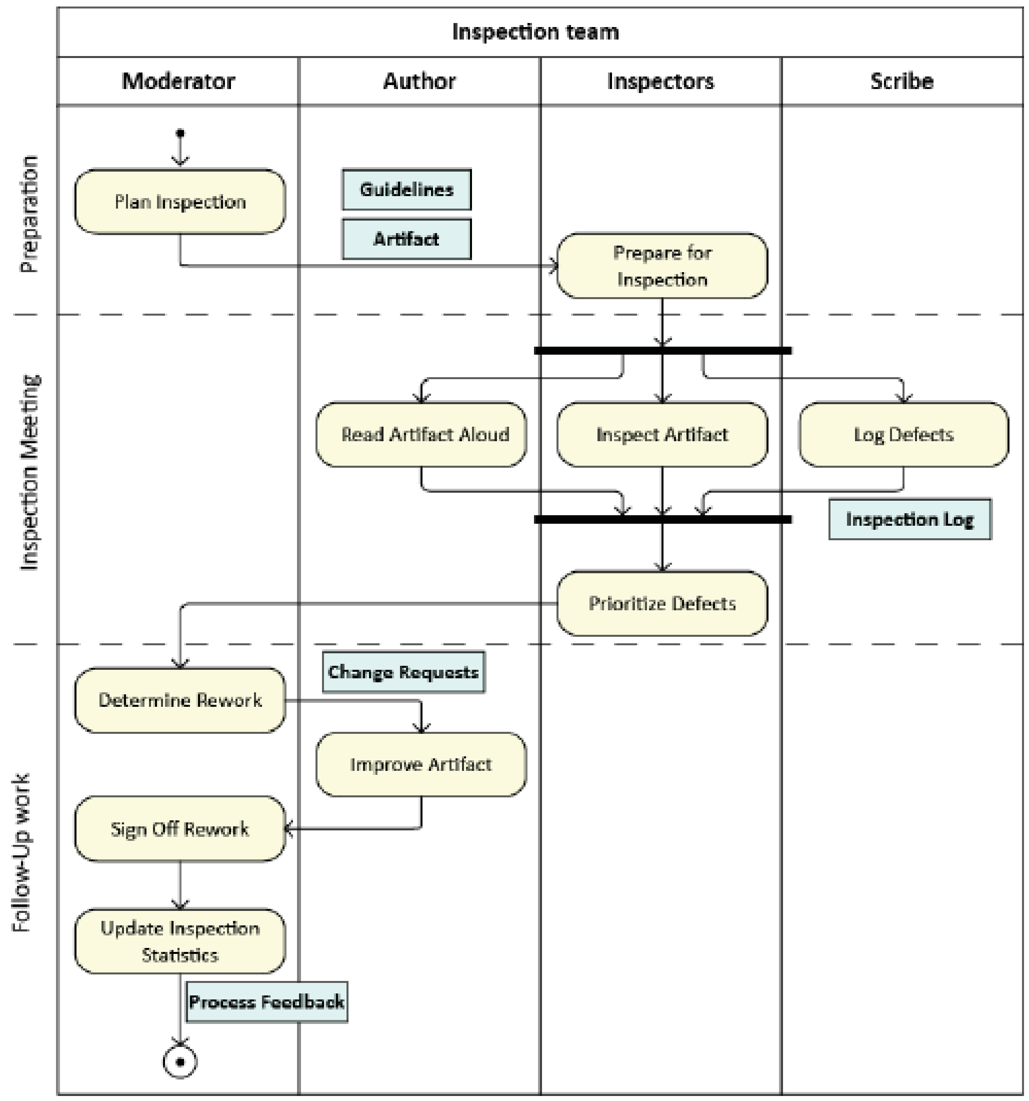


Figure 4: Inspection process (Störrle, Requirements Engineering, 2012)

The admin role is taken by teacher or teacher assistants; the role includes overall supervision of an inspection process. The admin provides help and assistance for the inspection team participants, assigns roles and aggregates the results of the process. Table 4 demonstrates compatibility of roles and their responsibilities.

Role	Responsibilities		Compatibility			
	Tasks	Forms to fill	*M*	*S*	*A*	*I*
Moderator	• Organize inspection • Collect preparation • Lead meeting • Determine rework • Follow up on rework • Sign off process steps	• Inspection Process Summary • Inspection Preparation Summary • Additional Rework Assignments	▨	✓	✗	✓
Scribe	• Compile and consolidate inspectors' comments • Take notes during inspection meeting • Support moderator, as assigned by moderator	• Inspection Preparation Summary • Additional Rework Assignments	✓	▨	✗	✓
Author	• Prepare and make available inspection artifact • Present inspection artifact in meeting • Answer questions to inspectors • Forward inspection results to co-authors (if any)	none	✗	✗	▨	✓
Inspector	• Inspect artifact as meeting preparation • Submit results to moderator • Explain and elaborate comments during inspection	• Individual Inspection Preparation • Additional Comments	✓	✓	✗	▨

Table 4: Roles and responsibilities in the inspection process (Störrle, QA3 Inspection Process Guide, 2012)

In what follows, 'Fagan-style inspection' process is assumed, when an 'inspection', 'inspection process' or a 'formal inspection' is mentioned.

1.3. *Need for Tool Support*

Figure 5 demonstrate the results of evaluation of formal inspection process.

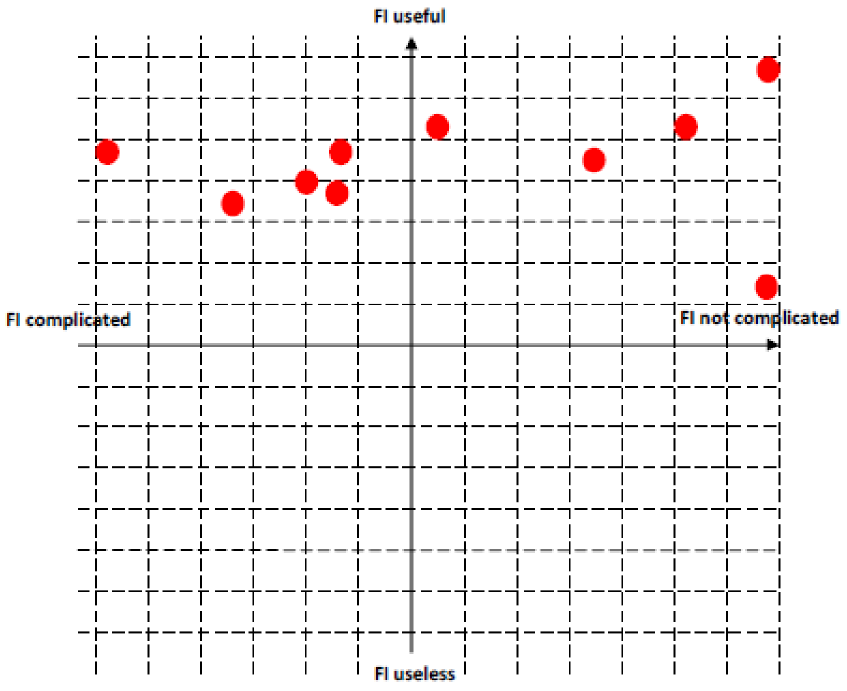


Figure 5: Evaluation of formal inspection process in utility and complexity scale (Petrolyte, 2011)

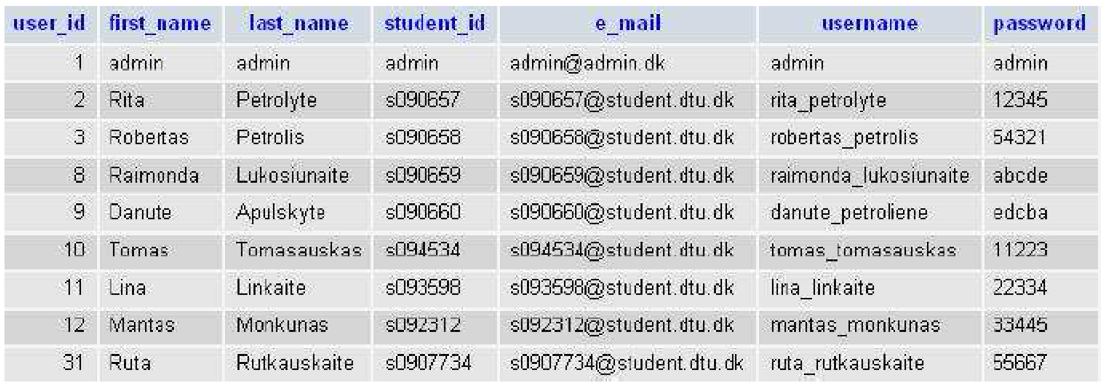
The evaluation was conducted among 02264 Requirement Engineering course listeners who participated in formal inspections. The evaluation showed that all participants recognize the inspection process as a useful technique, while half of the polled students consider the inspection process complicated.

As it is possible to conclude from the evaluation, there should be a support tool for the inspection process. In order to reduce the process complexity, support distributed work, and provide better control on the inspection process for a supervisor, a support tool should be created.

1.4. *Previous work (FIT-1)*

The previous work was conducted in 2011 (Petrolyte, 2011). The support tool that was created suffered from a group of major drawbacks.

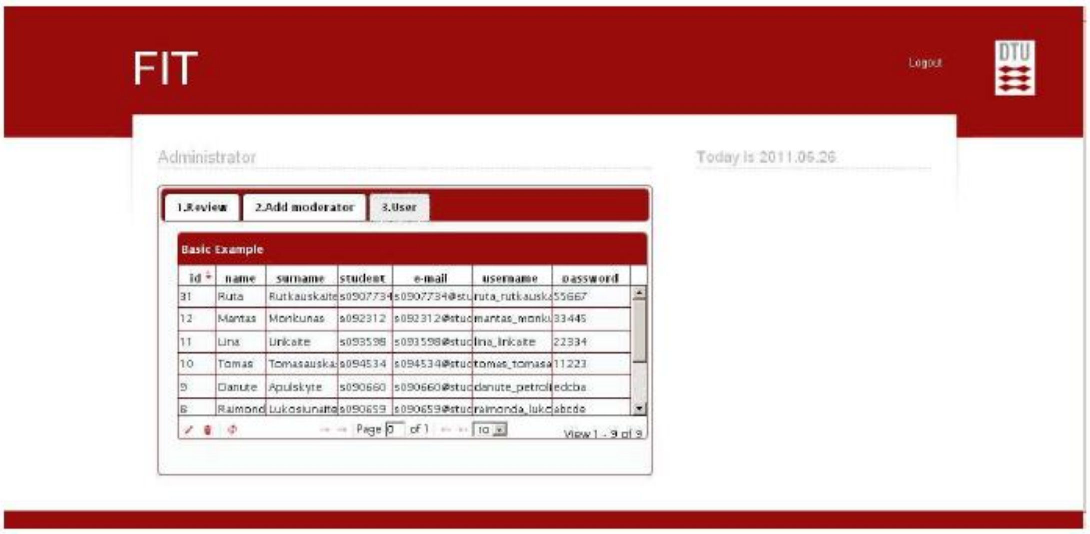
During the 02264 Requirements Engineering course the author of the thesis had experience with a previous version of a Formal Inspection Tool. The system handicaps became obvious from the very first moment after the system was presented to the course listeners. The author experienced a problem with logging in to the system. The problem was solved in an extravagant way; the author’s teammate on the course project, Attila Sukosd, appeared to be a very talented programmer. He was able to locate the error and exploit the tool vulnerabilities in order to gain privileged access to the system in 10 minutes. After another couple of minutes the problem was fixed and the author could log in to the system. The previous system was a subject to a great deal of vulnerabilities, such as SQL injection, JS injection and HTML injection. The input was not filtered at all and uploaded files were stored in a publicly accessible location. It was also possible to execute the uploaded script. Table 5 shows that user passwords were stored in a plain text, which is a great security threat (MITRE Corporation, 2011). It is essential to notice, that user passwords were also exposed to the administrator of the system as it is shown in Figure 6.



user_id	first_name	last_name	student_id	e_mail	username	password
1	admin	admin	admin	admin@admin.dk	admin	admin
2	Rita	Petrolyte	s090657	s090657@student.dtu.dk	rita_petrolyte	12345
3	Robertas	Petrolis	s090658	s090658@student.dtu.dk	robertas_petrolis	54321
8	Raimonda	Lukosiunaite	s090659	s090659@student.dtu.dk	raimonda_lukosiunaite	abcde
9	Danute	Apulskyte	s090660	s090660@student.dtu.dk	danute_petroliene	edcba
10	Tomas	Tomasauskas	s094534	s094534@student.dtu.dk	tomas_tomasauskas	11223
11	Lina	Linkaite	s093598	s093598@student.dtu.dk	lina_linkaite	22334
12	Mantas	Monkunas	s092312	s092312@student.dtu.dk	mantas_monkunas	33445
31	Ruta	Rutkauskaite	s0907734	s0907734@student.dtu.dk	ruta_rutkauskaite	55667

Table 5: FIT-1 database user table (Petrolyte, 2011)

The design decisions were rather controversial, as it may be seen from the Figure 6 while there is plenty of unused horizontal space on the page, an overflow occurs on all cells in e-mail column and many other cells where long text string is present.



id	name	surname	student	e-mail	username	password
31	Ruta	Rutkauskaite	s0907734	s0907734@stu	ruta_rutkausk	55667
12	Mantas	Monkunas	s092312	s092312@stud	mantas_monku	33445
11	Lina	Linkaite	s093598	s093598@stud	lina_linkaite	22334
10	Tomas	Tomasauska	s094534	s094534@stud	tomas_tomasa	11223
9	Danute	Apulskyte	s090660	s090660@stud	danute_petroli	edcba
8	Raimond	Lukosiunaite	s090659	s090659@stud	raimonda_luko	abcde

Page 0 of 1 10 View 1 - 9 of 9

Figure 6: FIT-1 administrator panel (Petrolyte, 2011)

The other system handicap was an absence of non-ASCII characters support, in other words it was impossible to use national keyboard layouts (e.g. Chinese, Russian) or special characters.

The absence of communication medium for the inspection participants may also be considered as a major drawback of the system.

Looking at the system from administrator’s point of view shown in Figure 6, it is possible to notice the lack of possibility for the administrator to have an overview on several inspection activities.

The summary of system handicaps is presented in Table 6.

Drawback
Absence of communication medium
Vulnerability to SQL injections
Vulnerability to Cross-Site Scripting
Artifacts stored in public folder
Passwords stored in plain text
Absence of non-ASCII character support
Absence of administrator tools

Table 6: Previous system drawbacks

1.5. *The New System*

Considering the drawback of the previous system it was decided to produce a completely new system that would utilize different principles, i.e. a modern web application framework should be involved into the development process, the system should be secure and modular.

In order to provide a quality services for the inspection participants the drawback of the previous system should be precluded. Hence the system should:

- be written in one of well acknowledged frameworks (in order to reduce overhead and do not repeat on what has been used)
- support non-ASCII characters

- not be a subject for simple attacks
- provide a communication medium for inspection participants
- have a support of administrative function
- provide guidance for the system users

2 System Analysis

2.1. *System Users*

In order to provide a system that satisfies user expectations it is vital to understand the target audience of the system. The system is supposed to implement Fagan-Style inspection process that is applied at the academic institutions, therefore students, professors and teacher assistants are representative auditory of the system.

Since future system users are determined it is important to understand what roles they would play during system operation. The determination can be carried out without any hesitation, on grounds of Fagan-Style inspection description. While students will utilize the system as ordinary users with no special privileges and restricted rights, i.e. they would have such roles as moderator, inspector, and author. Teaching staff will have full access to the system with their admin role. Considering that scribe role in most of the times is combined with the moderator role even in paper-based version it was decided to exclude scribe role from the inspection process in support tool.

Taking into the account the 02264 Requirements Engineering course[5] participants as a base for the system user estimation, the typical user would have knowledge in programming languages, understand basic concepts of software engineering, and have a basic knowledge of a modeling language, such as UML[6]. Users with full-fledged access to the system will have more advanced knowledge and skills than ordinary users.

2.2. *Use Cases*

The use cases provide the short summary on the required system features. Table 7 shows use cases for FIT system.

User Case Overview

Use case	Description
Create Campaign	The administrator creates a campaign. Is should be also possible to user a spreadsheet with staffing information in order to create underlying inspections and grant roles to users.
Edit Campaign	The administrator can edit the campaign, change name, delete campaign or add new inspection to campaign.
Delete Campaign	The administrator can delete the campaign with all inspections inside the campaign.
Write Message	Any system user can write a message in the inspection chat.
Create User	Anyone can create a new profile in the system.
Edit User	Non privileged user can edit his profile, while the admin can

[5] http://www.kurser.dtu.dk/02264.aspx?menulanguage=en-GB

[6] The estimation of user's knowledge and skills is taken from 02264 Requirements Engineering course prerequisites.

	edit any user profile.
Delete User	User account can be deleted by the administrator.
Upload Artifact	The author uploads the artifact.
Delete Artifact	The moderator can delete the artifact if it is consideret inappropriate. The author can delete the artifact he uploaded.
Edit Artifact	The author can change displayed name, comment, and content of the artifact he uploaded.
Download Artifact	Any inspection participant can download the artifact.
Create Remark	The inspector or the moderator can create a remark.
Upload Remarks	The inspector or the moderator can upload remarks from a spreadsheet.
Delete Remark	The inspector can delete remarks that he created, when the moderator can delete any remarks.
Download Remarks	Any inspection participant can download all remarks for the inspection as file.
Create Inspection	Inspection can be created by the admin only, during the creation process the admin can specify to what campaign the inspection would belong
Edit Inspection	Inspection can be edited by the admin, e.g. he can add or remove participants from the inspection.
Delete Inspection	The inspection can be deleted by the administrator only. This assumes that all uploaded artifacts and created remarks will be deleted.
Add Participant	The admin can add participant to the inspection with specified role.
Delete Participant	The admin can delete participant from the inspection.
Change Deadlines	Inspection deadline can be changed by the moderator. The deadline date of an inspection stage should be within the interval of neighboring inspection stages deadline dates.
Change Inspection Status	The inspection status can be change by the moderator.

Table 7: Use case overview

Use case diagram for overview is presented in Figure 7.

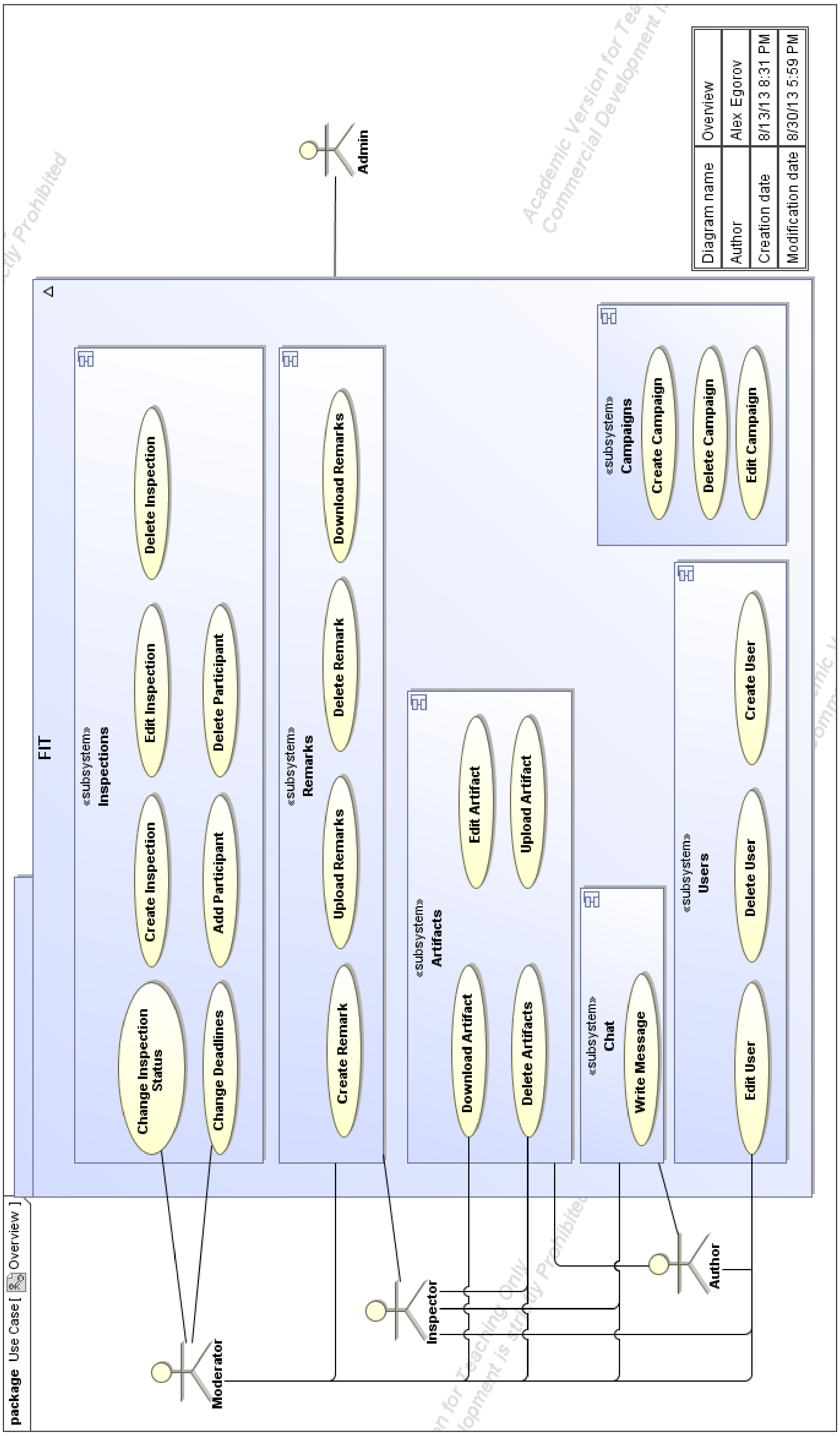
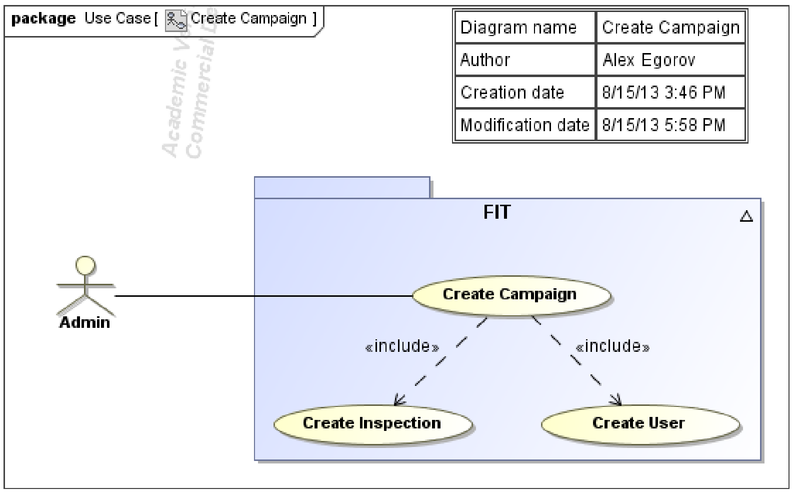


