Collaborative Platform for Analysis of Software Systems

Catarina Isabel Carvalho Santana

Thesis to obtain the Master of Science Degree in

Information and Software Engineering

Advisor(s)/Supervisor(s): Prof. Antnio Manuel Ferreira Rito da Silva

Examination Committee

Chairperson: Prof./Dr. Lorem Ipsum
Advisor: Prof./Dr. Antnio Manuel Ferreira Rito da Silva
Members of the Committee: Prof./Dr. Lorem Ipsum

October 2015

Acknowledgments

Abstract

The Software Architectures course teaches students the most important concepts on design and architecture of software systems and makes students apply these concepts to real, complex software systems. Organizing knowledge and applying theory to practice is not an easy task, and students often need to ask questions and discuss. State-of-the-art on collaborative, social software and knowledge structuring is analyzed and a social platform is proposed to solve the problems adjacent to this course.

Resumo

O teu resumo aqui...

Keywords: as tuas palavras chave

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Analyzing and discussing big, real, open-source and highly complex software system is a very important part of the Software Architectures course. In the course context, students must apply concepts and techniques for design and analysis of software architectures to descriptions of real systems.

However, applying these concepts and techniques is not a very easy task, and often students have questions and doubts regarding these descriptions. These questions sometimes require not only consulting the course bibliography, but also discussing with peers or asking questions to teachers.

The following document is organized as follows: Section **??** provides a more detailed description of the problem presented last paragraph and presents a possible solution for the problem. Section 4 presents the state-of-the-art on the components of the solution. Section 5 describes a possible solution for the problem and Section 9 describes how the implemented solution will be assessed.

Chapter 2

Problem Description

The course of Software Architectures teaches students the most important concepts in the field of software architectures and applies these concepts to real-life software systems.

The practical component of this course (where the theory is applied) is done by analyzing documents/articles that describe the architectures of real-life systems and applying the concepts learned in theory lessons: extracting stakeholders, scenarios, tactics, views, etc. This analysis is either done by students with the help of the teacher, during practical classes, or in work groups of usually three students, that read, discuss and analyze software descriptions together and present their results to both the teacher and the rest of the class.

Understanding the analysis done and how it was done is very important, since it means students can apply the concepts learned not only in the written exam to pass the course, but also in the future, in other real-life software systems. However, there are several issues regarding the application of theory concepts in this course:

- Each year, there are around one-hundred students signed in the course. All these students have roughly the same Computer Science background knowledge. However, not all students have the same maturity: while some may quickly understand what is taught in theory lessons, others may need some more time to assimilate what was taught.

- The output of the analysis done to software descriptions (the scenarios, etc. extracted) is not available in a consistent way: the analysis done in class is available for students if they took their own notes, and the analysis done by groups is available if:

  1. Students took notes of their colleagues' presentations
  2. Groups share their presentation slides among them. Slides may include errors pointed out by the teacher, but not corrected.

- The case description is usually a fairly long document (from ten to twenty pages approximately). The task of carefully reading and understanding all the text and do a mapping between the concrete descriptions in the text and the abstract concepts learned in theory classes is not easy, as the parts of the text that map to concepts are not always evident.

- The architectural elements extracted from a single software description are usually scattered along the whole text, and it is not evident the connection between all the elements.

Chapter 3

Objectives

To solve the mentioned problems, we intend to develop a platform, where students and teachers collaborate in the analysis and synthesis of the case descriptions, ending on a structured representation of the case descriptions, that is hooked on the concrete description.

The platform will provide ways for annotating the text on the case descriptions, which will help students organizing their thoughts and creating the structure.

To motivate the students to participate in this platform and to increase the collaborative component, elements of social software will be used (such as reputation and tagging systems), to promote, among other things, collaboration, mutual aid, discussion, learning, and even some healthy competition between students.

The existence of this collaborative platform provides not only a way for students to discuss, ask questions and consolidate their knowledge, but also a unique place where their study materials are stored and organized, facilitating their studies.