Figure 7: Overview use case diagram

Highlights of Important Use Cases

The most important and complex use cases are `Create Campaign`, `Upload Artifact`, and `Upload Remarks`. They are described in detail below.



package Use Case [Create Campaign]
Diagram name | Create Campaign
Author | Alex Egorov
Creation date | 8/15/13 3:46 PM
Modification date | 8/15/13 5:58 PM
FIT
Admin
Create Campaign
«include»
«include»
Create Inspection
Create User

Figure 8: Create campaign

UC-1	Create Campaign
Description	An administrator creates a campaign
Actors	Administrator
Triggers	'Create Campaign' URL visited
Preconditions	The administrator is authenticated
Regular Scenario	1) The administrator types a campaign name and uploads a spreadsheet file with role assignments. 2) The administrator inputs a default deadlines for inspections. 3) FIT ensures that the entered information is valid, creates a campaign, then parses the spreadsheet and creates inspections and assigns roles to users. 4) The campaign view with a list of the campaign inspections is presented to the administrator.
Exceptional Scenario	1) Entered data is wrong, error is displayed. 2) No spreadsheet was presented, an empty campaign is created. 3) The role assignments file contains missing or conflicting roles for several inspection participants, then error is displayed and corresponding inspection highlighted with red.
Results	New campaign with inspections is created. Roles to the participants are assigned
Frequency	2/4/6 (min/avg/max)[7] pr. semester

[7] Minimal frequency corresponds to two campaigns during one course, like 02264 Requirements Engineering

Table 8: Create Campaign Use Case

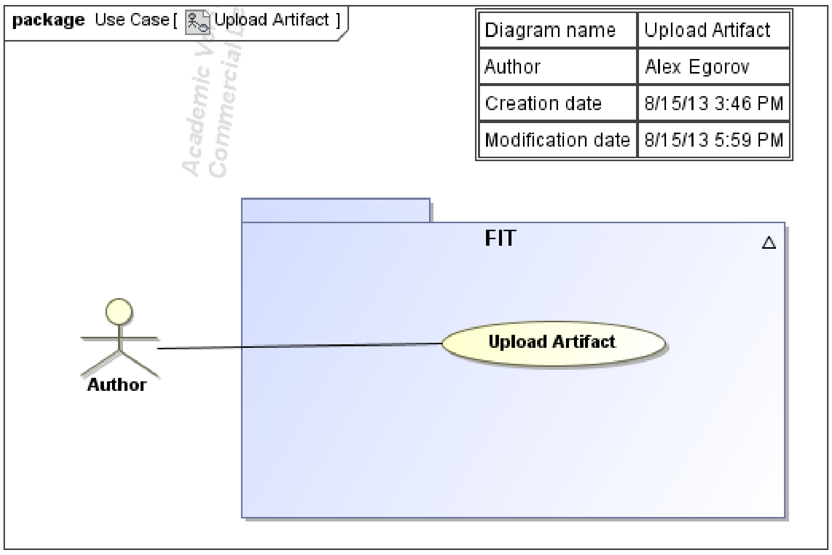


Figure 9: Upload artifact

UC-2	Upload Artifact
Description	An author uploads artifact to an inspection
Actors	Author
Triggers	‘Upload Artifact’ URL visited
Parameters	-
Preconditions	The author is authenticated and signed up for the inspection
Regular Scenario	1) The author visits an inspection where he wants to upload an artifact. 2) The author clicks to ‘Upload Artifact’ button, selects a file he wants to upload and then clicks ‘Upload’ button. 3) FIT ensures that the file size is within established limits. 4) Inspection view that contains newly created artifact is presented to the author.
Exceptional Scenario	1) Filename contains dangerous symbols; the symbols are changed with safe ones. 2) No file was presented, an error is displayed.
Results	New artifact is created
Frequency	1/6/15 (min/avg/max) pr. inspection
Comments	-

Table 9: Upload artifact use case

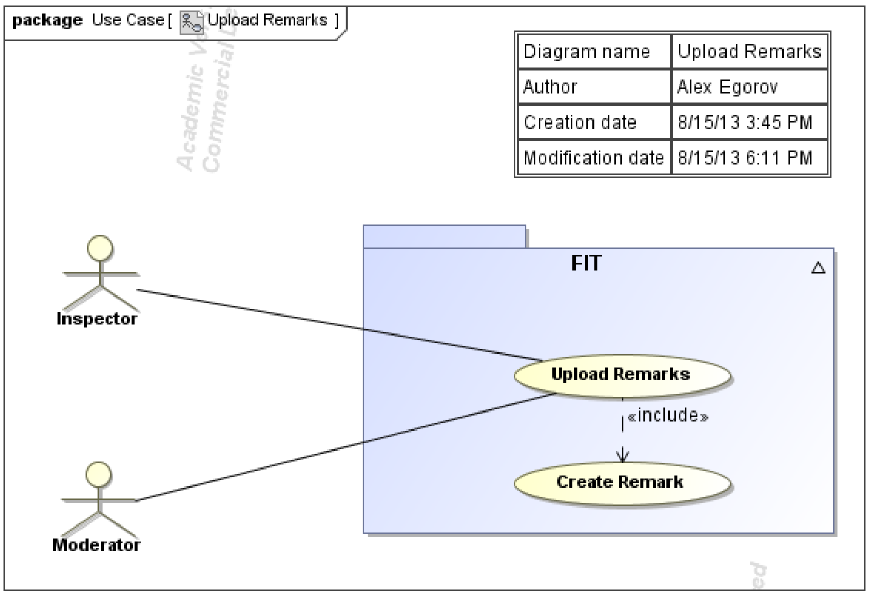


Figure 10: Upload remarks

UC-3	**Upload Remarks**
Description	An inspector or a moderator uploads remarks to an inspection
Actors	Inspector, Moderator
Triggers	‘Upload Remarks’ URL visited
Parameters	-
Preconditions	The participant is authenticated and signed up for the inspection
Regular Scenario	1) The participant visits the inspection where he wants to upload remarks. 2) The participant clicks to ‘Upload Remarks’ button, selects a file he wants to upload and then clicks ‘Upload’ button. 3) FIT ensures that the file has an appropriate format. 4) Inspection view that contains newly uploaded remarks is presented to the participant.
Exceptional Scenario	1) Several remarks have incorrect formatting and/or missing attributes and error is displayed. 2) No or unsupported file was presented, an error is displayed.
Results	Remarks from the file are appended to the Remarks table
Frequency	0/2/15 (min/avg/max) pr. participant
Comments	-

Table 10: Upload remarks use case

2.3. *Usage Scenario*

In the interest of providing better machine-user interaction it is crucial to understand how the system is going to be used. Usage scenario is one of many tools that help to build up this understanding. There are a few possible ways how usage scenarios can be implemented. It can be done in form of a group of simple scenarios or as one complex one, showing many possible user mistakes and their resolution. The last one approach reveals the complexity of the inspection process, hence it seems more beneficial. The usage scenario is presented in Table 11.

Role	Carried out Activities
admin	1) Administrator creates a campaign from a spreadsheet that contains staffing information, that does the following: • Grants roles to users according to staffing information (if user is not found, then it will be created). • Creates inspections, belonging to the campaign. 2) Administrator adds a new inspection participant that was not mentioned in the excel file from user lists. 3) Administrator removes incorrect participant from the inspection process. 4) Administrator reassigns roles in the inspection due to change in participants.
moderator	1) After establishing inspection guidelines and deadline, changes status of inspection to ‘upload’. 2) Looks at uploaded files: • Finds that one artifact is inappropriate. • Deletes inappropriate content from the inspection. 3) Writes chat message notifying author that the artifact will not be inspected and therefore it was deleted. 4) Changes status of inspection to ‘prepare’. 5) Looks at remarks that were made. • Notices a duplicate in remarks. • Deletes the duplicates. • Uploads remarks from a spreadsheet. 6) Changes status of inspection to ‘rework’. 7) Ensures that every issue is addressed in reloaded artifacts and changes the status of inspection to ‘finished’.
inspector	1) Downloads artifacts. 2) Screens artifacts and notices that one file is not an artifact expected. Writes a message in chat in order to notify the author. 3) Downloads corrected artifact. 4) Writes remarks considering inspected documents.
author	1) Uploads several artifacts. • After an artifact uploaded a participant finds a mistake that wrong file was uploaded and writes a message in chat. Author edits the artifact and changes file to a proper one. 2) Deletes one artifact that he thinks should not be inspected. 3) Writes couple of messages to chat, making comments to what he uploaded. 4) Refreshes required documents during rework.

Table 11: FIT usage scenario

2.4. *Inspection Process with FIT*

Business process reengineering is inevitable after an introduction of a new support system and formal inspection is not an exception. The participant's point of view on the inspection process with FIT support is presented in Figure 11. It is important to notice that in comparison with initial inspection process (shown in Figure 4) the inspection process stages were reconsidered.

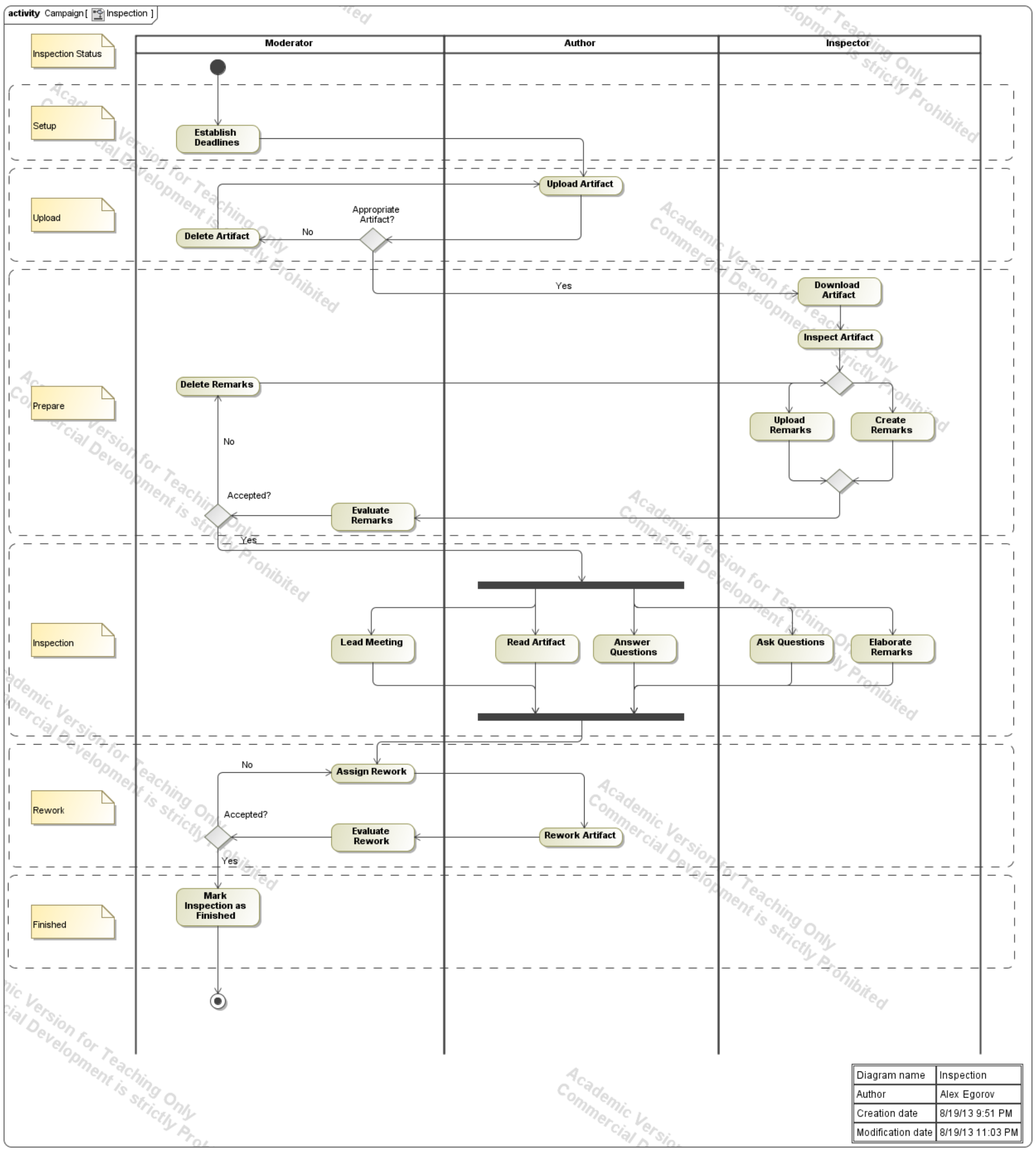


Figure 11: Inspection Process with FIT

There are seven inspection stages in the new process. Setup status is assigned straight after the inspection process was initialized by an administrator. After the moderator established guidelines and reconsidered the default deadlines, Upload inspection stage is reached. During the Upload stage the author uploads artifacts and the moderator checks if they satisfy the requirements, afterwards the Prepare inspection status is established. The inspector downloads and inspects uploaded artifacts, later he either uploads his remarks from a spreadsheet file or creates remarks directly. The moderator ensures that there are acceptable, otherwise he deletes inappropriate content. During the inspection the physical meeting of the inspection participants occurs. The author reads an artifact aloud and answers on the questions that were asked by the inspector. The inspector elaborates remarks and asks the questions regarding the artifact. The moderator leads the meeting and makes notes.

Basing on the results of the inspection meeting the moderator identifies what should be revised by the author in the Rework stage. The author reworks the artifacts and then after the moderator has accepted of the rework the Finished stage of inspections is set.

2.5. *Business Processes and Business Logic*

The campaign creation process is probably the first process and the most complex that will be executed by the administrator. The process is shown in Figure 12.

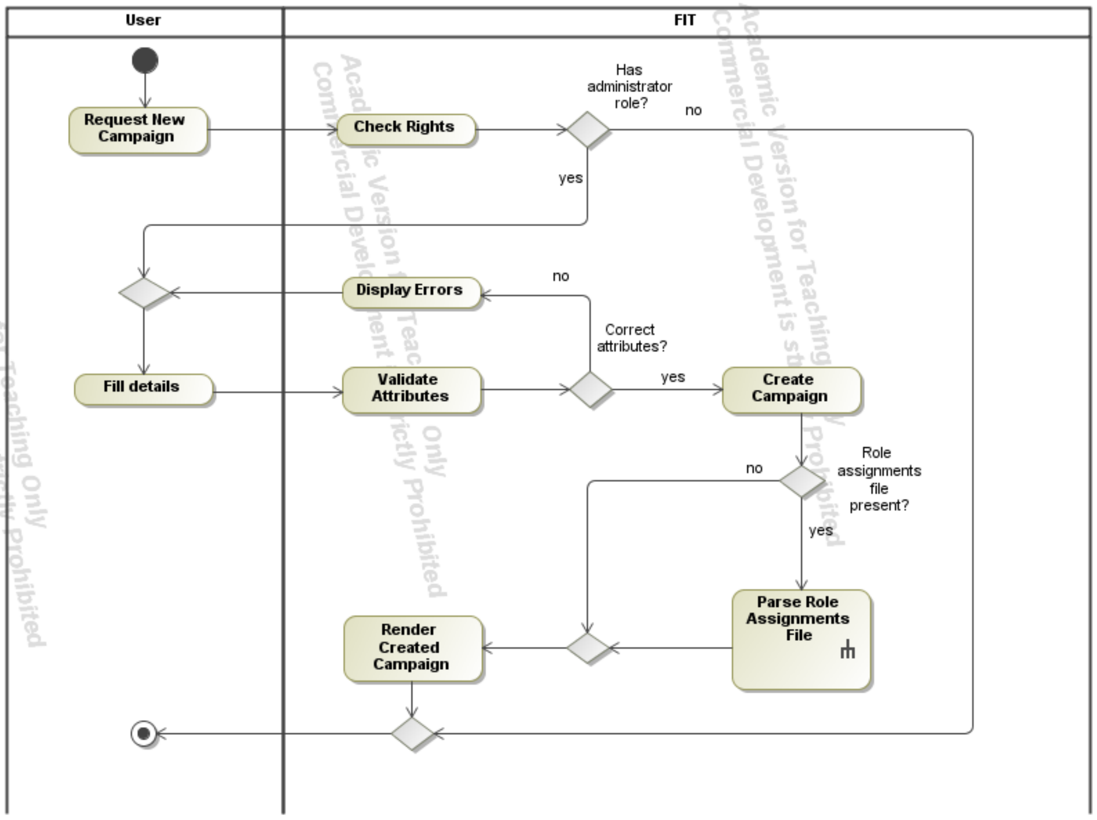


Figure 12: Campaign creation process

If the administrator has a spreadsheet file with role assignment information, then he would like to include the file into campaign creation process. By specifying role assignment file the administrator wants the system to create a group of inspections and assign role to user according to staffing information mentioned in the file. The Parse Role Assignments process displayed on Figure 13 explains how it should be carried out.

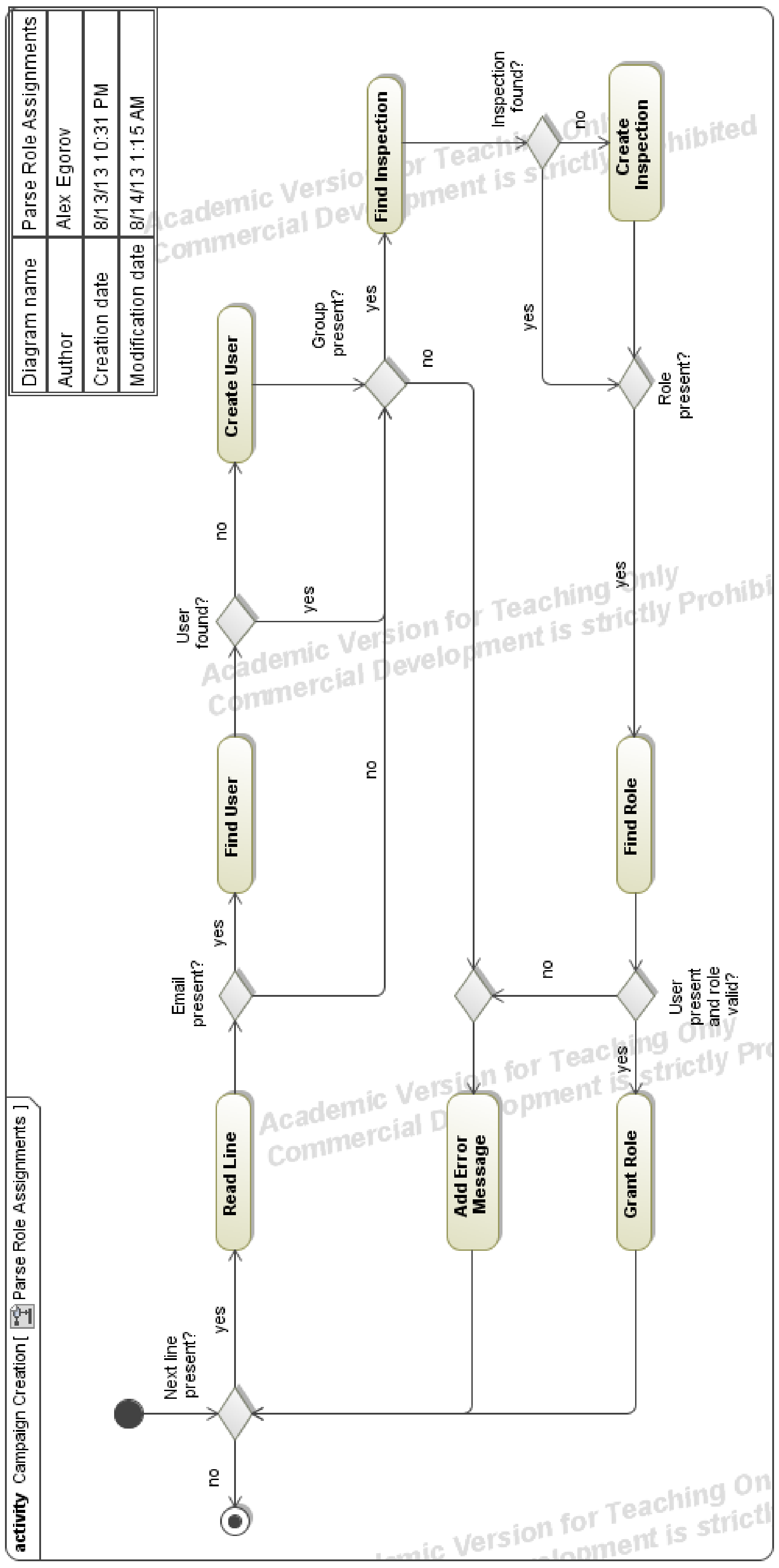


Figure 13: Role assignments file parsing process

2.6. *Information Model*

The information model consists of three diagrams: overview on information model, detailed information model, and a diagram of information model types. The overview of the information model of the FIT tools is presented in Figure 14.

The inspection is an object that represents a Fagan-style inspection process. The inspection can have six deadlines that correspond to inspection statuses: `Setup`, `Upload`, `Prepare`, `Inspection`, `Rework`, and `Finished`. A campaign unites a group of inspections that are conducted at the same time by different groups of users. Users communicate with each other via chat through writing chat messages. The chat is a messaging subsystem that is individual for every inspection.

An artifact is a representation of the existing piece of documentation that is to be reviewed during the inspection process. The user with appropriate rights is able to create an artifact instance via uploading a file. The artifact belongs to both user and inspection. The artifact must belong to one and exactly one inspection by grounds of flexibility and independence of inspections as well as provide modularity for the system, since it is much easier to control many-to-one relationship than many-to-many relationship. The main argument to implement many-to-one relationship between elements is that a consideration that a typical user participates in one inspection at time, thus allowing the artifact to belong to many inspections does not bring much sense.

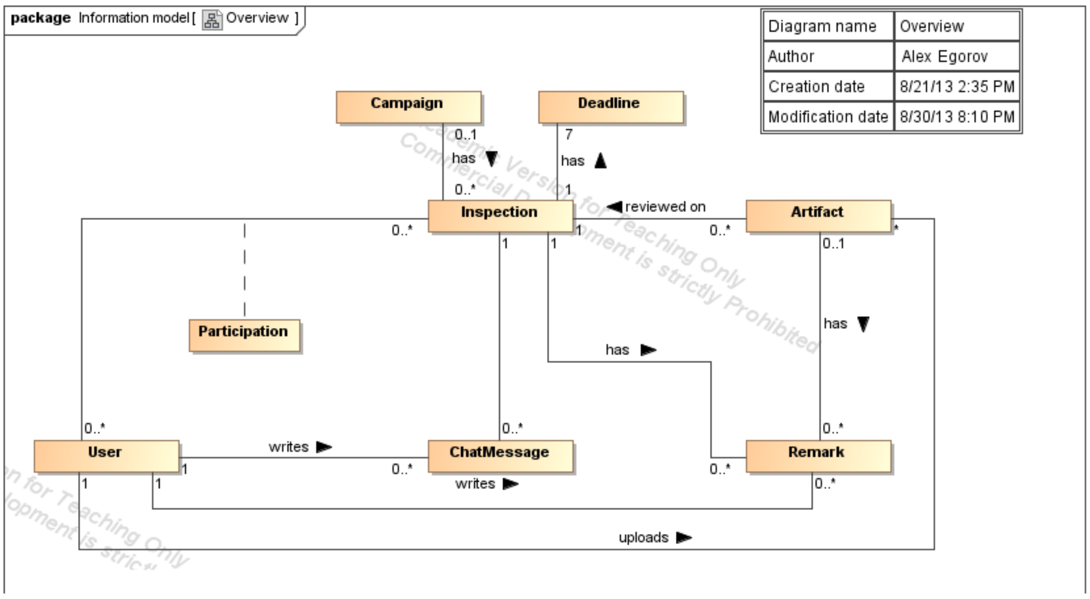


Diagram name	Overview
Author	Alex Egorov
Creation date	8/21/13 2:35 PM
Modification date	8/30/13 8:10 PM

Figure 14: FIT information model overview

A `remark` is a user comment on the `artifact`; therefore it can belong to only one `artifact` at the same time, while the `artifact` can have a multiple `remarks`. Moreover the `remark` can refer to a general comment regarding the `inspection`, thus it will belong to the `inspection`, not to the `artifact`. The detailed information model is shown in Figure 15, while the data types are presented in Figure 16. Most of information items have basic CRUD operations that are omitted in order to remove unnecessary details from the diagram.

`Deadline_missed?()` function allows to check if the deadline is missed. The possibility to upload remarks from a spreadsheet file is supported by `create_from_spreadsheet()` function. In order to help the administrator to determine inspections that has incomplete or incorrectly staffed teams Inspection class have `team_valid?()` and `team_complete?()` functions. The administrator can provide role assignments spreadsheet file to `create_from_role_assignment_spreadsheet()` that will automatically create stated amount of inspections, assign roles to users, and if necessary it will create users.

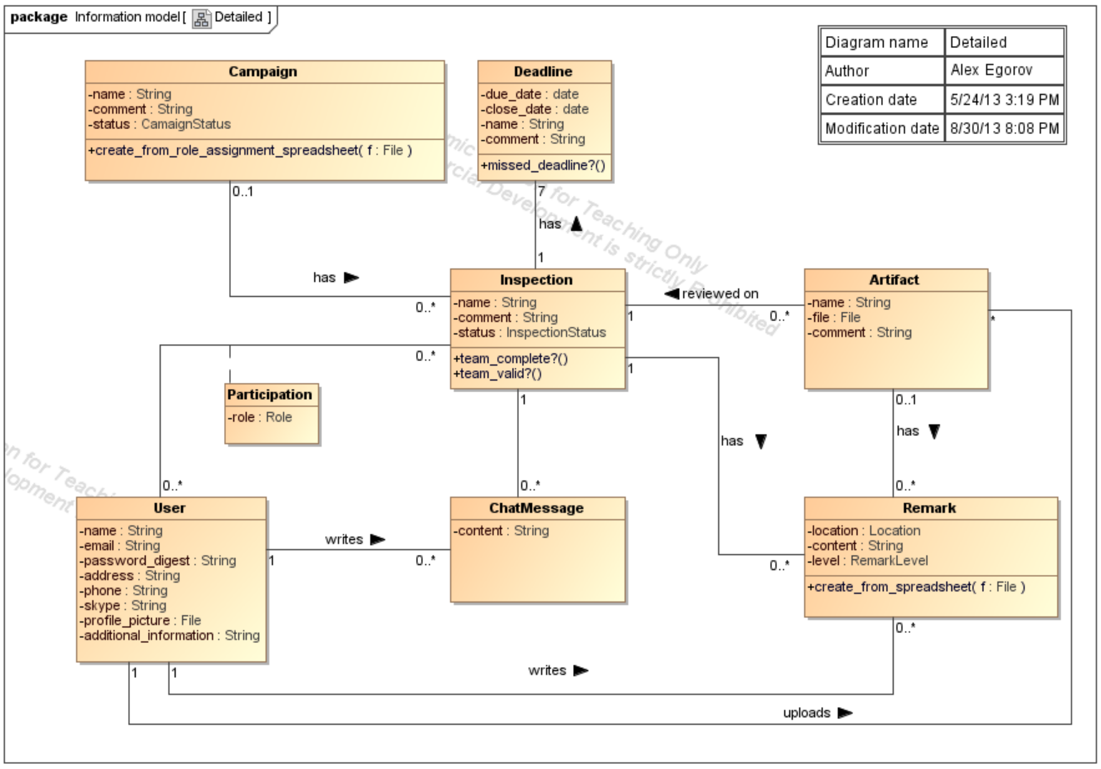


Diagram name	Detailed
Author	Alex Egorov
Creation date	5/24/13 3:19 PM
Modification date	8/30/13 8:08 PM

Figure 15: FIT information model details

`Location` is a very important field of Remark class and should be described additionally. Location field have an abstract type Location, which can be implemented by one of three concrete classes: `LocationDocument`, `LocationCode`, `LocationModel`. If the artifact is a text, spreadsheet or similar file type than location field of a corresponding remark should have `LocationDocument` type. In this case the element of documentation to which the remark is referencing would have such type as page, figure, table, line, etc. For the `LocationDocument` type presence of `element_type` and `element_name` is required, while `element_number` field is optional. When the artifact is a piece of code the `locationCode` type is applied, which describes to what line of code the remark is pointing.

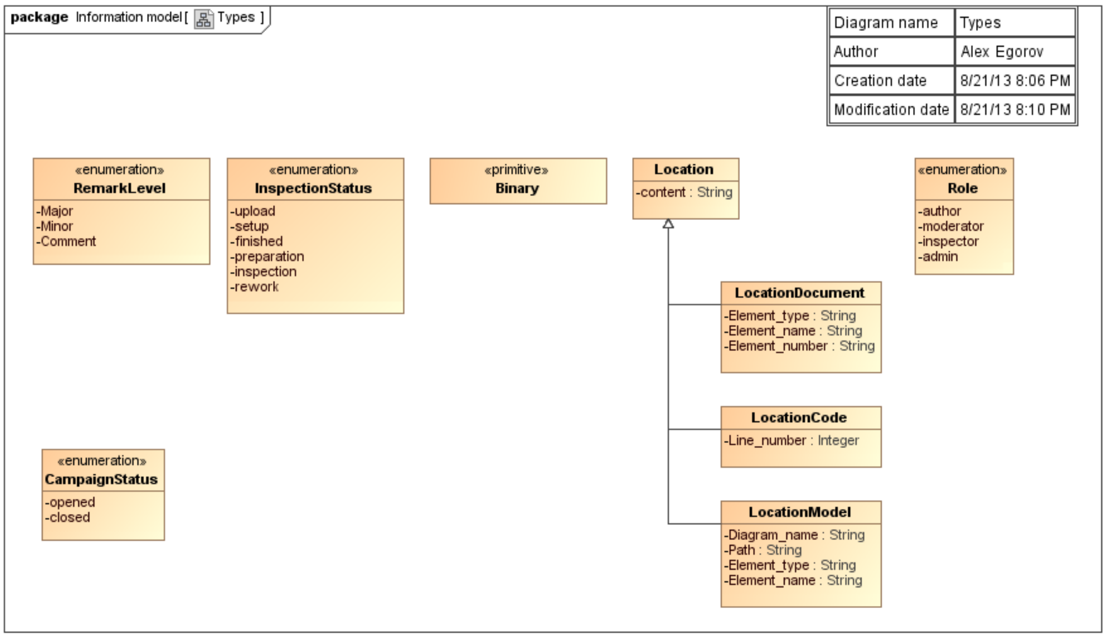


Figure 16: FIT information model data types

The last possible type of location is a `LocationModel` that is applied if the artifact is a model, e.g. UML model. There are three possible combinations in this case: `element_type` and `element_name` should not be empty, `path` to the referenced object (element of the model) should present or `diagram_name` is expected to be filled out.

3 System Design

3.1. *Technology choice*

There are many interesting technologies available on the market that can be used for the development of the FIT system. These technologies include: Django based on Python, Ruby on Rails based on Ruby, Spring based on Java, .Net MVC based on C#. Since all mentioned frameworks are equally good, well known and widely used, the choice process mostly becomes a matter of personal preferences. The framework should be a full stack framework with possibility to encapsulate low-level interactions, since the author haven't had any previous experience with web development and of course with any of mentioned frameworks. Due to author's personal preference to open-source software, the other requirement for the framework is to be open-source product.

The author decided to choose Ruby on Rails framework, since it does satisfy the requirements, i.e. it is a full stack open source cross-platform framework for creating web applications[8]. Ruby on Rails has database abstraction layer that encapsulates low-level database interactions. Ruby on Rails is considered to be one of the best frameworks for building web application. As it is possible to conclude from the framework name it is based on Ruby programming language. Ruby on Rails ships with a sensible set of build in libraries and provide a well-proven, multilayer system form organizing program files and concerns (Carneiro & Al Barazi, 2010).

In order to provide smooth and easy user-machine interaction a front-end framework should be selected. Bootstrap framework is considered to be one of the most popular frameworks[9] on GitHub, not only because it is sleek, intuitive, and powerful front-end framework for faster and easier web development[10], but also because of its standardization of HTML syntax[11].

3.2. *Design Patterns*

Ruby on Rails framework is shipped with a couple of very powerful instruments that implement Active Record and Model View Controller patterns. While the usage of the first pattern is optional, the second one is mandatory since the whole Rails framework was built around MVC pattern.

Active Record

Active Record (AR) is a software architectural pattern named by Martin Fowler. The pattern is presented by an object that wraps a row in a database table, encapsulates access and adds domain-specific logic on the top of the data (Fowler, 2003). Active Record object presents an interface to underlying relational database table of view

[8] http://rubyonrails.org/

[9] https://github.com/trending?since=monthly

[10] http://getbootstrap.com/2.3.2/index.html

[11] https://medium.com/what-i-learned-building/99fdd6e46586

that describes the way the data from the database should be created, accessed, modified or deleted.

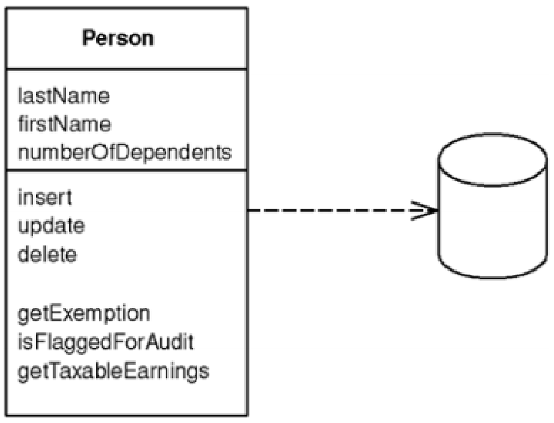


Figure 17: Active Record Example (Fowler, 2003)

The data structure of an Active Record class should correspond to the underlying database in the way that one field in the class is mapped to corresponding column in the table. The Active Record class incorporates wrapper methods for create, read, update and delete a SQL row, static finder methods to wrap commonly user SQL queries and return AR objects, field getters and setters, business logic (Fowler, 2003).

In what follows further when Active Record is mentioned the concrete implementation of the pattern in Ruby on Rails should be assumed.

Model View Controller

Model View Controller (MVC) pattern often used in UI frameworks[12]. There are three roles in the pattern: model, view and controller. Figure 18 presents MVC interaction.

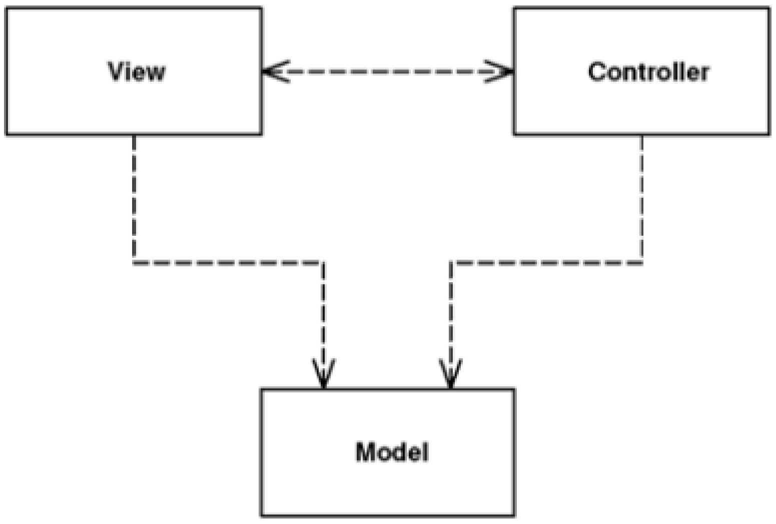


Figure 18: MVC Interaction (Fowler, 2003)

[12] http://martinfowler.com/eaaDev/uiArchs.html#ModelViewController

The domain information is represented by the model. The user interface is represented by the view that only responsible for displaying the information, while the controller is triggers the view update after the model manipulation in response to user input.

There are two principal separations in MVC: the model is separated from the presentation and the controller is separated from the view (Fowler, 2003). The key reason for separating the view from the model is that the presentation depends on the model but no otherwise. Creating presentation is mostly about creating a good and convenient user interface, while model creation implies consideration of business policies, database interactions. The main advantage of separating the model and the view is a possibility to alter the presentation without making changes in the model, i.e. it is possible to provide completely different interfaces to the same model.

The separation between the controller and the view is less important, the support of editable and not editable behavior is one the classic examples that motivates the separation (Fowler, 2003).

Considering a web site, the controller main responsibilities are presented in Table 12.

Responsibility	**Description**
URL decoding	The controller extracts data from a form and determines an appropriate action.
Model Invocation	The controller creates and invokes an object model to process the data.
View Determination	The controller determines which view should be displayed as the result of the user request and forwards the model information to the view.

Table 12: Controller responsibilities

In what follows further when model, view or controller is mentioned it should be understood in terms of Ruby on Rails as a concrete implementation.

3.3. *Architecture*

This section describes model architecture. The model in MVC is an object that represents the information about the domain and contains all the data and behavior other than the one used in the UI (Fowler, 2003). Model dependency diagram is used to represent model architecture. It is an elaboration on the information model (presented in Figure 15). In Ruby on Rails framework "Fat Model, Skinny Controller"[13] is a common practice. It means that all logic should be put into the model and the controller becomes just an interface between model and its graphical representation – view[14].

Looking at the model dependency diagram presented in Figure 19 it is possible to notice a few significant changes in comparison to information model. One of the most important ones is an introduction of Participation class between Inspection and

[13] http://www.sitepoint.com/10-ruby-on-rails-best-practices/

[14] http://guides.rubyonrails.org/layouts_and_rendering.html

User classes instead of many-to-many relationship with association class between Inspection and User. This change was implemented due to Active Record limitations in case of supporting many-to-many relationship. There are two possible different ways of implementing many-to-many relationship between models with Active Record in Rails: `has_and_belongs_to_many` association[15] and `has_many :through` association. The first one creates a direct many-to-many connection between the selected models without introducing an intervening model, but nonetheless the joining table should be explicitly specified. The second possibility involves creation of a join model. If validation is needed, that join model method is recommended for use. Nevertheless the main difference between those two methods is that in `has_and_belongs_to_many` association the underlying SQL join table will be created without primary key.

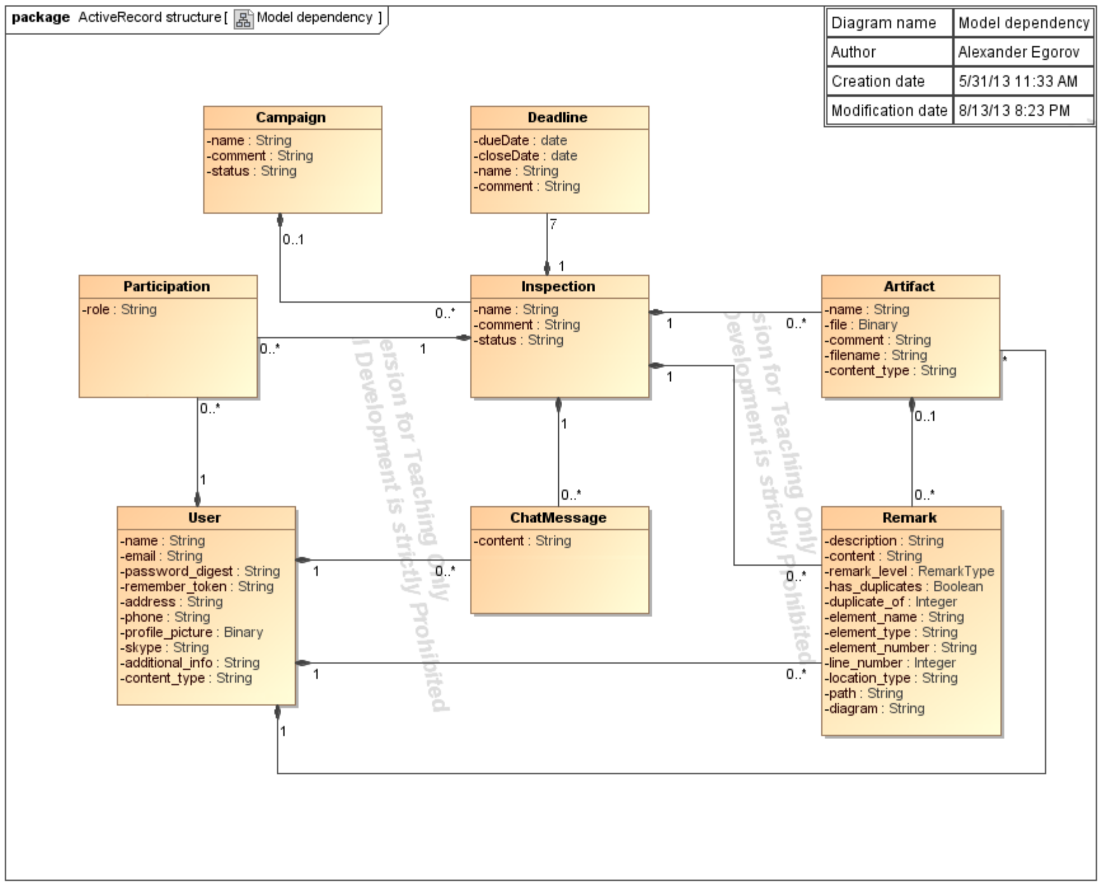


Figure 19 Model Dependency

There is a necessity to introduce validation on `User-Inspection` relationship, due to concerns identified in inspection process (There should be no more than one user playing author role participating in the inspection process). Therefore the second option of implementation many-to-many relationship was chosen. Also `has_many :through` association since it assumes more meaningful model name[16], i.e. `Participation`, that is rather self-explanatory. Taking into account

[15] http://guides.rubyonrails.org/association_basics.html

[16] http://railscasts.com/episodes/47-two-many-to-many

Rail conventions, the join table in case of `has_and_belongs_to_many` association between `Inspection` and `User` models should be named either `inspections_users` or `users_inspections` and that name can be confusing.

Mapping the `location` field of the abstract type `Location` of `Remark` class, which described at information model[17], directly to a group of fields with primitive types, such as `String` and `Integer` was another important decision. There are several possibilities of how to implement non-standard data type filed within Rails model[18]. Those techniques are presented in Table 13.

	Technique Description	**Advantages**	**Disadvantages**
Object serialization	There is a standard module Marshal in Ruby that serializes object to a string[19], which is later can be effectively translated into byte array. The result of marshaling is stored in the database as a standard type.	Relatively easy technique from configuration point of view.	Limited searching capabilities on the key values.
NoSQL datastore integration	The custom field is stored in NoSQL database.	Effective.	High configuration and integration cost.
Dynamic columns	Dynamically add columns to the Inspection table when concrete type of location is defined.	Allows strict type validation.	Difficulties in maintaining standard object-relational mapping.
PostgreSQL extensions	Use PostgreSQL extensions such as `hstore`[20]. The extension mixes stores sets of key/value pairs within a single PostgreSQL value. Keys and values are simply text strings.	Useful in various scenarios in case of semi-structured data or if the attributes are rarely examined[21].	Applicable only for certain Ruby standard types, such as `Hash`.

[17] Figure 15: FIT information model details

[18] http://blog.artlogic.com/2012/09/13/custom-fields-in-rails/

[19] http://www.ruby-doc.org/core-2.0/Marshal.html

[20] http://www.postgresql.org/docs/9.2/static/contrib.html

[21] http://www.postgresql.org/docs/9.2/static/hstore.html

Custom type table	Design a standard relational mapping to a new custom type table that has all required fields.	Widely used solution, which fits standard relational database management system schema.	Increase in the number of objects that needed to be managed.
Additional fields in the model	Mapping custom type fields to the fields in the underlying model, i.e. add series of `custom_field_#{n}` fields into the underlying table.	Straightforward technique.	Adds a great overhead for a custom type with many fields that may not be used.

Table 13: Comparison of custom field implementation techniques in Rails model

Taking into account that the strict type validation and efficient search for `Remark` location is needed, the first and the third techniques, i.e. object serialization and dynamic columns fell off. PostgreSQL extensions are also not appropriate in case of `Location` type, since it is well-structured type and its attributes are often examined, e.g. in case of uploading remarks from file. NoSQL datastore integration is effective, but not efficient for the `location` field, due to its extremely high integration and configuration overhead.

The last two techniques seem to be equivalent in implementation cost and they both seem to be appropriate. Custom type table techniques assumes three additional tables to the database, `Location` type is an abstract type and three are three concrete types that is `LocationDocument`, `LocationCode`, and `LocationModel`[22]. In case of using the last method six additional fields should be added to the Remark model.

Considering uploading remarks from a spreadsheet file as a critical functionality that would likely to be used often[23] and providing appropriate level of fault tolerance is critical, last technique was chosen. It hands over a simple solution for importing remarks from a spreadsheet that has incorrectly filled location cells. The solution is just to import all fields, construct the remark object, then determine the location type and just ignore incorrectly filled location cells.

`RemarkLevel`, `InspectionStatus`, `CampaignStatus` and `Role` classes can be easily implemented with such a powerful Active Record instrument as callbacks that hook into the life cycle of an Active Record object and trigger logic before or after an alteration of the object state.[24] The classes mentioned above implemented as a string that corresponds to regular expressions.

[22] Figure 15: FIT information model details

[23] Table 10: Upload remarks use case

[24] http://api.rubyonrails.org/classes/ActiveRecord/Callbacks.html

3.4. *Chat Capabilities*

Instant messaging capabilities in the web application assume some kind of mechanism to obtain updates from server. There are several technologies that can provide such capabilities, one of them are `Web-Sockets`, `Long Polling`, and `Server-Sent Events`.

Table 14 below demonstrates the comparison between mentioned above technologies.