Chapter 4

Related Work

The platform to develop can be thought of a social software, where different people communicate and collaborate to provide a structured representation of a software description. The next subsections provide an overview on the state-of-the-art of two main aspects:

- **Collaborative work**, presenting the state-of-the-art on social and collaborative aspects of software. This includes literature on:

  - The Honeycomb Framework (a framework that generalizes the most important components of social software);
  - Persuasive Software (systems that have impact on users behavior and thoughts, which is the case of the platform to develop);
  - Roles in Social Networks (describing the types of users of social software);
  - Reputation Systems (attribution of scores to users to provide a motivational component on the platform).

- **Knowledge Structuring**, presenting the state-of-the-art on ways of structuring information. This includes literature on:

  - Collaborative tagging systems (assigning keywords to documents or parts of documents);
  - Semi-structured content (providing a way of structuring knowledge without such strict rules as, for example, an ontology or a taxonomy);
  - Ontology Learning (extracting knowledge from text).

4.0.1 Honeycomb Framework

The term "Social Software" is used in very different contexts [1], and therefore, there are different definitions for it, given by different authors [2, 3, 4, 5, 6].

The definition of Social Software as "systems that allow people, in their particularities and diversity, to communicate (interact, collaborate, exchange ideas and information) mediating and facilitating any kind of social relationship and favoring the emergence of a collective wisdom and a bottom-up organization" [1] applies to our collaborative platform, in the sense that it should allow a large number of students, with different personalities and opinions, to ask questions to their colleagues, give opinions, receive feedback, etc, in order to correctly identify the architectural elements in a description of a software system, and therefore reach a complete and correct analysis of that system.

The Honeycomb Framework was proposed to illustrate the seven elements that give a functional definition for social software [7]:

- **Identity:** the unique identifier of a user within the system;

- **Presence:** resources that allow knowing if certain identity is online;

- **Relationship:** way to determine how users can relate/are related to others;

- **Reputation:** way of knowing the status of a user in the system;

- **Groups:** possibility to form communities of users that have common interests;

- **Conversation:** resources for communication among the users (synchronous and/or asynchronous);

- **Sharing:** possibility of sharing objects that are important to the users (videos, images, etc);

The Identity appears at the center of the framework, because it is the most basic requirement of any social system.

Although this framework identifies the major elements of a social system, it does not, in fact, specifies whether a system is social or not, a software system may implement all of these elements and still not be social.

Analogously, a system can be social with just a couple of elements implemented. So far, the only way to know if a system is social is to check if it complies with the definition (or with one of them).

In the platform to develop it is possible to identify the following elements of the framework:

- **Identity:** The identity is the central element of the framework and it is essential to the platform, as each student is a unique person and, therefore, must have a unique identification inside the platform. Since each student has already got a unique student number inside the whole university, the student number should be used for identification. The name of the student should also be added for purposes of identification, since it helps other users knowing who are they communicating with (it is easier to know who a student is if his name is available).

- **Reputation:** To assess the quality and relevance of the participations in the platform, a reputation system must be used, as discussed in 4.0.4.

- **Groups:** One of the evaluation methods of the Software Architectures' course is the realization of group works, where students form work groups, and each group will have to identify architectural elements in a description of a software system. The notion of these work groups must be present in the platform, since each group is a restricted set of people that are working towards the same goal: Correctly understand and identify the architectural elements in the description.

- **Relationship:** Although there is already an implicit relationship inside the platform (all students are colleagues of each others), this relationship has a specialization. Since students are divided in work groups, students inside a work group have a different relationship they are group colleagues, and this relationship must be differentiated. As the teacher will also be a member of the platform, the relationship student-professor must always be present as well: a student does not have the same kind of interactions he has with a colleague when interacting with a teacher therefore this kind of relationship must be present in the platform.

- **Conversation:** The existence of discussions around elements to add/already added to the platform provides a way of communication between the users.

- **Sharing:** Within the discussions around elements, it should be possible to share different kinds of objects (articles, links, etc), since they may be helpful to understand or complement what is being discussed. Since there are groups inside the platform, there should be present different levels of sharing:

1. *Group level:* Shared objects are only available for a single work group. This way, groups can share documents, links, etc., that are important for completing their assignments, or help understand the description document;

2. *General level:* Documents are shared inside a discussion, an added element, or as an answer to a question asked, and are available to all students in the platform;

- **Presence:** Considering that all communications are asynchronous, the notion of presence is not strictly necessary in the platform. However, it could be added, as a feature, a visible list of people online (logged in the platform) similar to the ones present in many discussion forums.