	Long-polling	**Server-Sent Events**	**WebSockets**
Browser support	Supported by the most of currently use d browsers.	Chrome 9+, Firefox 6+, Opera 11+, Safari 5+.	IE 10+, Firefox 7+, Chrome 14+, Safari 5+, Opera 12+.
Server load	Consumes a little of CPU resources, closes connection every time the event is sent.	Requires a very limited amount of resources, does not need to close connection after response event is sent.	Requires a very limited amount of resources, but requires constantly maintained HTTP connection.
Client load	Depends on implementation.	Consumes minimum resources since native implementation in the browser.	Consumes minimum resources since native implementation in the browser.
Timeliness	Depends on implementation.	Three seconds delay by default settings.	Real time.
Implementation Complexity	No additional gem required.	Additional Rails gem required.	Requires additional infrastructure, i.e. EventMachine server with custom port being open.

Table 14: Chat update technologies comparison[25]

After comparing the solutions it looks like `WebSockets` is the best solution, since it requires minimum resources from both client and server, followed by `SSE` and `Long-Polling` looks like the worst decision to implement it terms of consumed resources. But what is important to consider is that browser support is very important due to large share of users still use Internet Explorer and old versions of old browsers[26]. According to W3Counter statistics on July 2013 there is still more

[25] http://dsheiko.com/weblog/websockets-vs-sse-vs-long-polling

[26] http://www.w3counter.com/globalstats.php?year=2013&month=7

than 22 percent of users utilize Internet Explorer and there is only 7 percent that use IE 10 that natively supports `WebSockets` technology.

`Server-Sent Event` technology is not supported by any version of Internet Explorer and that `WebSockets` technology is not supported by approximately 15 percent of users in addition to requirements for maintaining constant connection and having additional infrastructure[27], therefore Long-polling technology should be used.

Sequence diagram presented in Figure 20 demonstrates how Long-polling technology works.

Every chat message is wrapped in HTML element with data attribute `data-id` that corresponds to the underlying `ChatMessage id` value. ChatMessage view uses AJAX calls and sends the attribute value to the ChatMessages controller. The controller asks ChatMessage model if there is any message with `id` bigger than the received one from the view. If there are no new messages the controller sends nothing to the view, otherwise it sends missing messages. The view in its turn on the reception of the new messages appends them to the chat.

[27] http://www.html5rocks.com/en/tutorials/websockets/basics/

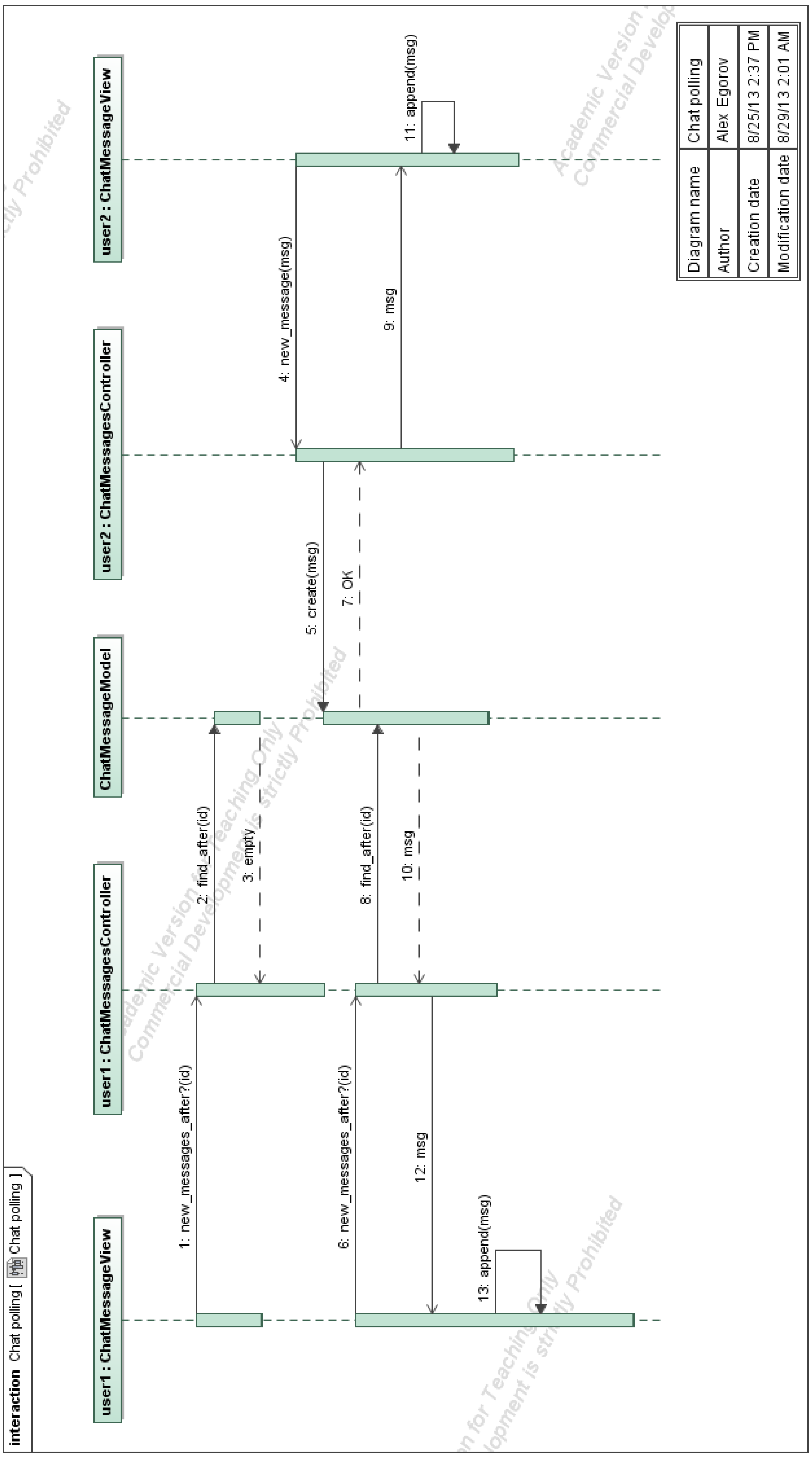


Figure 20: Chat long-polling

3.5. *User Interface*

Sketch

Sketching user interface using simple tools like pen and paper or marker and whiteboard is well-recognized solution for the very first round of product development since it is easy and cheap solution (Störrle, Requirements Engineering, 2012). The result of the sketching has been photographed; the inspection list is presented in Figure 21, opened inspection is presented in Figure 22 and the remark table is presented in Figure 23.

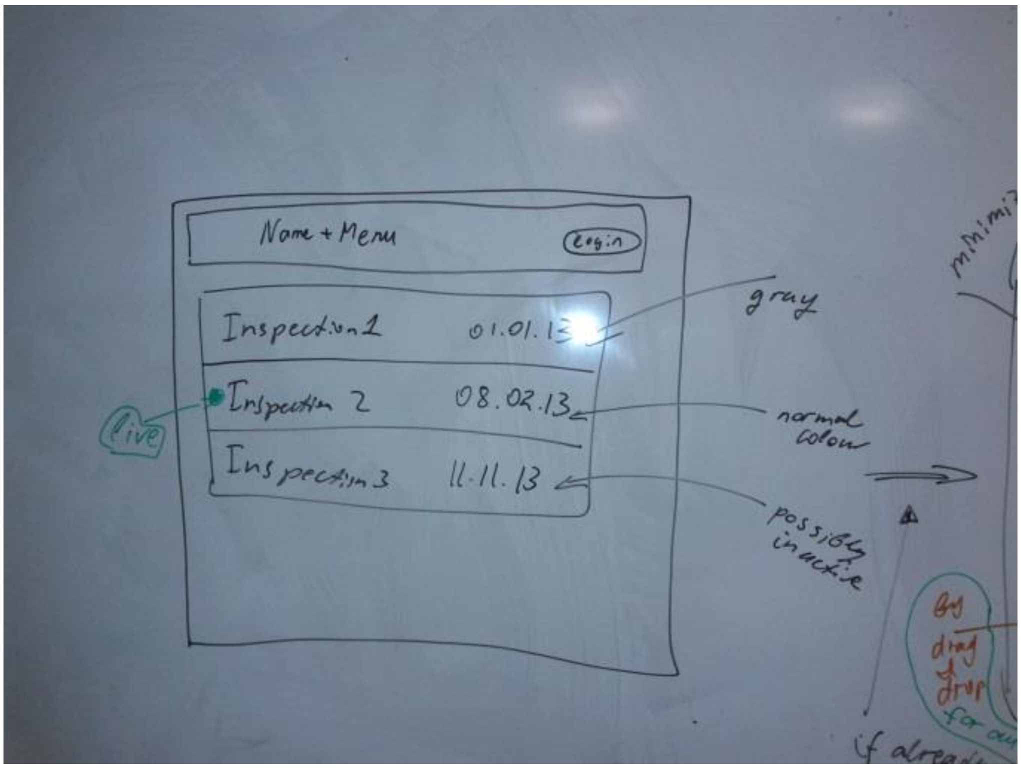


Figure 21: Inspection list sketch

There are several methodologies to substitute desktop's right-click menus in Web application such as Always-Visible and Hover-Reveal Tools that are united under the Contextual Tools name (Scott & Neil, 2009). As it is possible to notice from the options names, the first ones place contextual tools directly in the content, while the second one show tools on mouse hover.

During the sketching process the Hover-Reveal pattern was used.

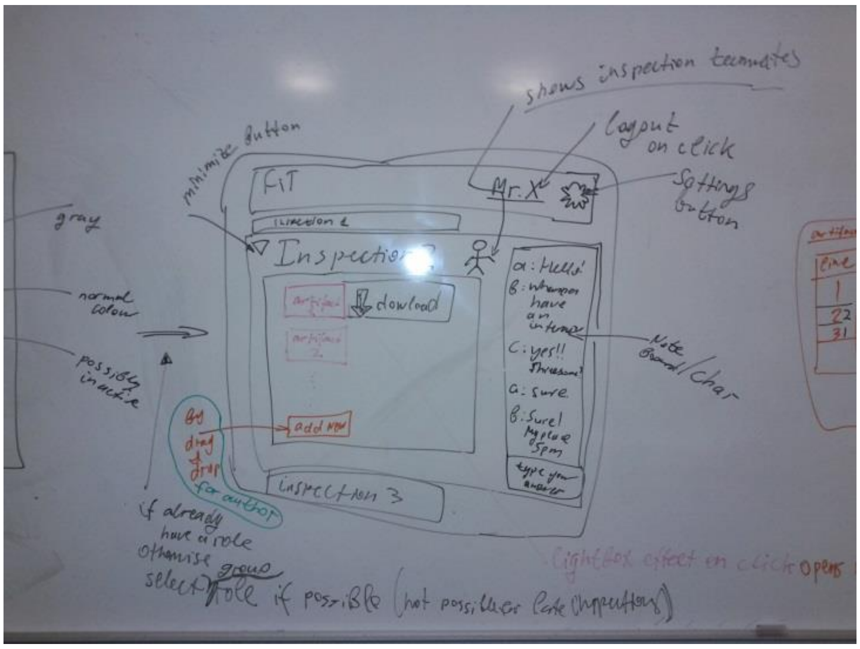


Figure 22: Current inspection sketch

The initial idea was to present user a list of inspection as a block with nearest deadline date, showing also inspections that are not available, i.e. either finished or announced. The interaction assumed that when a user clicks on an active inspection it will expand on the whole screen, showing the content and the inspection chat. The remarks table initially supposed to be shown only when user clicks on the artifact.

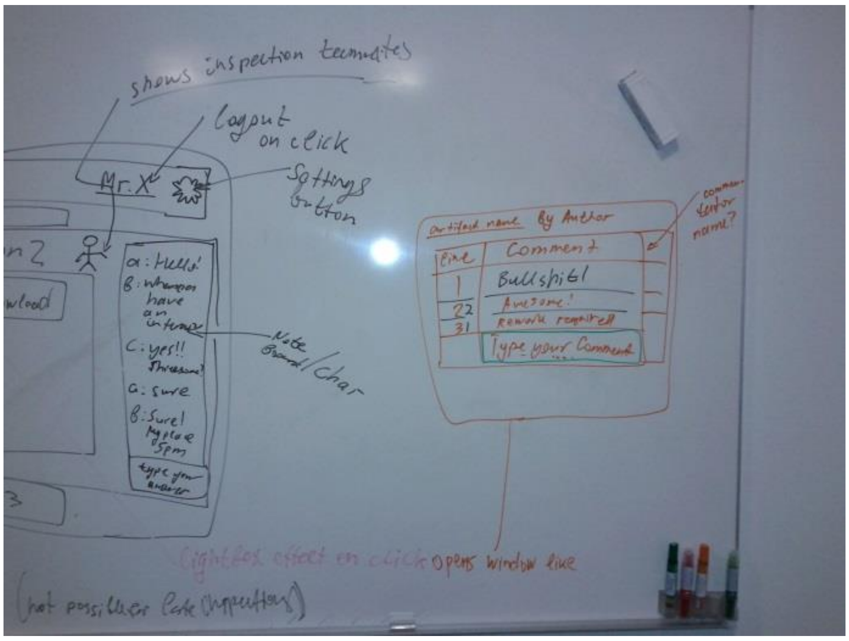


Figure 23: Remarks table sketch

Prototype

During the next stage of user interface development, a prototype was created. During the prototype implementation it was decided to change the list of inspections from a group of boxes to a carousel, since the expansion of the block looked too heavy and distracted the attention and was embarrassing especially if a wrong inspection box was clicked.

Figure 27 demonstrates the implementation of the whiteboard sketches made in Sublime Text 2 with Bootstrap library. The modal presents the PDF document displayed on the right and the remarks table with comments corresponding to this artifact. After clicking on artifact box the modal is displayed as it is demonstrated in Figure 27.

As it is possible to notice from Figure 22 most of the artifacts boxes and inspection boxes does not have any actions such as download, edit or delete being displayer, because Hover-Reveal principle was utilized and the actions were displayed only on mouse hover as it is presented in Figure 24. In order to show the design concept with actions the inspection view on Figure 25 has Artifact 2 and Inspection 2 with actions being forcefully displayed. That was impossible in the prototype to have more than one hover event at the time.

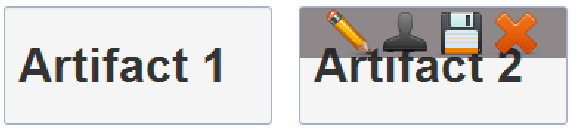
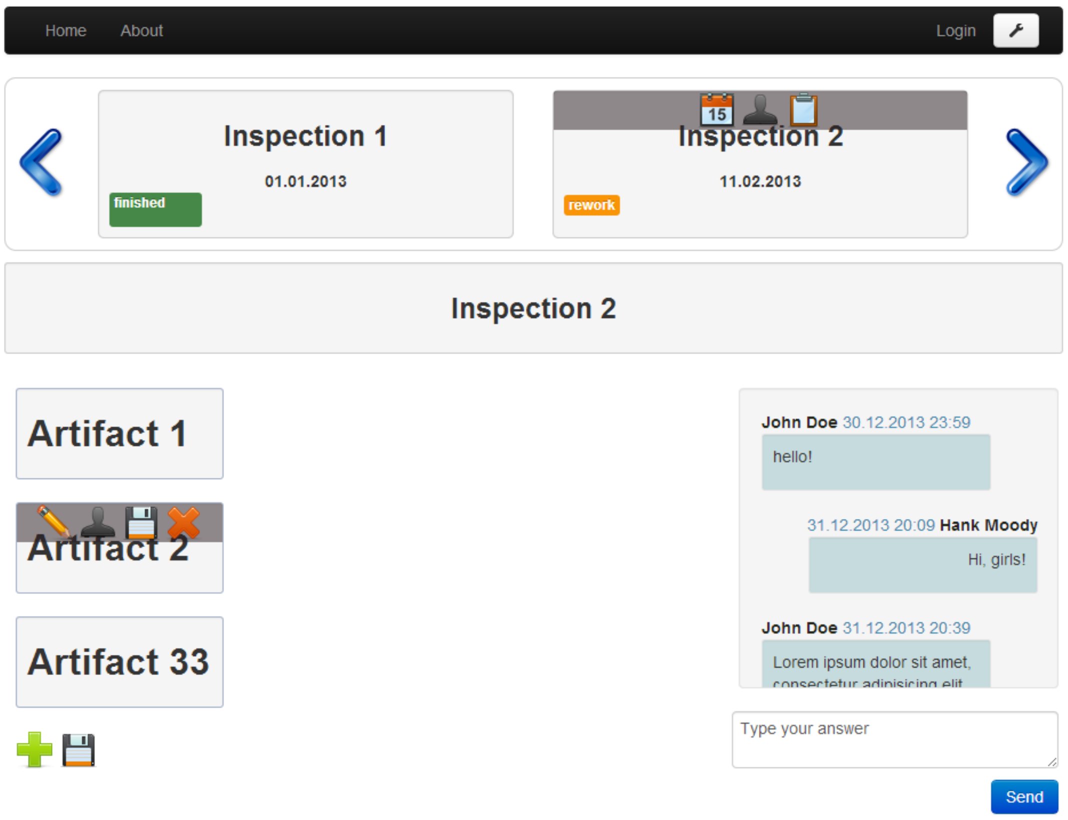


Figure 24: Panel with actions hidden (left) and displayed on mouse hover (right)



John Doe 30.12.2013 23:59
hello!

31.12.2013 20:09 Hank Moody
Hi, girls!

John Doe 31.12.2013 20:39
Lorem ipsum dolor sit amet, consectetur adipisicing elit

Figure 25: Inspection view prototype

Taking into account that only the current status of the inspection and only the closest deadline were shown in the inspection box, it was hard for unprepared user to understand the inspection process work flow. Therefore it was decided to implement some kind of a flow chart that would bring an understanding of the inspection process for unprepared user. Figure 26 shows the inspection status bar prototype.

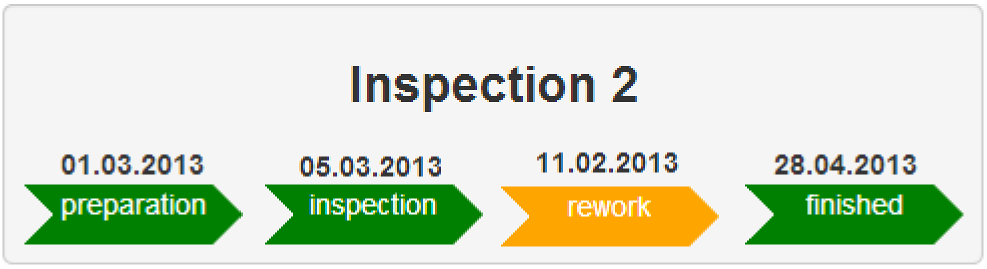
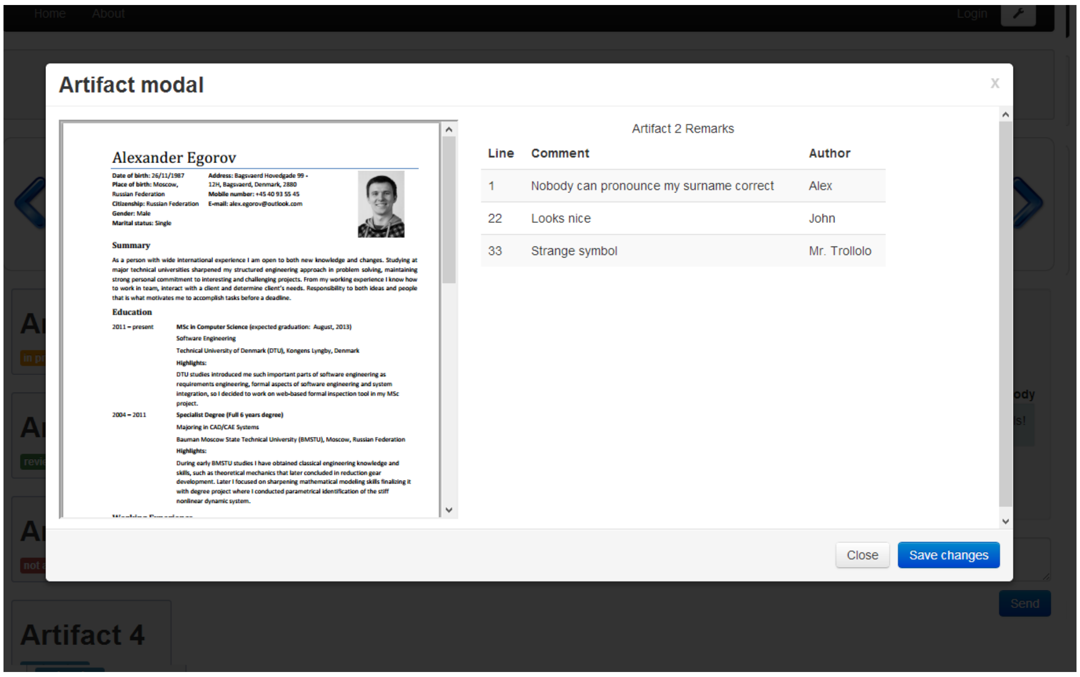


Figure 26: Status bar prototype



Alexander Egorov

Date of birth: 26/11/1987
Place of birth: Moscow, Russian Federation
Citizenship: Russian Federation
Gender: Male
Marital status: Single

Address: Bagsvaerd Hovedgade 99 - 12H, Bagsvaerd, Denmark, 2880
Mobile number: +45 40 93 55 45
E-mail: alex.egorov@outlook.com

Summary

As a person with wide international experience I am open to both new knowledge and changes. Studying at major technical universities sharpened my structured engineering approach in problem solving, maintaining strong personal commitment to interesting and challenging projects. From my working experience I know how to work in team, interact with a client and determine client's needs. Responsibility to both ideas and people that is what motivates me to accomplish tasks before a deadline.

Education

2011 – present	MSc in Computer Science (expected graduation: August, 2013) Software Engineering Technical University of Denmark (DTU), Kongens Lyngby, Denmark Highlights: DTU studies introduced me such important parts of software engineering as requirements engineering, formal aspects of software engineering and system integration, so I decided to work on web-based formal inspection tool in my MSc project.
2004 – 2011	Specialist Degree (Full 6 years degree) Majoring in CAD/CAE Systems Bauman Moscow State Technical University (BMSTU), Moscow, Russian Federation Highlights: During early BMSTU studies I have obtained classical engineering knowledge and skills, such as theoretical mechanics that later concluded in reduction gear development. Later I focused on sharpening mathematical modeling skills finalizing it with degree project where I conducted parametrical identification of the stiff nonlinear dynamic system.

Artifact 2 Remarks

Line	Comment	Author
1	Nobody can pronounce my surname correct	Alex
22	Looks nice	John
33	Strange symbol	Mr. Trollolo

Figure 27: Remarks table and artifact overview during the implemented of the prototype

Working Prototype

The general ideas of the prototype were accepted, but during the user interface design working prototype stage a few major changes were implemented. Hover-Reveal pattern was changed to Always-Visible patter. The main issue for Hover-Reveal tools is providing proper discoverability of the additional functionality, therefore, taking into account that the majority of the system users are students enrolled for courses similar to 02264 Requirements Engineering, they are likely would see the system for the first time and would use it only a few times, hence it is important that users could immediately understand what possible actions are available. Thus for the sake of discoverability Hover-Reveal were reconsidered in favor of Always-Visible Tools (Scott & Neil, 2009). In order to reduce visual noise icons symbolizing the actions are rendered in as visually light manner, that is achieved by making them half-transparent. Figure 28 shows that after user hovers on the icon it becomes fully visible.

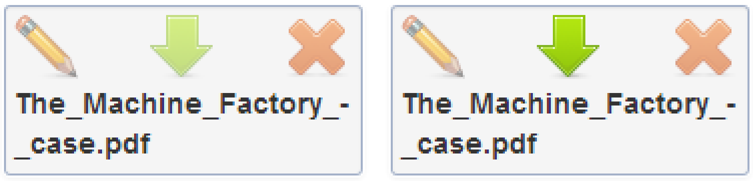
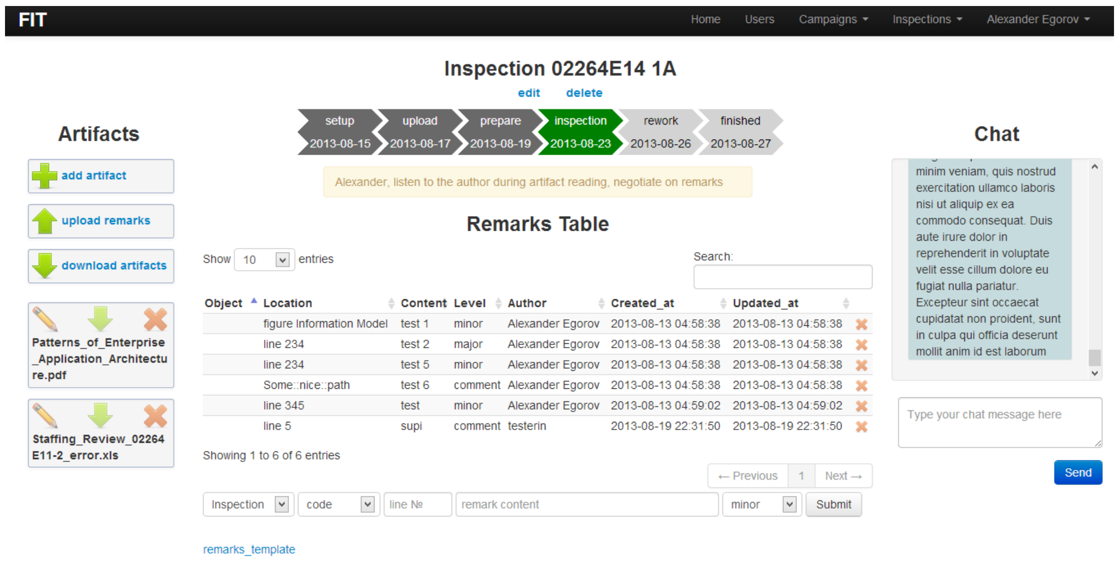

The_Machine_Factory_-_case.pdf
The_Machine_Factory_-_case.pdf

Figure 28: Artifact Action Panel rendered by default (left) and when download action is hovered (right)

Looking further at the artifact box it is possible to notice that the download icon was changed to a different one and upload user icon was removed.

The inspection view outline that is presented in Figure 29 was also changed. The status bar increased in size and incorporated all statuses from the inspection process adapted to the support system. Deadlines were modified in order to follow the flow created by the status bar. Considering the majority of users would participate at one inspection at time and in two inspections per semester it was decided to the inspection list from the carousel to a navigation bar menu item. Taking into account that the page space as a most valuable resource it is unwise to spend up to 20 percent (for standard 720p screen resolution) of it for a feature that would not be used often.

Since objective of the inspection process is to find defects in provided artifacts it is not wise to hide the instrument that presents possible defects, i.e. remarks table. Following Always-Visible tools principle it was decided to reveal remarks table and make it central element of the inspection page.



Inspection 02264E14 1A

Alexander, listen to the author during artifact reading, negotiate on remarks

Remarks Table

Object	Location	Content	Level	Author	Created_at	Updated_at	
	figure Information Model	test 1	minor	Alexander Egorov	2013-08-13 04:58:38	2013-08-13 04:58:38	✗
	line 234	test 2	major	Alexander Egorov	2013-08-13 04:58:38	2013-08-13 04:58:38	✗
	line 234	test 5	minor	Alexander Egorov	2013-08-13 04:58:38	2013-08-13 04:58:38	✗
	Some::nice::path	test 6	comment	Alexander Egorov	2013-08-13 04:58:38	2013-08-13 04:58:38	✗
	line 345	test	minor	Alexander Egorov	2013-08-13 04:59:02	2013-08-13 04:59:02	✗
	line 5	supi	comment	testerin	2013-08-19 22:31:50	2013-08-19 22:31:50	✗

Showing 1 to 6 of 6 entries

minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

Figure 29: Inspection final design

4 System Implementation

4.1. *Technology*

Ruby on Rails

Rails uses Ruby in order to create a domain-specific language. Rails utilizes several concepts and principles: `don’t repeat yourself (DRY)` principle, `you ait’t gonna neet it (YAGNI)`, and `convention over configuration` (Carneiro & Al Barazi, 2010).

`DRY` states that information in a system should be expressed in only one place. `YAGNI` assumes that only actually needed functions are implemented, i.e. do the simplest thing that could possibly work.

`Convention over configuration` is the most tangible principle in Ruby on Rails framework. The principle assumes that the developer needs to define only configuration that is unconventional. There are several naming conventions in Rails framework: the first one tells that class names should be name with `CamelCase`, methods and variables written with `snake_case`; a model should be named with a singular noun, e.g. `User` and it will be stored in `models/user.rb`, then the reciprocal controller should be named with a corresponding plural noun, i.e. `UsersController` that is defined in `controllers/users_controller.rb` file. Views that represent the model should be stored in `views/users/` folder, while stylesheets and javascript for those views shall be defined accordingly in `assets/stylesheets/users.css` and `assets/javascripts/users.js` files.

As it was mentioned earlier, Ruby on Rails implements Model View Controller pattern. A model typically represents a database table. Since the application uses SQL database the model is a subclass of `ActiveRecord::Base` class. All business logic should be implemented in models according to Rails convention.

A view in Rails is a HTML or a JavaScript template with embedded Ruby that is called ERB template. In general there are two types of views: a full-blow view and a partial view. The full-blow view typically rendered as a result of the controller’s action execution, e.g., in case an `index` action of `UsersController` is called, then by default the action will render view that is stored in `views/users/index.html.erb` file. The partial view (or just partial) is rendered inside other views and can be nested. According to the convention the partial should be denoted with underscore at the beginning of its name, e.g. `_status.html.erb`.

A controller is a subclass of an abstract `ApplicationController` class in Ruby on Rails framework. According to Rails convention controller’s public methods are called actions[28]. Depending on received URL the framework will determine which controller and which action should be called. After that the framework creates an

[28] http://guides.rubyonrails.org/action_controller_overview.html

instance of that controller and runs the action. By default the controller in Rails automatically renders the view that has the name of the action due to convention over configuration principle.

Version Control

Modern software development process assumes that it is virtually impossible to imagine any reasonable excuse for not using version control system. There are several alternatives present on the market nowadays such as Concurrent Version System (CVS), Subversion (SVN), Mercurial and Git. Taking into the account that the project is supposed to be an individual work any of previously mentioned version control tools are suitable. The main requirement the tool is that it should be an open-source project; it should have an implementation and free online repository. Git has been chosen as the version control system for the project since it is particularly powerful, flexible and low-overhead version control tool (Loeliger, 2009) and there are several web-based hosting services such as Github[29] and Bitbucket that provide free accounts for open source projects. Github has been chosen because the latest versions of many third parties Ruby on Rails libraries are accessible via Github repositories and it was built using Rails.

Integrated Development Environment

On the first stages of the development process Sublime Text 2 was used, taking into account that it is not an IDE the decision to switch on a different product had been made. The set of requirement was formed based on experience with Sublime Text 2. An appropriate IDE contains Ruby debugger, version control integration, Ruby Gems management system, code analysis and assistance capabilities as well as easy project configuration and HTML, CSS, and JavaScript editing. There are several such systems on such as TextMate, NetBeans for Ruby, JetBrains RubyMine and Aptana Studio. The first one solution is designed for Mac OS X and was turned down automatically, since the development was conducted under Microsoft Windows. NetBeans stopped Ruby support and starting from version 7.0 and higher Ruby support was removed[30]. Therefore NetBeans 6.9 for Ruby has been chosen, but it appeared that the IDE cannot properly import an existing project, which was generated by Ruby on Rails built-in generator, the IDE wiped out several important project files that fortunately did not lead to disastrous events due to version control system usage and regular commits to remote repository. Certainly the system usage was discarded and after reading an article[31], regarding JetBrains RubyMine was not initially considered as a good choice due to it is commercial origin that implies obtaining a license, it has been decided to try out 30 days trial version. After using the IDE for a couple of weeks without any crash, RubyMine proved that it is a very stable IDE that has all required features. Taking into account that FIT is an open-source project designed for

[29] https://help.github.com/articles/github-terms-of-service

[30] http://wiki.netbeans.org/RubySupport

[31] http://habrahabr.ru/company/JetBrains/blog/176891/

educational purposes it was possible to apply for a free educational license JetBrains RubyMine IDE.

4.2. *Code Structure*

It is reasonable to present code structure according to the MVC pattern. The model dependency, controller hierarchy and view hierarchy is presented in this subsection.

Models

The actual implementation of the model dependency from design stage is presented in Figure 30.

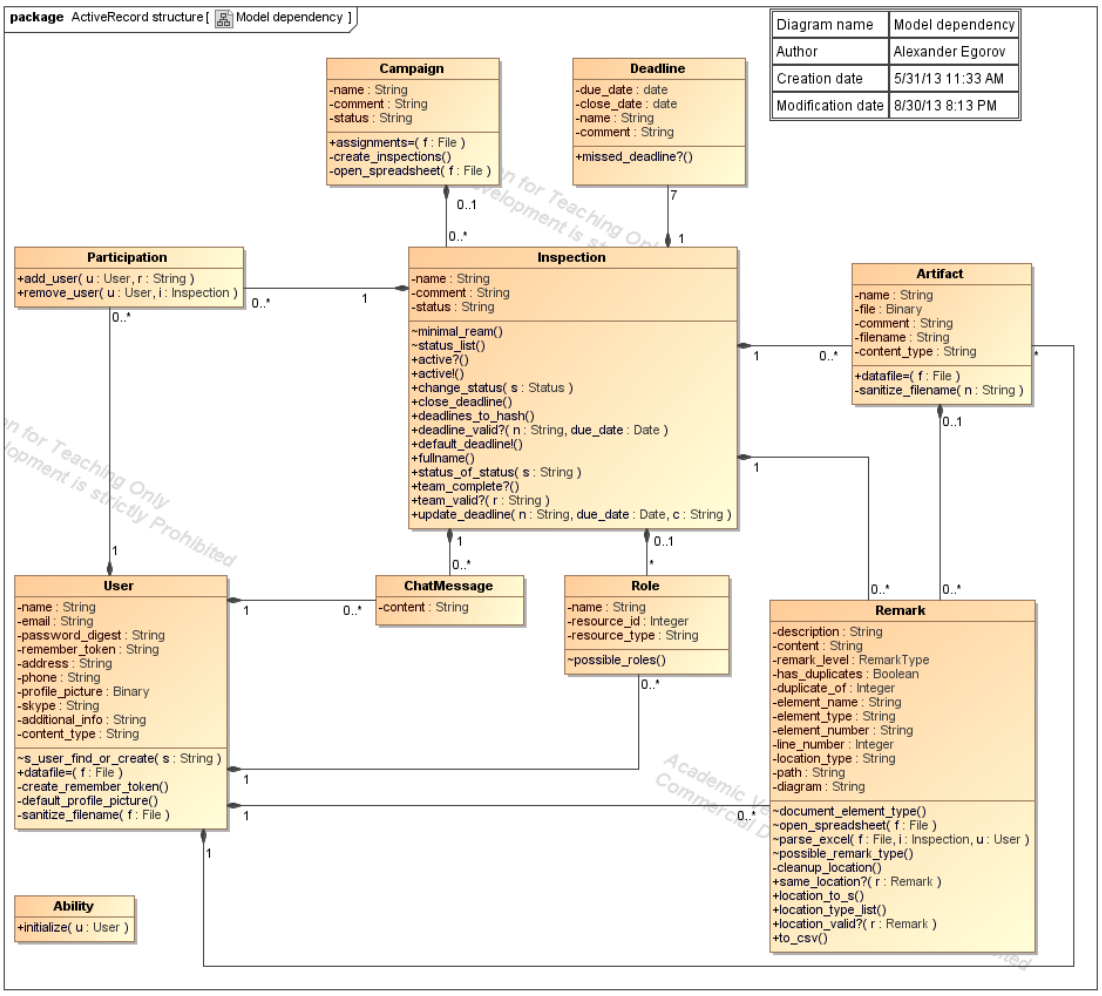


Figure 30: Model final design

`Ability` model is created by CanCan gem[32] that handles authorization. The model does not create a table in the database and it is used for describing the authorization rules.

`Campaign` model has `assignments=(f: File)` function that acts as virtual attribute in campaign form in the corresponding view. In other words when a simple

[32] Described in Libraries and Components on page 53

form file input field with conforming id and name attributes accepts a file `assignments=(f: File)` is called. The function is responsible for parsing role assignments file according to the business process described in Figure 13.

During the implementation an interesting bug of the Rails framework was discovered. Since the virtual attribute assumes that `assignments=(f: File)` function is called before the corresponding campaign is created, hence the campaign object is not presented in the underlying database table at this moment, thus it does not have an id. Taking into account that the campaign object has no id, then an inspection object that belongs to the campaign cannot be created, it can only be built, i.e. it will also not saved into the database until the campaign is saved. In case of a missing or corrupted value in role column is the role assignments file for the inspection that hasn't been built yet, the record in `Participation` table will not be created, since the conforming user does not have a proper role in this inspection. If there are no errors with the inspection object it would be built. But `find_by_name()`[33] function of the `Inspection` model cannot find already built inspection object when the next line with the same inspection name in the corresponding cell in the file for user and corresponding inspection is parsed. This event leads to the fact that the after the campaign object is created there would be a set of inspections with the same name.

In order to induce error tolerance in the `create campaign` business process[34] was changed to the current state, i.e. the campaign object should be created before the role assignments file is parsed. Therefore to implement proper handling of the create campaign business process, the following measures has been conducted. The `assignments=(f: File)` opens the role assignments file, reads the location of the temp file (corresponding to the role assignments file) and the creates a soft link with an extension corresponding to the spreadsheet file[35] and stores it in temporary variable. Then when the campaign object is created, `create_inspections()` is executed with `after_create` Active Record callback. `create_inspections()` calls `open_spreadsheet(f: File)` with the soft link as the argument to handle the spreadsheet file. After the role assignments file is parsed by `create_inspections()`, the next `after_create` callback removes soft link that was created earlier.

Controllers

There are seven controllers in the application and there is one to one correspondence between the controller and the model, except `SessionsController` that does not have an underlying model. Figure 31 presents controllers hierarchy.

[33] The function is not shown on the diagram in Figure 30 since the method was created by the Rails framework automatically.

[34] The process is described in Figure 13.

[35] A temporary file create by Rails framework does not have any extension

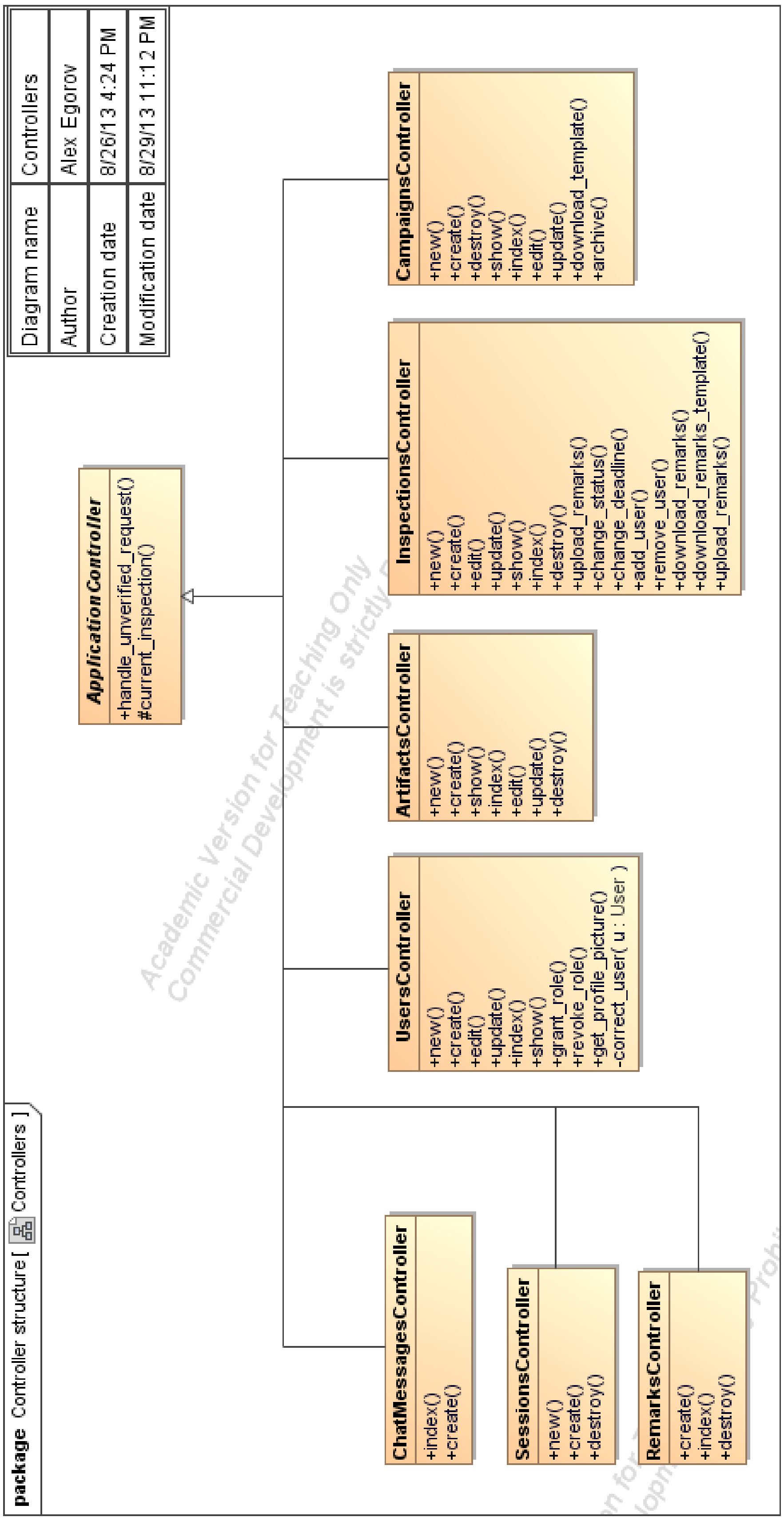


Figure 31: Controllers hierarchy

Taking into account that according to Rails convention the controller should be kept "slim", i.e it should be free from business logic, thus since controllers' actions are self-descriptive, there is no much sense in describing the action. The only function that should be explained is `correct_user(u: User)` function in `UsersController`, that is not an action. The functions is used in `update()` and `edit()` actions, it checks that the User object is ether current user or an administrative user, in order to prevent unauthorized user from modifying other user's profile.

There is no need in parameters for Ruby controllers' actions, since there is a special variable, called `params` that stores all necessary information, e.g. HTTP Get request the `params` is a query string from the request, while for HHTP Post `params` is the corresponding form data.

Views

Figure 32 demonstrates views structure and how assets are used. In the application JavaScript ERB templates are used as a response on AJAX HTTP request. All views are placed into folders according to the controller to whom they refer.

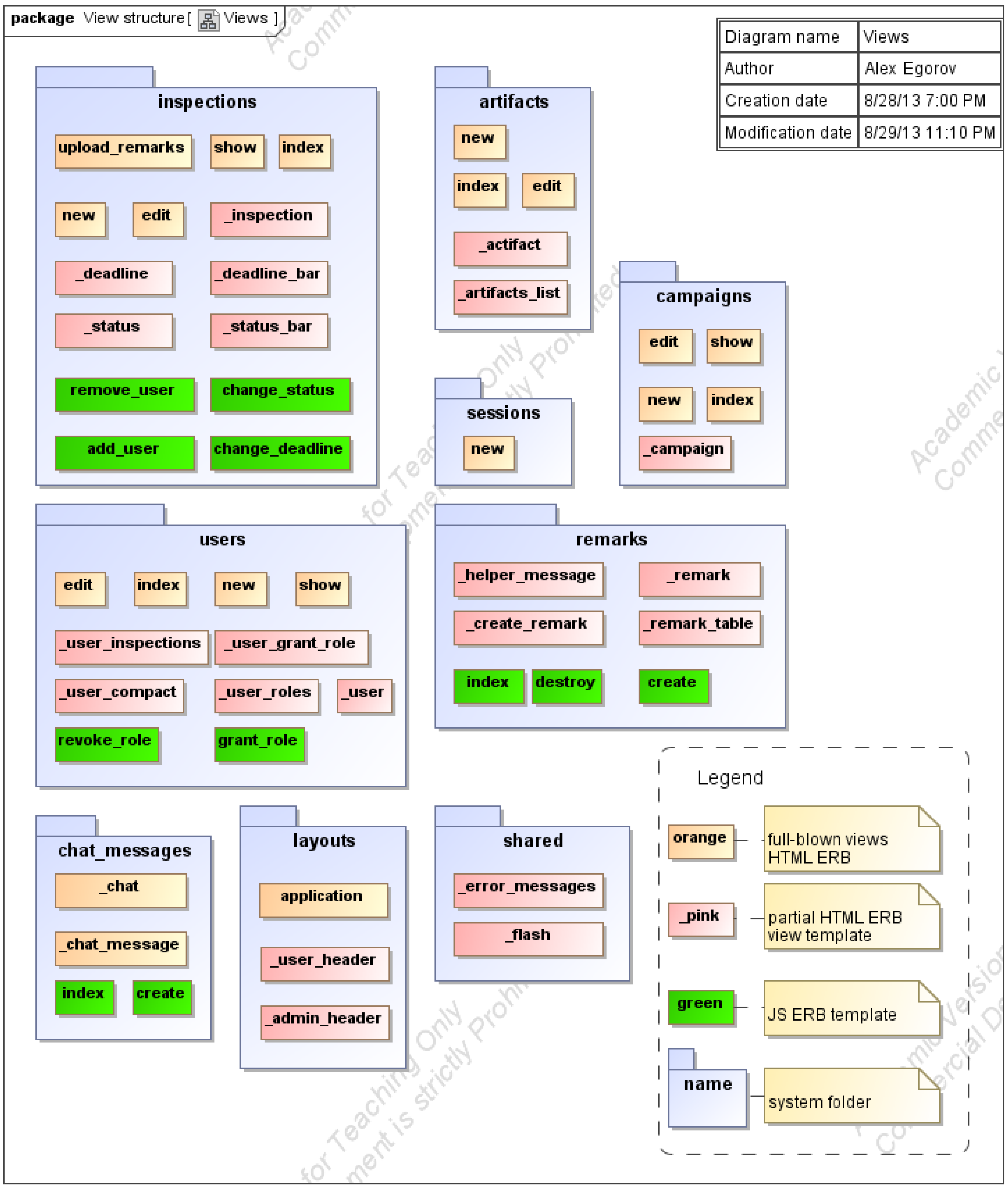


Figure 32: Views and asset usage

4.3. *Security Analysis*

Security issues were one of the many reasons why previous work on the similar project had failed. The goal is to minimize the possibility of vulnerabilities exploit by evildoer. SQL injection, Cross-Site Scripting, Mass Assignment Misuse, and Cross-Site Request Forgery attacks should be address by the FIT application. The security vulnerabilities and attack descriptions are not going to be elaborated as long as they are well-known (MITRE Corporation, 2011). Passwords are hashed by `brypt`. Session hijacking is prevented by forcing user to use HTTP SSL connection.