4.0.2 Persuasive Software Design Patterns

The term "persuasive technology" is used to describe computer systems that have an impact on user's thoughts and may even lead to changes in their behavior [8, 9].

There are three different possible outcomes for a persuasive system:

- Reinforcement - making current attitudes resistant to change;

- Changing Outcome - changes in a person's response to an issue;

- Shaping outcome - formulate a pattern for a situation where one did not exist before [10];

The main users of the collaborative platform to develop will be MSc students attending the 'Software Architecture' course. Even though they are familiar with some of the terms used in the context of the course, new terms are introduced as the course progresses. This platform is persuasive in the sense that it must change the way its users will look at a description of a Software System: reading is not enough, it is necessary to relate what is read with the terms and concepts learned in the classes. Therefore, the most important outcome for this platform is the Changing Outcome: When a description of a system is presented, the students most not only read it, but also relate it to the concepts learned and use the platform to try and extract the correct scenarios and views from it.

The concept of social influence shows that in daily activities, human beings often influence or are influenced by the actions of other people. One's opinion might be a reflection of someone else's opinion, which is also directly or indirectly influenced by cultural norms, mass media, social networks, etc [11]. Therefore, social influence is a change in one's behavior (or attitudes or beliefs), caused by external pressures [12].

In our context, we want the students to contribute to the platform. Either by adding new elements (scenarios, views), discussing the elements added by their colleagues, or asking/answering questions. The platform should provide a way to do this, by allowing:

- Insertion and edition of new elements to the conceptual structure, done by any student;

- Comments (and responses to comments) on an element inserted;

- Discussion Forums and Q&A systems

While some students will want to start contributing to the platform right away, some of them will not initially feel compelled to that. However, as their colleagues and friends add contributions, and since one's actions are somehow influenced by other people, these students may probably start contributing to the platform as well. Even in a social context, because people do not live in isolation, students may talk, ask/answer questions and discuss in person about the system being analyzed in the platform. The outcome of these social interactions may be a new, relevant, contribution to the platform.

Four design patterns are proposed for persuasive systems, in order to introduce social influence in software features: Social Learning and Facilitation (SLF), Competition (COM), Cooperation (COO) and Recognition (REC) [13].

Social Learning and Facilitation (SLF):

The concept of Observational Learning, introduced by the Social Learning Theory, states that people's behavior and skills are augmented through the act of observing others [14], whilst the concept of Social Facilitation states that people's awareness of being observed/evaluated by others also has an influence on their behaviors [15]. Therefore, the Social Learning and Facilitation pattern has the main purpose of enabling and enhancing the design of software features that allow to visualize the presence of other people, with the motivation that it is easier for individuals to pursue their goals if there is a clear awareness of other people pursuing the same goal and facing the same issues [13].

In the context of our collaborative platform, it is necessary to persuade the students to add contributions as they may, at first, not feeling confident enough to add their own contributions, whether they are new elements, comments, questions/answers, etc. The platform should provide a way to visualize the elements recently added or that are currently under discussion as students see their colleagues contributing towards the same goal, they may want to add their own contribution.

Considering that the users of the platform will be students, the Social Learning should have more emphasis over the Social Facilitation. Students have different rhythms of learning, and sometimes a concept may not be correctly understood at first. Asking questions or discussing elements of the analysis will help in the correct assimilation of concepts, and therefore it must be encouraged.

The Social Facilitation comes in the sense that moderation is necessary: while students are encouraged to contribute and ask questions in order to learn, they should strive to add relevant comments and ask relevant questions, as their contributions are seen by everyone.

Competition (COM):

Human beings have a natural drive for competition. Therefore, it can be used as a social support guideline, to motivate users to adopt attitudes and behaviors [9]. The Competition pattern has the main purpose of using competitive elements, such as ranks, scores and levels, that allow users to compare their performance with others, and adjust their target goals based on these. However, as some people might see competition as a motivational factor for improvement, other people may see it as a source of anxiety therefore, participation in a competition must always be voluntary [13].

Adding competitive elements to a social system will, as said above, trigger the human being's natural drive for competition. Adding a score for each student and a rank hierarchy based on the range of scores will start a hopefully healthy competition between students. Each contribution should have a positive score, given either by the teacher or by the colleagues. Less correct contributions should have a lower score, but positive nonetheless. When a student adds something to the platform, it means that he is making an effort to apply the concepts learned in the classes. As said before, not all students can do this correctly at first, and therefore if one of them specifies an incorrect scenario, for example, it shouldn't have a negative score: trying and failing is a way of learning. In addition, when another student adds a contribution to correct his colleagues, he is also gaining score points, and therefore, increasing the competition.

Even this kind of competition, that is supposed to be healthy and motivating, can be a source of anxiety for some people. Therefore, it should be optional for students to be assigned a score and receiving points, or even to show their identity when contributing (add anonymity to the platform).

Cooperation (COO)

Besides competition, reciprocity is also another human beings' natural drive [16, 17], and cooperation between people has been proven to cause positive impact on user behaviors [18]. The Cooperation pattern has the main purpose of providing software features that allow users to engage in mutual goals and provide ways for them, supporting each other in reaching their goals. The motivation for this pattern is that it is easier to cooperate if there is a clear awareness of what needs to be done, what other people are doing, and how far they are in achieving their goals. Group discussion supports these needs and encourages giving and receiving support on the matter. However, some people prefer to perform alone, and therefore cooperation must not be forced [13].

The students that will use this platform will be working towards a main goal: correctly understanding and correlating what was learned in the Software Architectures classes with real examples of software systems. But concerning the analysis of a single software system, there are several concepts and structures to identify on a single description. Students may be having some trouble identifying these elements, and therefore the system must provide a way of defining goals. A goal, for example, could be "Add a new scenario for Availability".

The goals system should work similarly to a discussion forum: Students can add, delete or change goals (changing the description or the title, for example), and for each goal it is possible to send messages to discuss it and ask or answer questions. As discussion around the goal progresses, students collaborate with each other by answering questions asked by their colleagues or correcting what some other student wrote before. Discussions can converge, then, to a correct identification of architectural elements in the platform. A goal should be able to be marked as "in progress" or "complete", to indicate that discussions around that goal are still in progress or have finished. Goals can also be split in sub-goals if the discussion around a single goal becomes too big and complex, it might be wise to be able to split the discussion in sub-goals, each with its own discussion so that when all sub-goals are complete, the main goal will also be nearly or fully complete.

Recognition (REC)

In addition to competition and cooperation, recognition is introduced as another interpersonal motivational factor. This happens because people take pleasure in having their efforts recognized and appreciated by others, in order to feel fulfilled for their achievements [17]. The Recognition pattern has the purpose of providing software features that enable users to get recognition from their peers. The motivation for this pattern is that users, when working towards a goal, they sometimes need a reason to focus on that same goal. Having their efforts recognized may be a good reason to keep the good work, and therefore systems should provide opportunities for public recognition of top achievers [13].

As mentioned before, the users of this platform are students that are pursuing a Masters degree. And just like any other human being, students like to feel that their efforts are recognized. Not only recognition leads to a sense of fulfillment, but also it is a motivational factor to keep up the good work.

Since the system should have a score system, this score should be used as a way of creating a weekly spotlight: by keeping track of the amount of points scored by each student along the week, as the week ends, the list of the Top 10 students with most points gathered in that week should be visible when entering the platform. The list of all students, in decreasing order of weekly points, could also be consulted.

Recognition is then achieved when students see their names in the spotlight. In addition to that, there may be some (hopefully healthy) competition for a spot in the Top 10 - as all, or at least most students will want to have their names on the spotlight.

4.0.3 Roles in Social Networks

A survey on the state-of-the-art regarding identification of roles in social networks ([19]) starts by stating the difference between a "role" and a "position", by giving a definition for both of these concepts:

- A *position* is a "well-defined place" of an individual in a social structure, and is usually associated with certain attitudes, mental health, knowledge, etc., of individuals [19, 20].