SQL injections

Whereas the application uses SQL database it is important to filter possible `SQL injections` or use methods that do not allow `SQL injection` to happen. Active Record provides parameterization of queries for many methods and therefore it is fair to state that methods are secure, except extreme rare cases when vulnerability found in the framework. As it was mentioned before, most of the methods in Active Record escape input from SQL[36], but there are plenty of methods that does not do this in order to allow the programmer to have flexibility. There is a list of methods and their parameters that are subject to `SQL injection`[37] and thus the usage of those methods should be conducted with care. It is worthwhile to mention only methods that are used in the application. Methods with `SQL injection` vulnerability used in the application together with measures taken to prevent `SQL injection` are shown in Table 15.

Method	Vulnerability	Prevention Measures in FIT
find()	Any argument following after the first one will not be escaped	Method is used the only one parameter, which is primary key
order()	Accepts any SQL string	User input is not passed to the method
where()	SQL query can be passed as a parameter value.	The value for primary key parameter is manually casted to integer

Table 15: Methods with SQL injection vulnerability and prevention measures

Mass Assignment Misuse Vulnerability

Mass assignment misuse vulnerability is specific to Ruby on Rails applications. Active Record methods like `update_attributes(params)` or `build(params)` are quite often used in controller actions that respond to HTML Post request, e.g. `update` or `create`[38]. The method takes a hash of attributes and their values that are going to be assigned to the record. In order to prevent the misuse of the method, e.g. an evildoer sending additional parameter that is not stated in the form through the HTML Post action, a whitelist of attributes allowed to be mass assigned should be declared. There is a special function `attr_accessible` in ActiveRecord `Base` class that is superclass for all models used in the FIT application. The method specifies which attributes may be assigned with mass assignment; any other attributes not on the list are blocked (Ediger, 2008).

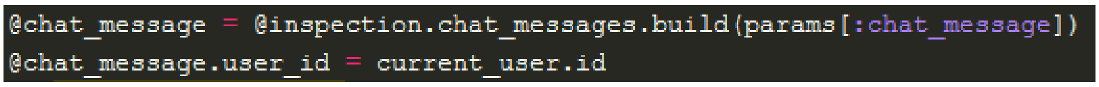
In FIT application all critical parameters are not stated in the whitelist, e.g. the piece of code from create action in `ChatMessages` controller that is shown in Figure 33 `user_id` is not passed to the method that uses mass assignment.

---

[36] http://blog.phusion.nl/2013/01/03/rails-sql-injection-vulnerability-hold-your-horses-here-are-the-facts/

[37] http://rails-sqli.org/

[38] http://happybearsoftware.com/how-i-avoid-the-rails-mass-assignment-security-mistake.html



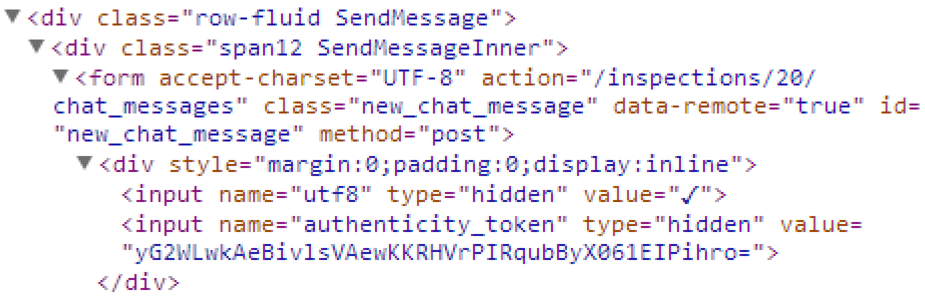
```
@chat_message = @inspection.chat_messages.build(params[:chat_message])
@chat_message.user_id = current_user.id
```

Figure 33: Chat message protection from mass assignment misuse

Instead of it `current_user` method is used that is helper method, which obtains the user who performed the action, thus it is not possible for one user to write a message on other user's behalf.

Cross-Site Request Forgery

Cross-Site Request Forgery exploits the trust that the application has for a particular user. In case of this attack both the web application and the user are victims[39]. But Ruby on Rails has a feature called `protect_from_forgery` that adds a hidden field to any form in the application. The value in the form will be compared to an ID stored in the session variable; if they differ the action will not be executed[40]. Figure 34 presents the code of send chat message form with authenticity token added by `protect_from_forgery` feature.



```
▼<div class="row-fluid SendMessage">
  ▼<div class="span12 SendMessageInner">
    ▼<form accept-charset="UTF-8" action="/inspections/20/
    chat_messages" class="new_chat_message" data-remote="true" id=
    "new_chat_message" method="post">
      ▼<div style="margin:0;padding:0;display:inline">
          <input name="utf8" type="hidden" value="✓">
          <input name="authenticity_token" type="hidden" value=
          "yG2WLwkAeBivlsVAewKKRHVrPIRqubByX061EIPihro=">
        </div>
```

Figure 34: HTML code with authenticity token of send chat message form

Cross-Site Scripting

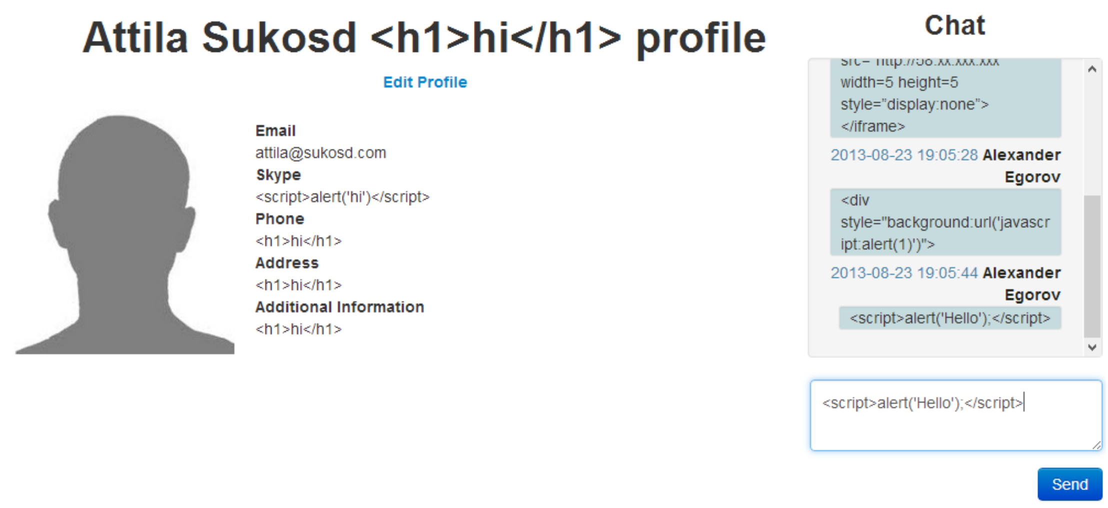
According to Ruby on Rails security guides, `Cross-Site Scripting` is the most widespread, and one of the most devastating security vulnerabilities in web applications[41]. The attack injects malicious client-side executable code such as JavaScript.

FIT application sanitize user input, therefore `XSS` menace is eliminated. Figure 35 demonstrates the result of failed `XSS` attack attempts, JS and HTML code is properly filtered, therefore the alert message is not displayed on either AJAX create request or page reload.

[39] http://shiflett.org/articles/cross-site-request-forgeries

[40] http://ruby.about.com/od/security/a/forgeryprotect.htm

[41] http://guides.rubyonrails.org/security.html



Attila Sukosd <h1>hi</h1> profile

Email
attila@sukosd.com
Skype
<script>alert('hi')</script>
Phone
<h1>hi</h1>
Address
<h1>hi</h1>
Additional Information
<h1>hi</h1>

Chat

width=5 height=5
style="display:none">
</iframe>

2013-08-23 19:05:28 **Alexander Egorov**

<div
style="background:url('javascr
ipt:alert(1)')">

2013-08-23 19:05:44 **Alexander Egorov**

<script>alert('Hello');</script>

<script>alert('Hello');</script>

Figure 35: XSS vulnerability test, the left part is conducted by Attila Sukosd, where the right one is done by Alexander Egorov. Alert message is not displayed, since the attack was not successful.

4.4. *Deployment*

There are several possibilities of how to deploy a web application. In general there are two possibilities either deploying the web application on a privately owned web server or using third party services, e.g. cloud services. Taking into account that during the focus of the thesis implementation part is on the development of the web application and there was no infrastructure provided, the first option is not applicable, since it take a great deal of resources to setup and configure a web server. Hence, in order to minimize the efforts spent on deployment, cloud services should be utilized. Considering fast deployment of the application as a goal, there are two primary options for of what cloud areas are applicable for the case: Infrastructure as a Service (`IaaS`) and Platform as a Service (`PaaS`). `IaaS` provides physical of virtual machines and other resources such as firewalls, file-based storage, load balancers, and software bundles (Amies, Sluiman, Qiang, & Guo, 2012). `PaaS` offers a platform that includes operating system and middleware. In other words `PaaS` takes care of everything needed to run a specific language or technology stack (McGrath, 2012).

The decision to deploy FIT application on `PaaS` was made, since using `PaaS` allows not caring about configuration and maintenance of the infrastructure and underlying platform. Considering the absence of budget for the application, possibility of deploying the application for free is the main requirement for `PaaS` provider.

Heroku

Heroku is a platform as a service provider that supports Ruby programming language and Ruby on Rails Rack-compatible projects. Heroku was choosen as `PaaS` provider, since Heroku offers charge-free services for developing purposes and provides the ability to deploy an application using a one-line command[42].

Heroku terminology is presented in Table 16.

[42] http://www.building43.com/videos/2010/04/20/herokus-ruby-cloud-platform/

Term Name	Term Description
Dyno	An isolated, virtualized Unix container that provides the environment required to run an application[43].
Slug	A compressed and pre-packaged copy of an application that is optimized for distribution to the dyno-manager[44].
Stack	A complete deployment environment that includes the base operating system, the language runtime and associated libraries[45].

Table 16: Heroku terminology overview

Rails application supported by Heroku stack based on Ubuntu 10.04 that is called Cedar. A dyno get 512MB of RAM and 1x CPU share in its default configuration ("1X") If the application requires more memory or CPU share it is possible to resize dyno to a "2X" configuration that gives 1024MB of RAM an doubles the CPU share on a per process-type basis[46].

FIT Deployment on Heroku

A free Heroku account is limited to one "1X" dyno available, the PostgreSQL database size is limited to 10000 rows, and there are only 20 simultaneous HTTP session are allowed.

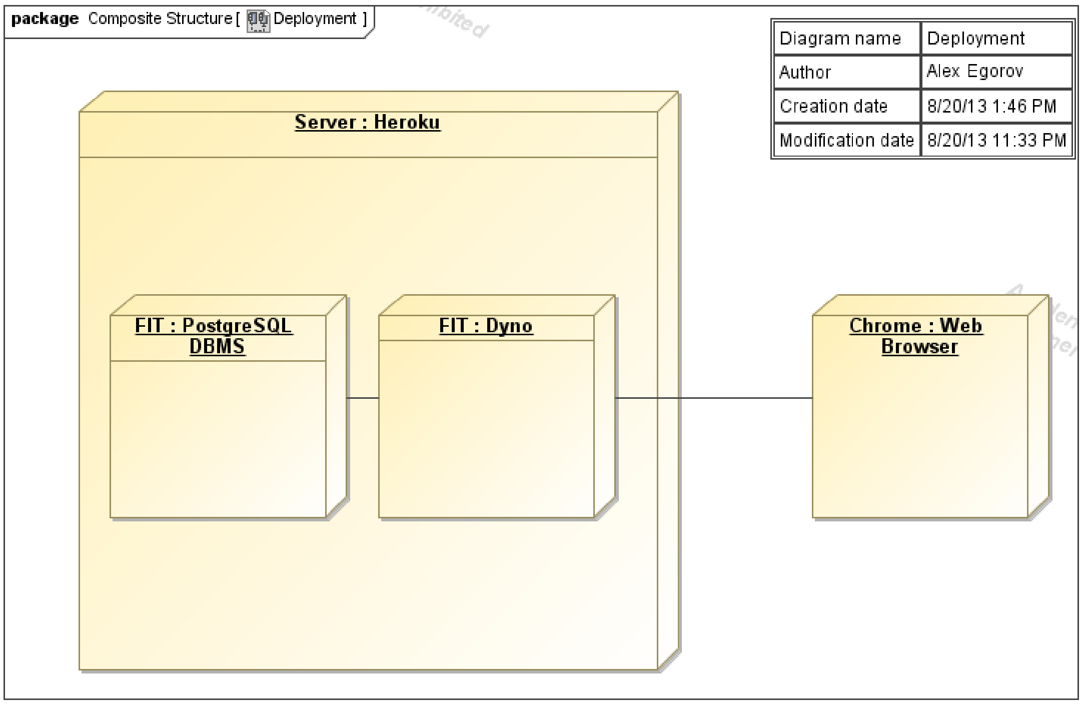


Diagram name	Deployment
Author	Alex Egorov
Creation date	8/20/13 1:46 PM
Modification date	8/20/13 11:33 PM

Figure 36: FIT deployment on Heroku

[43] https://devcenter.heroku.com/articles/how-heroku-works

[44] https://devcenter.heroku.com/articles/slug-compiler

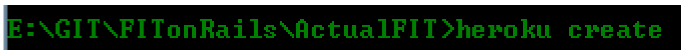
[45] https://devcenter.heroku.com/articles/stack

[46] https://devcenter.heroku.com/articles/dyno-size

The expected uptime is 99.5%, but it is worthwhile to mention that is the application deployed form free account has not been used for a while Heroku suspends the stack and it takes approximately three to five minutes to start the application. To prevent the application from sleeping there should be more than one dyno associated to the application[47].

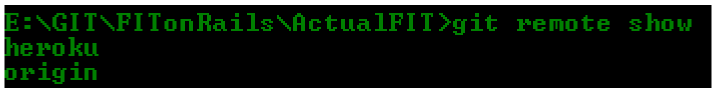
The deployment diagram is presented in Figure 36.

Taking into account that during FIT development Git VCS was used and Heroku Toolbelt is installed, the application deployment to Heroku can be done relatively easy in a few steps using command line interface. After creating the application on Heroku by executing create command from Git repository as it is shown in Figure 37 that will create remote repository called `heroku`. List of remote repositories shown in Figure 38, where `origin` is a default remote repository on GitHub for FIT application.



```
E:\GIT\FITonRails\ActualFIT>heroku create
```

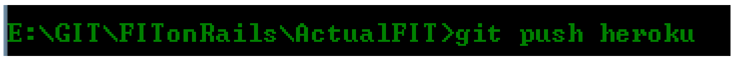
Figure 37: The application creation on Heroku



```
E:\GIT\FITonRails\ActualFIT>git remote show
heroku
origin
```

Figure 38: Git remote repository list of FIT application

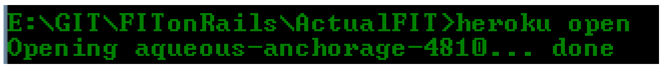
Then the desired Git branch should be pushed to Heroku with one-line command from command line interface that is shown in Figure 39.



```
E:\GIT\FITonRails\ActualFIT>git push heroku
```

Figure 39: The application deployment on Heroku

The command shown above pushes Git master branch to the Heroku that will automatically compile the application to a slug. After the compilation is finished with success, the application is deployed on Heroku and can be checked but the command shown in Figure 40 that will open a new tab in default web browser with FIT application.



```
E:\GIT\FITonRails\ActualFIT>heroku open
Opening aqueous-anchorage-4810... done
```

Figure 40: Heroku open command to start FIT application in web browser

[47] https://devcenter.heroku.com/articles/dynos#dyno-idling

4.5. *Libraries and Components*

All gems used in the application are free of charge and licensed under different open-source licenses. Icons used in the application are free for any usage.

Gem name	Description
rails	Ruby on Rails application framework.
bootstrap-sass	Includes and enables Bootstrap framework.
bcrypt-ruby	The gem includes `bcrypt` function. Used for hashing passwords.
cancan	The gem provides authorization capabilities.
rolify	Allows attaching roles to any resource, i.e. any class or concrete class instance.
thin	A web-server used for development.
roo	Spreadsheet managing library.
bootstrap-datepicker-rails	A component for picking dates with bootstrap styling.
kaminari	The gem provides pagination functions.
jquery-datatables-rails	HTML table wrappers.
sqlite3	Development database.
postgreSQL	Production database.
annotate	Annotates models with database fields.
rspec-rails	Includes RSPEC language for writing unit tests and testing environment.
factory_girl_rails	A library for creating, building instances from objects.
faker	Generates fake date, like email addresses or names.
capybara	Simulates user's activity, used in testing.
launchy	Allows launching external application from within Ruby programs.
database_cleaner	Provides database cleaning features.
guard-rspec	Automatically detects changes in code and launches test suites.
sass-rails	Includes SASS support.
coffee-rails	Allows using CoffeeScript in Rails.

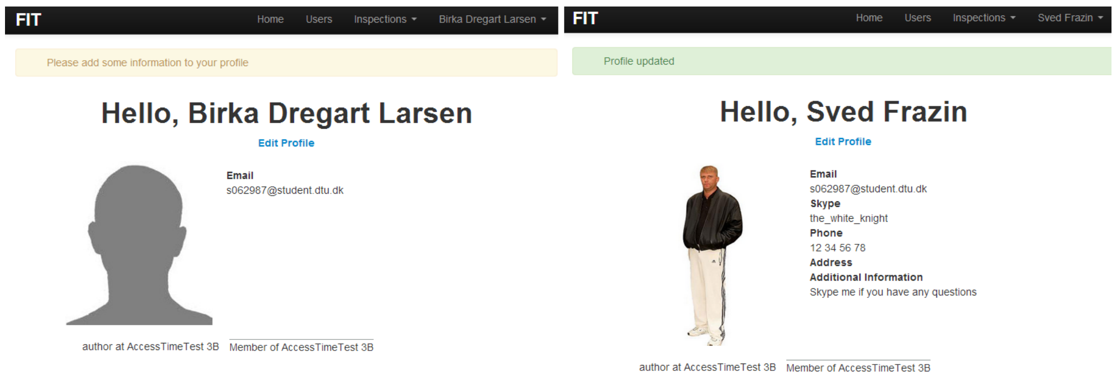
Figure 41: List of Rails gems

5 System Operation and Test

5.1. *Operation from User Perspective*

Taking into the account that many users would have the first encounter with the system shortly after introduction of Fagan-Style Inspections concept it is vital to provide appropriate guidance for the inspection participants.

After a user logs-in to the system his profile details are presented to him. If the profile is incomplete, then it is recommended to add some information to the profile. Figure 42 demonstrates changes (after the profile was updated, the notification of incomplete profile disappeared).



Please add some information to your profile

Hello, Birka Dregart Larsen

Email
s062987@student.dtu.dk

author at AccessTimeTest 3B
Member of AccessTimeTest 3B

Profile updated

Hello, Sved Frazin

Email
s062987@student.dtu.dk
Skype
the_white_knight
Phone
12 34 56 78
Address
Additional Information
Skype me if you have any questions

author at AccessTimeTest 3B
Member of AccessTimeTest 3B

Figure 42: User profile before change (left) and after (right)

After clicking on the inspections list at the navigation panel at the top as it is show on Figure 43, the list of inspections where the user is a member is demonstrated. When the user clicks on the inspection name he is redirected to the inspection page, which is shown in Figure 44.

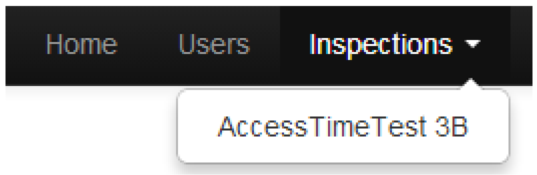
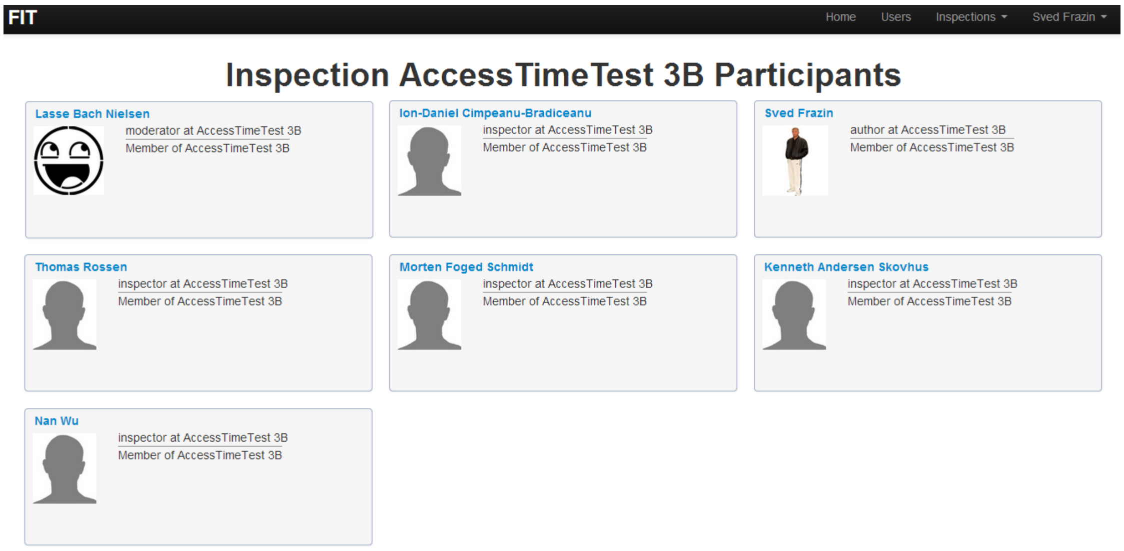


Figure 43: Inspections list

Alternatively, if the user decides to click on Users menu item then Inspection Participants are presented. The page with participants list for the inspection is shown in Figure 44. The page contains list of users with small amount of information such as roles in inspections, profile pictures. The links to user profiles with full information are available at this page.