- A *role*, in a social structure, is a set of expectations for an individual in a certain position. For example, the role of "secretary" is associated to what secretaries are expected to do [19, 21].

The survey divides the roles in two categories:

- *Non-explicit roles*: roles that are not defined a-priori

- *Explicit roles*: predefined types of actors in the social network, such as "experts" or "influencers".

Methodologies for inferring *non-explicit* roles analyze the structure of the social network and communications between users, in order to identify patterns in communication that allow the identification of roles. Common approaches are based on blockmodels [22, 23] and probabilistic bayesian models [24, 25, 26].

Methodologies for identifying *explicit* roles in a social network compare the activity of the user inside a social network against the criteria that characterizes each role. For example, techniques for identification of "experts" (users that answer correctly to questions asked) inside forums [27] analyze, among other criteria, the number of answers given by the user and how many different people a user answers.

In the platform to develop, it is possible to define a priori the two main roles:

- Student: it is the role assigned to most of the users. Students are the most active users, and their contributions to the platform include tagging parts of the document, adding and editing parts of the semi-structure, asking and answering questions and rate other students' contributions.

- Teacher: this role is assigned to a very small number of users in the platform, but there must be at least one teacher present. Their number of contributions is smaller and they are mostly for correcting (through adding comments to the elements added, or answering questions) the contents added by the students to the platform. Teachers can also give ratings to the contents of the platform. It is up to the teachers to define the weight of their ratings on a case by case strategy.

The methodologies for identification of explicit roles in a social network [27, 28] have in consideration social networks around the web, with hundreds or even thousands of users and where the roles cannot be easily assigned. Given the small amount of users expected in the platform, the roles of Student and Teacher will be assigned in the moment of registration, and there is no need for this methodologies.

A methodology similar to the ones described for identification of experts [27] could possibly be added to the platform. By analyzing user interactions and the ratings saved in the reputation system, a role similar to "expert" could be assigned to some students to identify the ones that do the most relevant contributions to the platform. Students with the role of "experts" could have some responsibilities assigned to them, in order to help other students understanding and applying the concepts. However, given the small number of users in the platform, the number of interactions to analyze may not be enough to do a correct inference.

4.0.4 Reputation Systems

Reputation systems are of extreme importance for certain kinds of applications, namely e-commerce websites (Amazon, eBay, etc.). In these websites, buyers and sellers execute money transactions and

therefore, the reputation of a seller will denote the degree of trust that possible buyers will have in them [29].

Most of the existing literature focuses on these types of applications, and analyses them according to certain defined criteria [29, 30].

For the platform to develop, reputation does not play such an important role as it does for e-commerce and similar applications. For this context, reputation will add a motivational component to the platform: Students rate their colleagues' contributions and ratings are aggregated to calculate a reputation score for each student. Students should strive for getting and keeping a high reputation score. Contributions to the platform include parts of the description that are identified as semantically significant for the software architecture, elements added to the semi-structure, questions and answers added to the Q&A section, comments added (in the discussion forums, or to the elements added).

Motivation:

A study was conducted [31], concerning the lack of participation in Moknowpedia, a Wiki system: People were not motivated enough to overcome any obstacles (lack of time, lack of a structure for the article, etc.) and start contributing to the wiki. A reputation system was added to the wiki in order to solve these problems and improve both content quality and quantity. The system used MediaWiki, a widely used wiki software, and CollabReview, a Java-based web application for reputation management [32].

Results show an increase of 62% in the number of article revisions and an increase of 42% in the number of viewed articles. In the end of the four-month evaluation phase, during which the number of users fluctuated between 18 and 16, 237 reviews had been added to the databases. Overall, the users accessed the wiki more often, read more articles and made more contributions.

This shows the importance of a reputation system in a collaborative system, even if its role is only motivational, and given the similarities between Moknowpedia and the platform to develop (small number of users with similar backgrounds), it is expected that the presence of a reputation system in the platform leads to significant numbers of contributions.