Inspection AccessTimeTest 3B Participants

Lasse Bach Nielsen
moderator at AccessTimeTest 3B
Member of AccessTimeTest 3B

Ion-Daniel Cimpeanu-Bradiceanu
inspector at AccessTimeTest 3B
Member of AccessTimeTest 3B

Sved Frazin
author at AccessTimeTest 3B
Member of AccessTimeTest 3B

Thomas Rossen
inspector at AccessTimeTest 3B
Member of AccessTimeTest 3B

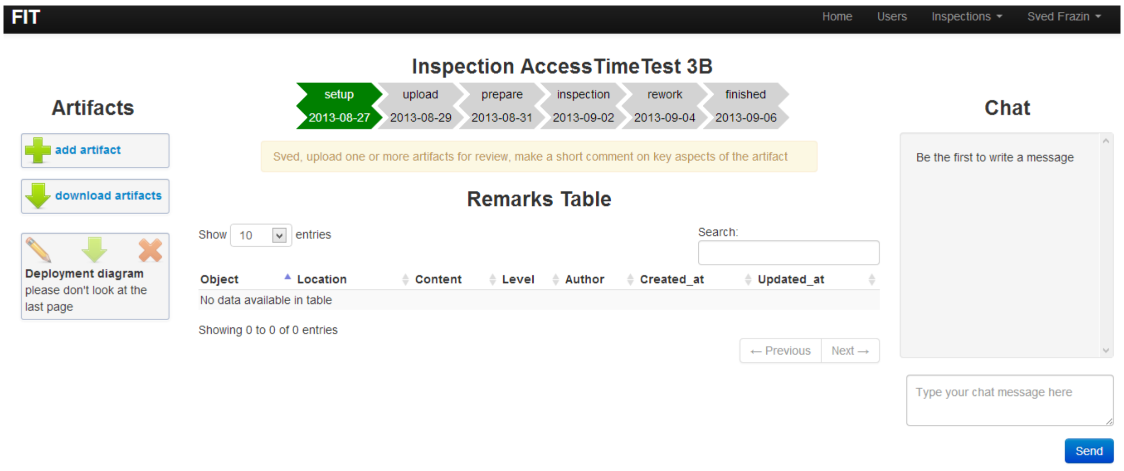
Morten Foged Schmidt
inspector at AccessTimeTest 3B
Member of AccessTimeTest 3B

Kenneth Andersen Skovhus
inspector at AccessTimeTest 3B
Member of AccessTimeTest 3B

Nan Wu
inspector at AccessTimeTest 3B
Member of AccessTimeTest 3B

Figure 44: Inspection Participants page

The figure below shows how FIT guides authors through the various stages of the inspection process. The guidance contains information what author should do in particular stage of the inspection process. During the setup and upload stages author should upload one or more artifacts, so the inspectors can review them.



Inspection AccessTimeTest 3B

setup	upload	prepare	inspection	rework	finished
2013-08-27	2013-08-29	2013-08-31	2013-09-02	2013-09-04	2013-09-06

Sved, upload one or more artifacts for review, make a short comment on key aspects of the artifact

Deployment diagram
please don't look at the last page

Remarks Table

Show 10 entries Search:

Object	Location	Content	Level	Author	Created_at	Updated_at
No data available in table						

Showing 0 to 0 of 0 entries

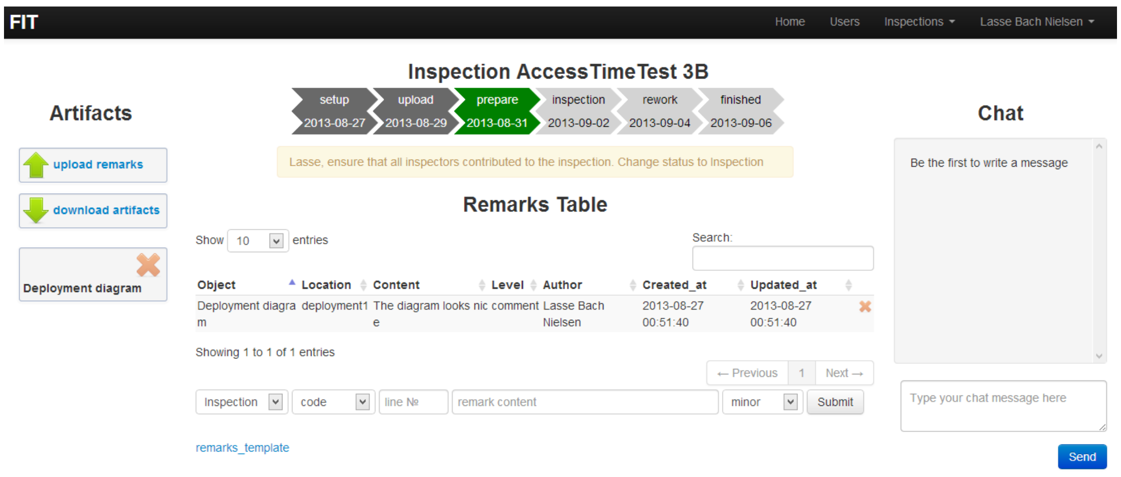
← Previous Next →

Be the first to write a message

Type your chat message here

Figure 45: Author perspective on the inspection process

The inspection page from the moderator perspective is presented in Figure 46. As it is possible to notice the status of the inspection was changed, thus the status bar displayed that change by coloring past stages with dark gray color. The moderator was able to change the status because one artifact was uploaded. During the prepare stage the moderator received a personal guidance, which asks him to ensure that the inspectors contributed to the inspection during this stage.


FIT Home Users Inspections Lasse Bach Nielsen

Inspection AccessTimeTest 3B

setup 2013-08-27 | upload 2013-08-29 | prepare 2013-08-31 | inspection 2013-09-02 | rework 2013-09-04 | finished 2013-09-06

Artifacts

upload remarks

download artifacts

Deployment diagram

Lasse, ensure that all inspectors contributed to the inspection. Change status to inspection

Remarks Table

Show 10 entries

Search:

Object	Location	Content	Level	Author	Created_at	Updated_at
Deployment diagra m	deployment1	The diagram looks nic e	comment	Lasse Bach Nielsen	2013-08-27 00:51:40	2013-08-27 00:51:40

Showing 1 to 1 of 1 entries

← Previous 1 Next →

Inspection | code | line № | remark content | minor | Submit

remarks_template

Chat

Be the first to write a message

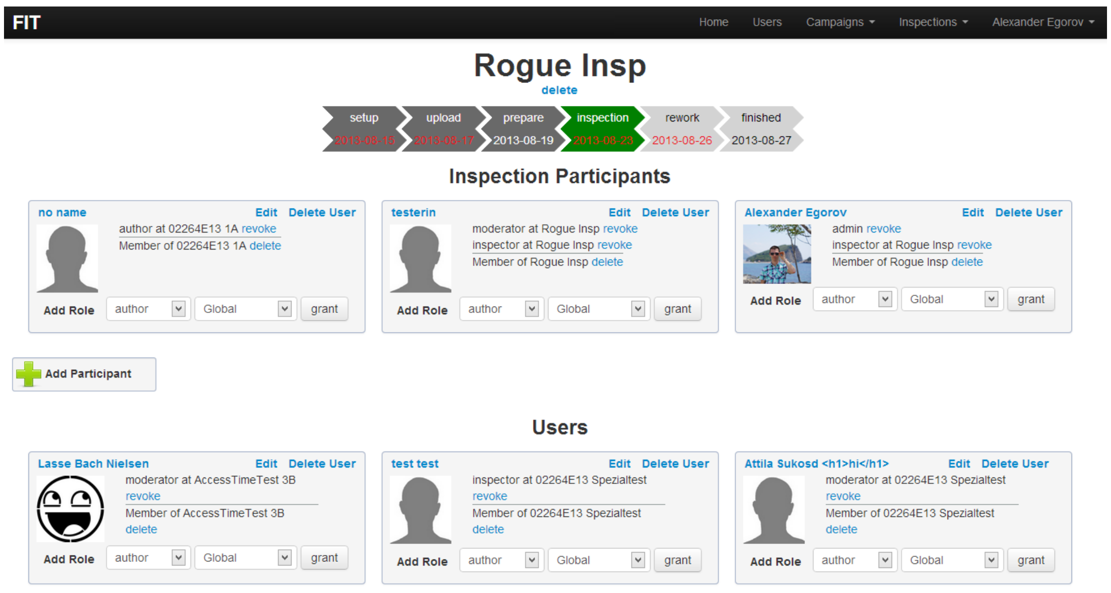
Type your chat message here

Send

Figure 46: Moderator perspective on the inspection page

5.2. *Operation from Administrator Perspective*

Looking from the administrator point of view it is possible to notice some additional features of the system. The administrator is able to perform any action that any other user can and also he is able to create, modify and delete inspections, campaigns and users. The inspection edit page is presented in Figure 47, where it is possible to notice that additional actions became available in comparison to users participants list.


FIT Home Users Campaigns Inspections Alexander Egorov

Rogue Insp

delete

setup 2013-08-15 | upload 2013-08-17 | prepare 2013-08-19 | inspection 2013-08-23 | rework 2013-08-26 | finished 2013-08-27

Inspection Participants

no name Edit Delete User
author at 02264E13 1A revoke
Member of 02264E13 1A delete
Add Role author Global grant

testerin Edit Delete User
moderator at Rogue Insp revoke
inspector at Rogue Insp revoke
Member of Rogue Insp delete
Add Role author Global grant

Alexander Egorov Edit Delete User
admin revoke
inspector at Rogue Insp revoke
Member of Rogue Insp delete
Add Role author Global grant

Add Participant

Users

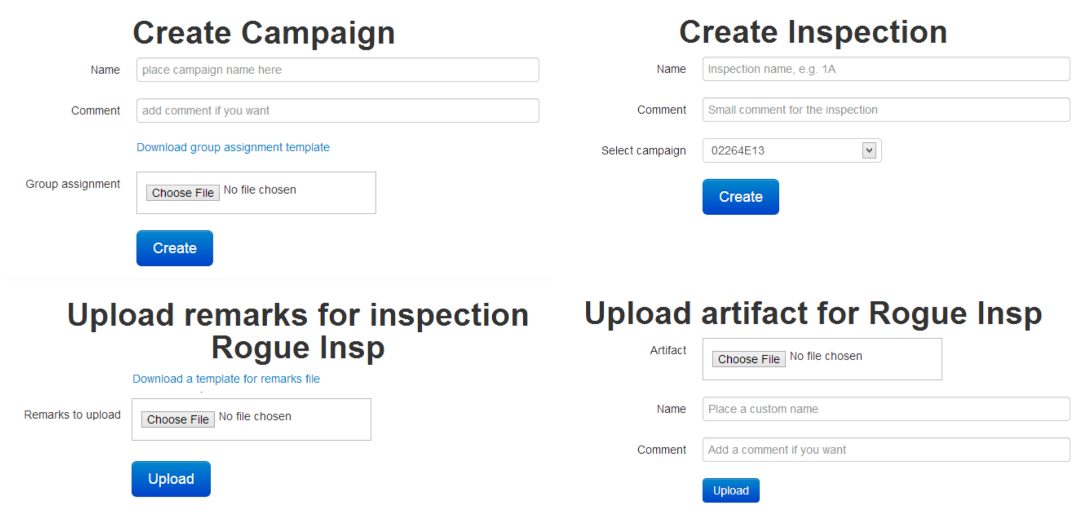
Lasse Bach Nielsen Edit Delete User
moderator at AccessTimeTest 3B revoke
Member of AccessTimeTest 3B delete
Add Role author Global grant

test test Edit Delete User
inspector at 02264E13 Spezialtest revoke
Member of 02264E13 Spezialtest delete
Add Role author Global grant

Attila Sukosd <h1>hi</h1> Edit Delete User
moderator at 02264E13 Spezialtest revoke
Member of 02264E13 Spezialtest delete
Add Role author Global grant

Figure 47: Inspection edit

Figure 48 shows the combination of create and upload forms.


Create Campaign

Name: place campaign name here

Comment: add comment if you want

Download group assignment template

Group assignment: Choose File No file chosen

Create

Create Inspection

Name: Inspection name, e.g. 1A

Comment: Small comment for the inspection

Select campaign: 02264E13

Create

Upload remarks for inspection Rogue Insp

Download a template for remarks file

Remarks to upload: Choose File No file chosen

Upload

Upload artifact for Rogue Insp

Artifact: Choose File No file chosen

Name: Place a custom name

Comment: Add a comment if you want

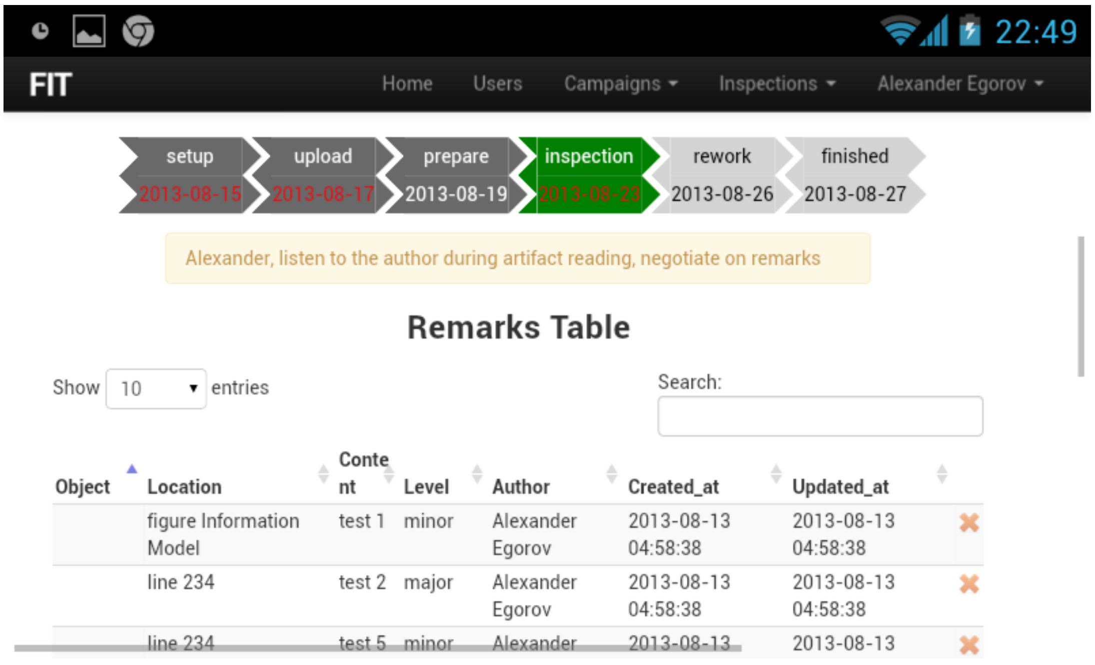
Upload

Figure 48: Create and upload pages combined

5.3. *Testing*

Browser Compatibility

The application was tested on several different browsers such as Chrome 28, Mozilla Firefox 22, and Internet Explorer 10 with different screen resolutions starting from 1378x768 to 1920x1080. The application showed full compatibility with all tested browsers.


22:49

FIT Home Users Campaigns Inspections Alexander Egorov

setup 2013-08-15 | upload 2013-08-17 | prepare 2013-08-19 | inspection 2013-08-23 | rework 2013-08-26 | finished 2013-08-27

Alexander, listen to the author during artifact reading, negotiate on remarks

Remarks Table

Show 10 entries Search:

Object	Location	Content	Level	Author	Created_at	Updated_at	
	figure Information Model	test 1	minor	Alexander Egorov	2013-08-13 04:58:38	2013-08-13 04:58:38	✖
	line 234	test 2	major	Alexander Egorov	2013-08-13 04:58:38	2013-08-13 04:58:38	✖
	line 234	test 5	minor	Alexander	2013-08-13	2013-08-13	✖

Figure 49: Inspection view on mobile device

The application was also tested on different devices and operating systems. It was tested under Windows 8, Mac OS X and Android 4.1. FIT showed acceptable results for mobile devices, since the support of the mobile devices was not a goal for the system development. The scaling problems may occur. Inspection view is presented in Figure 49 as it is seen on Google Nexus S mobile phone perspective using Google Chrome browser.

The application does not scale properly on some places, but it works with an appropriate speed, and full functionality is preserved with a small limitations.

Inspection Errors

Figure 50 shows the campaign with four inspections, three inspections are properly staffed with author, moderator and at least one inspector, while one inspection, called 2B, is staffed incorrectly, therefore it is highlighted with red box and the warning message appeared, notifying the administrator that there is a problem.

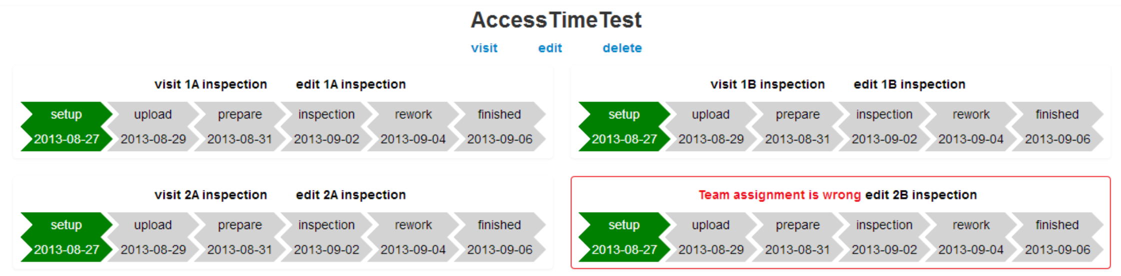

AccessTimeTest
visit edit delete
visit 1A inspection edit 1A inspection
setup upload prepare inspection rework finished
2013-08-27 2013-08-29 2013-08-31 2013-09-02 2013-09-04 2013-09-06
visit 1B inspection edit 1B inspection
setup upload prepare inspection rework finished
2013-08-27 2013-08-29 2013-08-31 2013-09-02 2013-09-04 2013-09-06
visit 2A inspection edit 2A inspection
setup upload prepare inspection rework finished
2013-08-27 2013-08-29 2013-08-31 2013-09-02 2013-09-04 2013-09-06
Team assignment is wrong edit 2B inspection
setup upload prepare inspection rework finished
2013-08-27 2013-08-29 2013-08-31 2013-09-02 2013-09-04 2013-09-06

Figure 50: Inspection with staffing problems

If a deadline for an inspection is missed then the date font turns into red color. Figure 51 displays inspections a campaign with three inspections that have problems with deadlines. A deadline can be missed in two cases: the first one if the deadline was not clothed and current date is in highlighted (`Upload` and `Prepare` deadlines for `foobar` inspection), the second one when the deadline was closed after the due date (`Setup` and `Upload` deadlines on `1A` inspection).

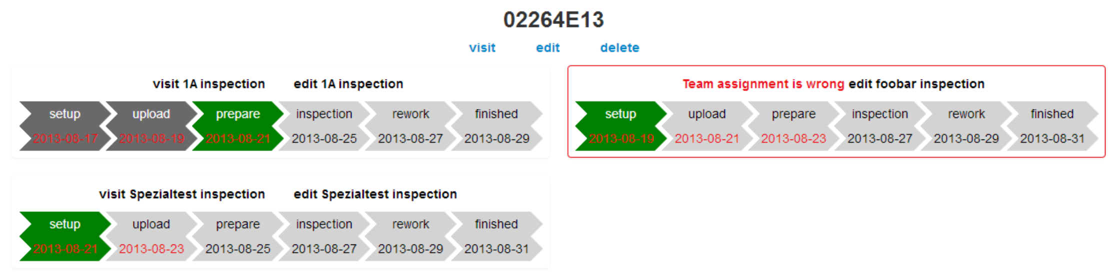

02264E13
visit edit delete
visit 1A inspection edit 1A inspection
setup upload prepare inspection rework finished
2013-08-17 2013-08-19 2013-08-21 2013-08-25 2013-08-27 2013-08-29
Team assignment is wrong edit foobar inspection
setup upload prepare inspection rework finished
2013-08-19 2013-08-21 2013-08-23 2013-08-27 2013-08-29 2013-08-31
visit Spezialtest inspection edit Spezialtest inspection
setup upload prepare inspection rework finished
2013-08-21 2013-08-23 2013-08-25 2013-08-27 2013-08-29 2013-08-31

Figure 51: Missed deadlines

Deadline Errors

During the inspection process the moderator or the administrator can change a deadline. Figure 52 displays the successful deadline change. The user after clicking on the deadline (marked with one on the figure) will see the calendar appeared, a date highlighted with blue show current date. When user clicks on the desired date (marked with two on the figure) the calendar closes and the save icon appears.

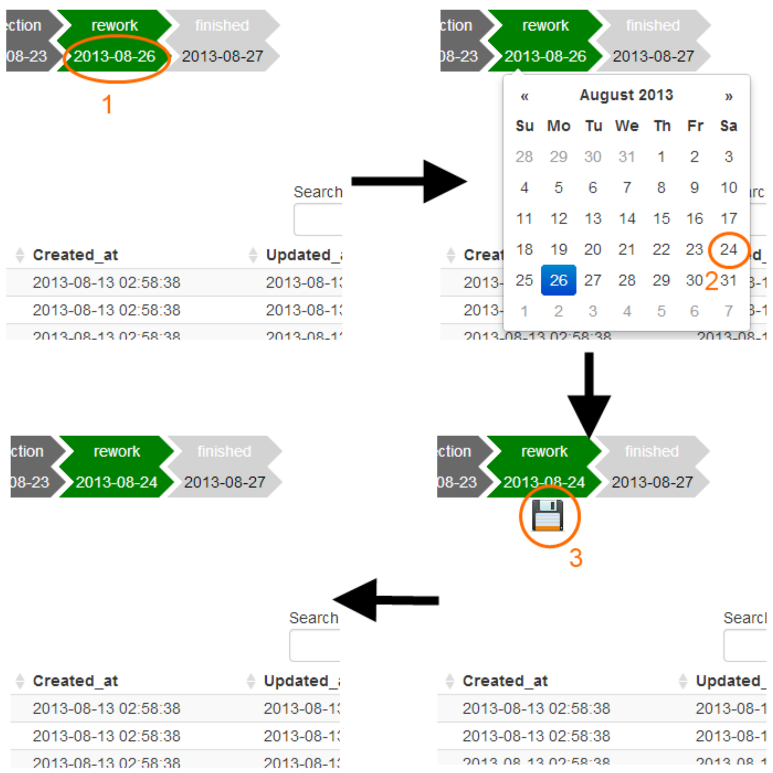
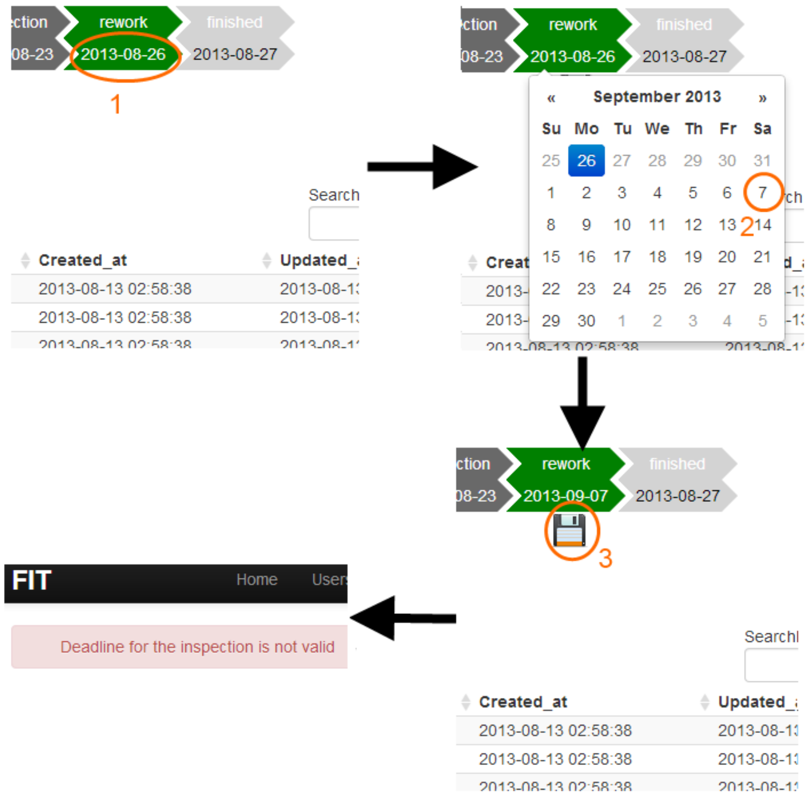


Figure 52: Change Deadline

After user clicks on the icon it will disappear and no error message will be shown. It means that the change was successful.