Requirements and Features:

A framework for analysis of reputation systems is proposed in [29], where the general requirements for reputation systems are elicited and the corresponding features needed for their fulfillment. Since these requirements try to include all possible kinds of contexts for reputation systems, not all requirements apply to all systems. Given the context of the platform, the following requirements are identified for its reputation system:

- **R1 (Ratings should discriminate user behavior)** and **R2 (Reputation should discriminate user behavior):**

  The ratings (score that a student assigns to a participation from other student) and the reputation score (the score assigned to a student after processing all ratings given by other students) should have a range of possible values that discriminate all the possible user behaviors, from a student with very few and/or less relevant contributions to a student with many and/or very relevant ones.

- **R3 (The reputation system should be able to discriminate "incorrect" ratings):**

  Despite the small size of the platform to develop, malicious users are present everywhere. Intentionally inaccurate ratings can be given to contributions, in order to increase or decrease a user's reputation score. Therefore, some management for this kind of situations should be present in the platform.

15

- **R11 (Users should not be able to directly modify ratings)**, **R12 (Users should not be able to directly modify reputation values)** and **R13 (Users should not be responsible to directly calculate their own reputation):**

  When being processed, data should not be accessible to users. The reputation score of a student should be calculated by the system and neither this value, nor the values of the ratings attributed should be modifiable by the users.

To satisfy the identified requirements, the reputation system should implement the following features:

- **F1 (Trust/Distrust):** The metrics for reputation and ratings should discriminate the range of all possible user behaviors, from reliable to questionable. In this context, a range of values should be used, where the lowest value represents, as said above, a student with very few and/or less relevant contributions, and the highest value represents a student with many and/or very relevant ones.

- **F2 (Absolute Reputation Values):** The reputation score of a student should not be calculated with respect to another students. Scores should be absolute and calculated independently for each user by the reputation system.

- **F3 (Origin/Target):** To prevent students from self-rating themselves and to control malicious ratings (for example, a student with a lower reputation score could assign lower ratings to their colleagues' participation in order to lower their scores), the origin and the target (students) of each rating given should be identified in the system. By comparing origin and target self-ratings are prevented, and by assigning a weight proportional to the origin's reputation score (a rating given by a student with a lower reputation score will have a smaller weight when processing) malicious users are controlled.

Components of a Reputation System

In [33, 30], reputation systems are divided in four components and it is defined a set of criteria for evaluation:

- **Input:** the process of collecting reputation information from information sources

- **Processing:** the procedure of computing and aggregating the reputation information

- **Output:** the dissemination of the reputation information

- **Feedback Loop:** collection of feedback of the output (review of the review). Does not apply to the context of the platform.

1. **Input:** Concerning input, a series of evaluation criteria is defined [30], from which the following apply to the platform:

   - *Collection channel:* The collection channel is the way how a reputation system collects information. This platform will use a collection channel of type 1 *(CC1)*: ratings are left directly in the platform, by providing a way of rating for each element that can be added.

   - *Information Source:* The evaluators that provide the ratings to the system. The information source in this context are the students logged in the platform, and is measured by:

- **Information Source Scale** (whether a reputation system has restrictions on the sources): In this context there is a type 2 Information Source Scale *(ISC2)*, which means only registered users can assign ratings. This makes sense, since only students registered in the platform are familiarized enough with the contents to provide ratings.

- **Granularity** (how information sources relate with the target): This concerns people with very different backgrounds being sources of information in very different domains. Some reputation systems can require a certain reputation score in a certain domain in order for a user to be able to provide ratings. Since there is only a main domain in this platform, the granularity of the information source is of type *GRN1*: There are no restrictions on evaluators. This means every student can rate their colleagues' contributions.

- *Reputation Information:* The information collected can be classified by:

  - **Breadth**: Number of properties collected. In this context, only a single property is collected: the ratings assigned.

  - **Format**: The format of the reputation information. In this reputation system, the format should be of type *IPF1*: A rating scale, that provides a range of possible scores to assign to elements in the platform.

- *Collection Costs:* The time cost of collecting and processing a single rating. Since the reputation system is small and not a very relevant part of the whole platform, the time costs for collecting and processing ratings are not relevant.

2. **Processing:** Concerning the Processing component, a set of criteria is defined for its evaluation [33], from which the following applies to the platform:

- *P1(Target rating algorithm)*: The algorithm used to aggregate the ratings assigned to students. There are many algorithms that can be used, from the most simple (summation, average, percentage) to the most complex (Bayesian system and fuzzy models). Given the simplicity of the reputation system to develop, a weighted average algorithm is the algorithm that fits best for the context. Assigning weights when calculating the average is important because, as said above, users with lower reputation scores have a probability of also becoming malicious users in the platform, by, for example, giving low ratings to other users in order to decrease their reputation scores.

- *P4 (Update Frequency)*: how often the system updates the reputation information. Again, give the simplicity and the small amount of users, information should be updated as soon as the ratings are assigned.

- *P6 (Algorithm complexity)*: refers to the complexity of each algorithm. The weighted average algorithm is a very simple algorithm, with low complexity.

- *P7 (System complexity)*: refers to the complexity of the whole system. The reputation system to develop is, as mentioned before, very simple and therefore has very low complexity.

The remaining criteria did not apply to the reputation system to develop due to its simplicity. For example, criteria P3 (Feedback aggregation algorithms) identifies how feedback ratings are aggregated and is concerned with the information collected from the Feedback Loop (reviews of the reviews). Since this Feedback Loop does not apply to the reputation system to develop, neither does P3.

3. **Output:** Concerning the Output component, there is also a set of defined criteria [33] for both the reporting and dissemination of reputation information For dissemination, the following criteria apply:

- **O1** (Set of end users): this refers to the users that will be able to see the reputation score assigned to the users of the system. In this case, the set of end users is U_r (the users registered in the platform).
- **O2** (Access methods): this criteria focuses on whether reputation systems provide alternative ways for users to get the reputation information other than the website. In this platform there is no need for an alternate dissemination of information, since reputation is a minor component of the whole platform. Therefore, no other access methods should be provided.

Concerning information reporting, two kinds of information are defined [33]:

- Aggregated (the results of the Processing component)
- Individual (all individual ratings)

While individual information is necessary for some kinds of applications (for example, Amazon shows the information collected in every single review, which is important because users are mostly likely to describe the reason for the rating score attributed), it does not add anything relevant to the context of the reputation system to develop, since only a single rating value is collected. Therefore, only the criteria for aggregated information applies:

- **O4** (Descriptive dimensions): this criteria identifies how many dimensions are used to illustrate the aggregated information. Only a single dimension is used, the weighted average of the ratings, calculated by the Processing component of the reputation system.

O3 (Timeliness) concerns how the reputation system shows the evolution of reputation information, we do not consider relevant to the system to develop.

4.0.5 Collaborative Tagging

Collaborative tagging is the practice of allowing anyone to freely attach keywords or tags to content [34].

While in some document repositories or digital libraries there is an authority responsible for assigning and organizing documents by keywords, in collaborative tagging anyone is able to freely attach keywords or tags to the contents. This is very useful when there is no authority for organizing content or simply there is too much content for a single authority to organize: It is the case of the web [34].

Tagging-based systems contrast with taxonomies: while taxonomies organize contents into unambiguous categories that are within more general categories, tagging is neither exclusive nor hierarchical, and therefore it alows to identify content as being about a great variety of things simultaneously [34].

However, both these methods of organizing information have problems with the word semantics, namely polysemous words (a single word with many, related, senses), synonyms (multiple words with the same/closely related sense), and the basic level variation: different people consider different levels of specificity for describing the same entity for example, a person may describe a dog as 'dog' or as 'beagle') [35].

The tags added to content may describe the content itself, or describe the category in which the content falls [36]. It is possible to identify several functions for the tags, within the system that uses the tagging system [34].

In this platform we are aiming for using tagging as a way of identifying the main parts of a software system within a software description, namely stakeholders, scenarios and their parts, tactics, etc.

For that, and to avoid problems of word semantics, tags added to the text of the description must belong to a closed taxonomy of Software Architecture terms, which will contain all terms necessary to describe an Architecture, organized hierarchically.