Deadline for the inspection is not valid

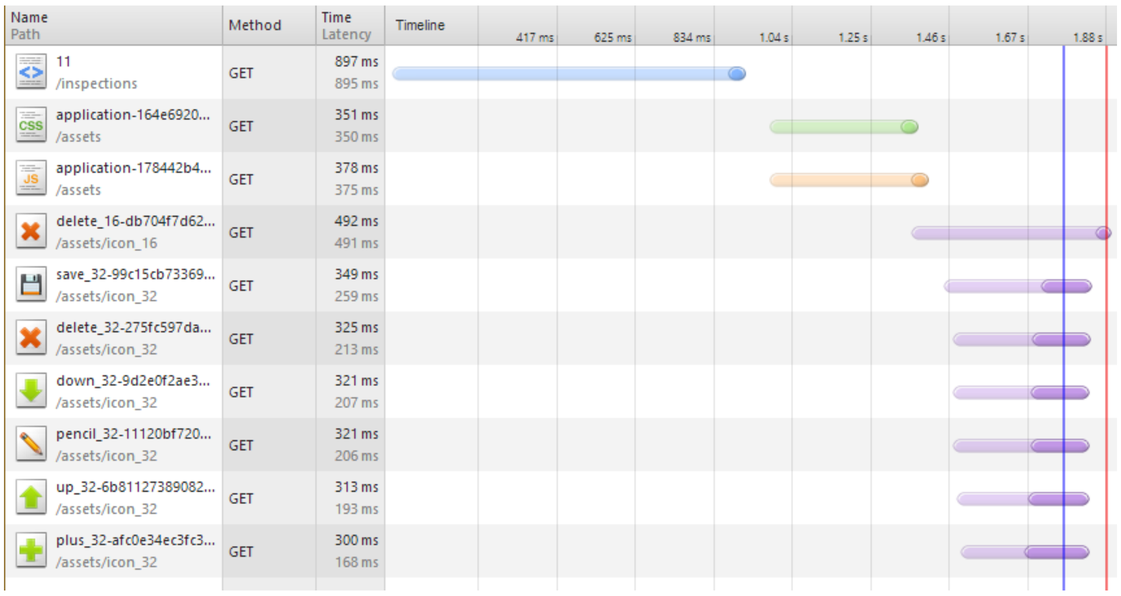
Figure 53: Select Wrong Deadline

If the user select incorrect deadline as it is presented in Figure 53 after pressing the save button an error message would be shown notifying user that there a mistake was done.

5.4. *Loading Time*

The loading time was measured for the application deployed on Heroku with Chrome build-in developer tool. The average ping time for the Heroku website was 140ms.

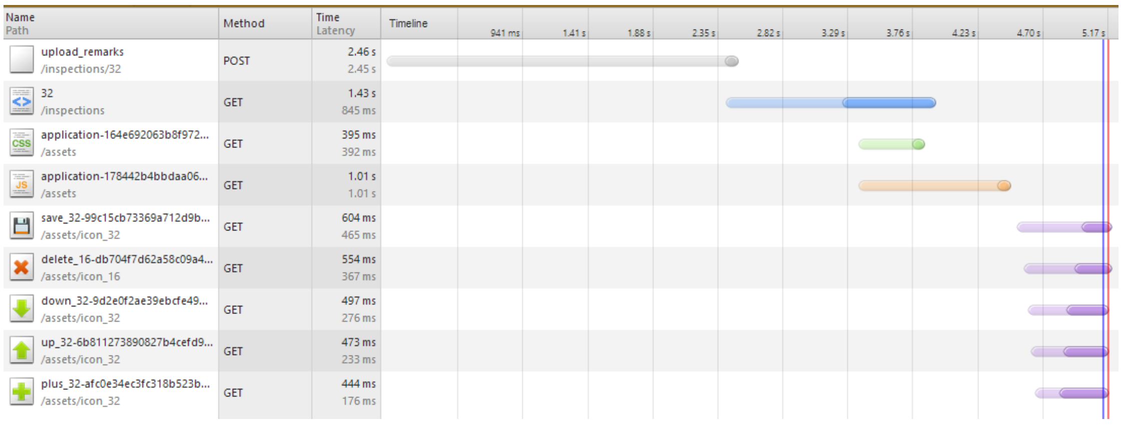
The loading time for an inspection page was around 1.9s, the inspection contained several artifacts, remarks and chat messages. The loading time graph is presented in Figure 54. The HTML document was loaded in 0.9 seconds and during the other 0.9 second CSS, JS and image assets were loaded.



Name Path	Method	Time Latency
11 /inspections	GET	897 ms 895 ms
application-164e6920... /assets	GET	351 ms 350 ms
application-178442b4... /assets	GET	378 ms 375 ms
delete_16-db704f7d62... /assets/icon_16	GET	492 ms 491 ms
save_32-99c15cb73369... /assets/icon_32	GET	349 ms 259 ms
delete_32-275fc597da... /assets/icon_32	GET	325 ms 213 ms
down_32-9d2e0f2ae3... /assets/icon_32	GET	321 ms 207 ms
pencil_32-11120bf720... /assets/icon_32	GET	321 ms 206 ms
up_32-6b81127389082... /assets/icon_32	GET	313 ms 193 ms
plus_32-afc0e34ec3fc3... /assets/icon_32	GET	300 ms 168 ms

Figure 54: Loading time for inspection page

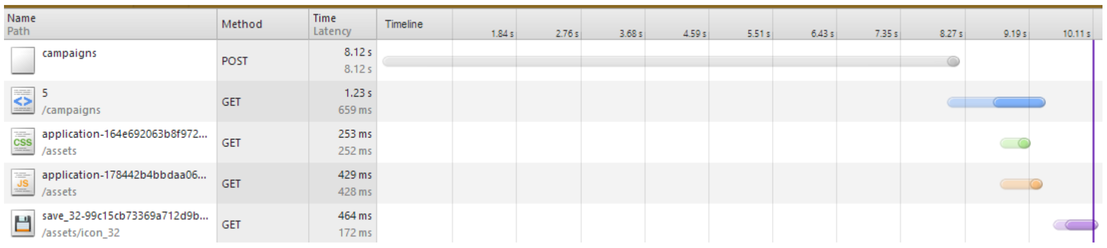
Remark upload is an important action during the inspection process, thus it is important to ensure that the application quickly process the spreadsheet. The response time for remark upload and subsequent inspection rendering reached 5.2 seconds for the test spreadsheet with 100 remarks. The graph with latencies is shown in Figure 55.



Name Path	Method	Time Latency
upload_remarks /inspections/32	POST	2.46 s 2.45 s
32 /inspections	GET	1.43 s 845 ms
application-164e692063b8f972... /assets	GET	395 ms 392 ms
application-178442b4bbdaa06... /assets	GET	1.01 s 1.01 s
save_32-99c15cb73369a712d9b... /assets/icon_32	GET	604 ms 465 ms
delete_16-db704f7d62a58c09a4... /assets/icon_16	GET	554 ms 367 ms
down_32-9d2e0f2ae39ebcfe49... /assets/icon_32	GET	497 ms 276 ms
up_32-6b811273890827b4cefd9... /assets/icon_32	GET	473 ms 233 ms
plus_32-afc0e34ec3fc318b523b... /assets/icon_32	GET	444 ms 176 ms

Figure 55: Loading time for upload remarks

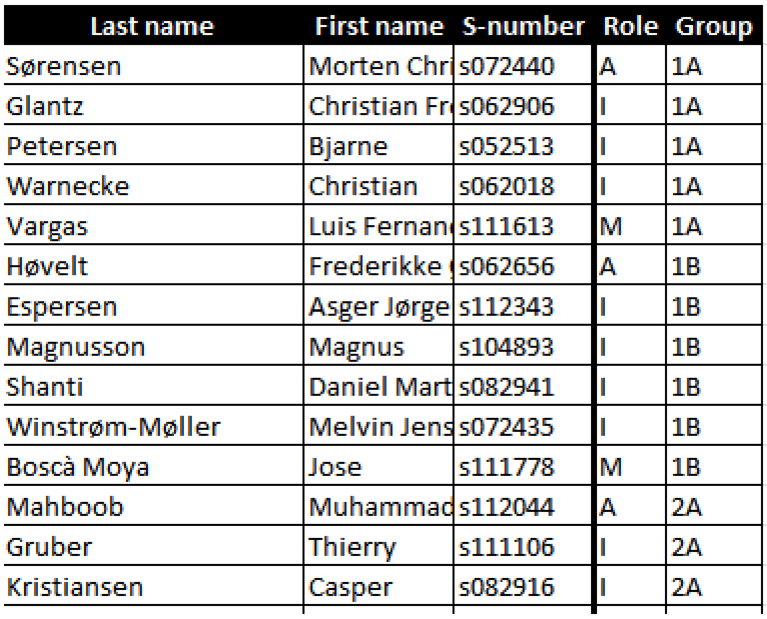
Campaign creation is also important action for an administrative user. Even that it occurs relatively rarely[48] in is essential to maintain adequate processing time for the user request. Taking into account that the administrator user would like to use role assignments file in order to make the system create inspections, assign roles and create users if necessary. The response time graph for create campaign action is presented in Figure 56.



Name / Path	Method	Time / Latency
campaigns	POST	8.12 s / 8.12 s
5 /campaigns	GET	1.23 s / 659 ms
application-164e692063b8f972... /assets	GET	253 ms / 252 ms
application-178442b4bbdaa06... /assets	GET	429 ms / 428 ms
save_32-99c15cb73369a712d9b... /assets/icon_32	GET	464 ms / 172 ms

Figure 56: Loading time for create campaign with role assignments file

In order to understand the system performance on real data, the special dataset was used. The example of the spreadsheet structure is presented in Table 17. It should be noticed that this file was used in 02264 Requirement Engineering in 2011. The spreadsheet was used in create campaign action, thus according to the spreadsheet, FIT had to create nine inspections, 59 new users and assign 59 roles.



Last name	First name	S-number	Role	Group
Sørensen	Morten Chri	s072440	A	1A
Glantz	Christian Fr	s062906	I	1A
Petersen	Bjarne	s052513	I	1A
Warnecke	Christian	s062018	I	1A
Vargas	Luis Fernan	s111613	M	1A
Høvelt	Frederikke	s062656	A	1B
Espersen	Asger Jørge	s112343	I	1B
Magnusson	Magnus	s104893	I	1B
Shanti	Daniel Mart	s082941	I	1B
Winstrøm-Møller	Melvin Jens	s072435	I	1B
Boscà Moya	Jose	s111778	M	1B
Mahboob	Muhammad	s112044	A	2A
Gruber	Thierry	s111106	I	2A
Kristiansen	Casper	s082916	I	2A

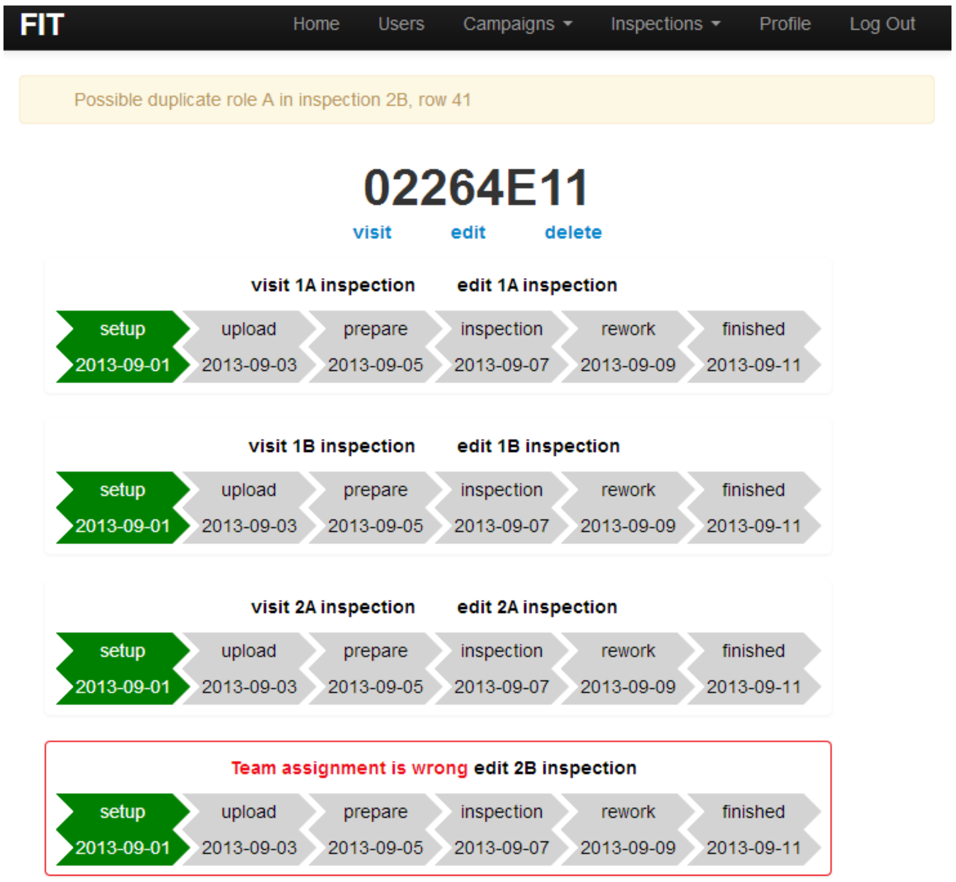
Table 17: Role assignments spreadsheet example

Since it was a real data, the spreadsheet had a mistake: inspection 2B had two authors, instead of one, therefore FIT discovered this inconsistency and one user became an Author, the other one - didn't. After the campaign was created a notification message appeared as it shown in Figure 57.

[48] According to Table 8: Create Campaign Use Case

The latency for the create campaign action appeared to be 10.2 seconds, which may seem a little bit too high, but can be considered as an appropriate result, since it occurs only a few times during a semester.

The upload and processing time for an artifact with 1.8 Mb size had reached 23 seconds.



FIT Home Users Campaigns Inspections Profile Log Out

Possible duplicate role A in inspection 2B, row 41

02264E11

visit edit delete

visit 1A inspection edit 1A inspection

setup 2013-09-01 | upload 2013-09-03 | prepare 2013-09-05 | inspection 2013-09-07 | rework 2013-09-09 | finished 2013-09-11

visit 1B inspection edit 1B inspection

setup 2013-09-01 | upload 2013-09-03 | prepare 2013-09-05 | inspection 2013-09-07 | rework 2013-09-09 | finished 2013-09-11

visit 2A inspection edit 2A inspection

setup 2013-09-01 | upload 2013-09-03 | prepare 2013-09-05 | inspection 2013-09-07 | rework 2013-09-09 | finished 2013-09-11

Team assignment is wrong edit 2B inspection

setup 2013-09-01 | upload 2013-09-03 | prepare 2013-09-05 | inspection 2013-09-07 | rework 2013-09-09 | finished 2013-09-11

Figure 57: Error notification on role duplicate

Figure 58 demonstrates FIT performance evaluation on uploading and processing remarks depending on a file size.

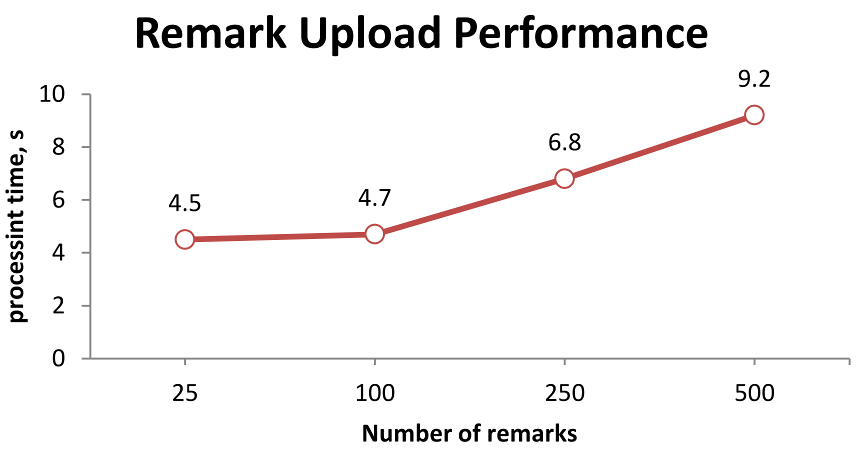


Figure 58: Remark upload performance

Considering that loading time test result may vary due to a complex combination of factors not depending from the system such as internet connection speed, browser version, and load on Heroku and that the application was tested on representative data, the obtained results on the most critical actions is can be considered as excellent, since the application is hosted in North America using free services.

6 Conclusion

The targets that were set by the thesis were successfully accomplished. Preliminary research on previous works revealed a great number of drawbacks and weaknesses. They were carefully analyzed and measures had been taken to improve in the new application.

During system analysis phase the formal inspection process was modified to comply with the introduction of the online support tool. The business processes were determined and reengineered on later stage.

The system was designed with extensive usage of prototyping. The whiteboard sketching was used during the first round of the prototyping. On the next stage of the prototyping the user interface was implemented using Bootstrap framework. The working prototype was created with Bootstrap and Ruby on Rails framework.

During the implementation process the application was created. FIT is based on Model View Controller architecture, which was implemented according to modern trends in web application development with an extensive use of most reliable and popular frameworks, such as Bootstrap and Ruby on Rails. The system is modular and flexible; therefore it provides great extensibility possibilities.

During the testing phase the tool was checked for possible drawbacks. It was concluded that FIT does not suffer from the drawbacks that led predecessor system to failure. FIT does support national character sets, it provides guidance for the inspection participants and is equipped with communication medium. The administrative load on privileged users lowered in great extent, due to FIT tolerance to errors and automation in most routine tasks, such as a campaign creation or overview on current status of multiple inspections.

The security analysis was conducted, it showed that the system is reliable and it is not a subject to the most popular attacks on web applications. The application was deployed on Heroku cloud platform that makes FIT easily maintainable, since concerns related to a web server setup and maintenance are eliminated. Formal Inspection Tool is a free open-source software product, which sources are worldwide publicly available through GitHub, therefore other developers can use the experience and results of this work.

The author waits with anticipation when the application is going to be used during autumn semester in 02264 Requirements Engineering course.

7 Bibliography

Amies, A., Sluiman, H., Qiang, G., & Guo, N. (2012). Developing and Hosting Applications on the Cloud. *IBM Press*.

Carneiro, C. J., & Al Barazi, R. (2010). *Beginning Rails 3.* Apress.

Ediger, B. (2008). *Advanced Rails.* Sebastopol: O'Relly.

Fagan, M. E. (1976). Design and Code inspection to reduce errors in program development. *IBM Systems Journal*, 182-211.

Fagan, M. E. (1986, July 7). Advances in Software Inspections. *IEEE Transactions on Software Engineering*, pp. 744-751.

Fowler, M. (2003). *Patterns of Enterprise Application Architecture.* Addison-Wesley Professional.

Loeliger, J. (2009). *Version Control with Git.* O’Reilly Media.

McDonald, M., Musson, R., & Smith, R. (2007). *The Practical Guide to Defect Prevention.* Microsoft Press.

McGrath, M. P. (2012). *Understanding Paas.* Sebastopol: O'Reilly.

MITRE Corporation. (2011, September 13). 2011 CWE/SANS Top 25 Most Dangerous Software Errors. USA.

Petrolyte, R. (2011). *FIT - an Online Inspection Support Tool.* Kgs. Lyngby: DTU.

Scott, B., & Neil, T. (2009). *Designing Web Interfaces.* O’Reilly Media.

Störrle, H. (2012). QA3 Inspection Process Guide.

Störrle, H. (2012). Requirements Engineering.

A Delivery

The table below contains information regarding delivery of the project data. Filenames are written in “quotation marks”, while directories are written without.

File or Directory	Description
“readme.txt”	An explanation of the contents of the delivery.
“MSc_Alexander_Egorov_s111888.pdf”	The thesis as such (this document), as submitted to the IMM librarian for print.
https://aqueous-anchorage-4810.herokuapp.com	It is important to remember that since the project is open-source and does not have any budget, a free account was used. The account assumes that the process manager puts the application into ‘sleep’[49] if it is not used for a while. It takes approximately from 3 to 5 minutes to ‘wake up’ the application and an error may be presented, to eliminate error the browser page should be refreshed, e.g. by pressing F5.
Thesis	
“MSc_Alexander_Egorov_s111888.docx”	Microsoft Word file user to generate the PDF version of this paper.
Picture	Pictures included as figures
“Project_s111888.zip”	The compressed development version of the project.
Project	The development version of the project.
https://github.com/AlEg0r0v/ActualFIT	FIT Github repository.

Table 18: Delivery

[49] https://devcenter.heroku.com/articles/dynos#dyno-sleeping

B Glossary

Term	Description
AJAX	Asynchronous JavaScript and XML
AR	Active Record pattern
Artifact	Any kind of document, i.e. design specification, model or part of model, code documentation, code listings, etc.
CSRF, XSRF	Cross-Site Request forgery
DBMS	Database management system
Dyno	Virtualized lightweight container on Heroku
ERB	HTML or JS template with embedded Ruby
Fagan Inspection, formal inspection	A process for detecting defect in software engineering process
FIT	formal inspection tool
Gem	Ruby pluggable library
Heroku	PaaS provider
IaaS	Infrastructure as a Service
IDE	Integrated development environment
Inspection, Inspection process	Formal inspection process
JS	JavaScript
Long-Polling	Technique to obtain updates from server
MVC	Model View Controller pattern
NoSQL	Non relation DBMS that does not use SQL
PaaS	Platform as a Service
Rack	Rack is web server interface for Ruby
Rails, RoR, Ruby on Rails	Web application framework based on Ruby
RDBMS	Relational DBMS
Remark	Comment to artifact
Slug	Precompiled application on Heroku
SSE	Server-Sent Events, technology used for server push
Stack	Heroku…
VCS	Version control system
WebSockets	Technique to obtain updates from server
XSS	Cross-Site Scripting

Table 19: Glossary

C List of Figures

D List of Tables