It will be possible to elicit relationships between tags in the text (for example, two tags X and Y added to two different parts of the text might have some kind of relationship like is-tactic-for-scenario(X,Y)), which will then facilitate the construction of semi-structured contents.

4.0.6 Semi-Structured Content

Knowledge can be obtained from many different resources, ranging from unstructured (for example, language models obtained from plain text) to structured ones (for example, ontologies) [37].

Unstructured resources are as simple as collections of text, images and other media contents. Thanks to the Web, it is possible to collect huge amounts of unstructured information such as raw text, which allowed major advances in the Natural Language Processing field. However, unstructured knowledge has its limitations: the resources do not provide all the knowledge necessary for complex inference chains [38] and the information is not ontologized not included within a semantic network of unambiguously defined concepts and their semantic relations [37].

Structured resources are machine-readable, and there are several kinds, such as Thesauri (collections of related terms), Taxonomies (hierarchically structured classification of terms) and Ontologies (knowledge model that includes concepts, relations between them and even axioms and rules). These kinds of resources provide information of the highest quality, since their contents were developed with the contribution of experts. However, they require a huge amount of effort in creating and maintaining them, which is very difficult and time-consuming. Also not all topics are covered by the experts, and information might even be slightly culturally-biased. And since these resources are manually annotated, it is hard to keep updating them and they might not contain the lexicalizations of the concepts in different languages [37].

Semi-structured contents try to create a middle-ground between these two types. The most important example of their use is the Wikipedia.

Wikipedia consists of a repository of webpages, each of them containing an entry about a certain concept. There are relations between the different kinds of pages, for example, the redirections (several concepts that redirect for the same article), internal hyperlinks (links for articles about concepts that were included in the text), interlanguage links (links for the same article written in different languages) and category pages (used to classify entries). Pages can also contain infoboxes (tables summarizing the most important information).

Wikipedia then relies on large amounts of mannually-input knowledge, provided via massive online collaboration. Information is ontologized with high quality thanks to the collaborative editing of articles and it's possible to keep information continuously updated and achieve a wide coverage for almost all domains.

To sum up, using semi-structured resources provides the best of both worlds: high quality information with wide coverage of almost all domains [37].

The main goal of this platform is to allow students to extract a semi-structured representation of a software system from a description in plain-text - have all the parts that constitute a software system (Stakeholders, Scenarios, Tactics, Views, etc) described in a semi-structure (similar to a Wikipedia page), instead of scattered along a ten or more pages article.

The idea is to create templates for all the possible entities that can be added to the semi-structured representation: a template for describing stakeholders, another for describing scenarios, views, styles and so long. Each template should, then, allow a correct description of a component. For example,

a scenario, which captures and expresses quality requirements, is defined by a stimulus (a condition arriving at the system), a source of stimulus (entity that generates it), an artifact (the part of the system being stimulated, an environment (the conditions of the system), the response (activity after the stimulus) and a response measure (some way of measuring the response). Whilst the stimulus and the response are mandatory in a scenario, it may not contain all the parts described.

When adding a component to the semi-structure, students will follow the template and fill it with their own words (for example, for describing each part of a scenario). However, all this information should be somehow linked with the parts of the text that describe the scenario.

By using terms from a taxonomy to tag parts of the text and allowing elicitation of relationships between terms (described below) and linking from the semi-structure to a tag in the text, this problem is solved and people are able to create a bridge between the text in the description and the semi-structured content, even if the information for a single component is scattered along the text.

In the field of Artificial Intelligence and Natural Language Processing, there are methods for extraction of machine-readable information from semi-structured contents, namely thesauri and relationships extraction, and for enrichment of structured information with semi-structured content (that also extracts relationships for taxonomy and ontology induction) [37]. In this platform there is no need for Artificial Intelligence techniques, but the concept of relationship extraction is very important. As said before, students are allowed to tag parts of the text from the description with terms from a taxonomy of Software Architecture concepts - namely tag a part of the text as a scenario, a stimulus, a response measure, etc. But this tagged text may be related: a part of text tagged as 'stimulus' may be a stimulus for another part marked as 'scenario' (there is a relationship of the kind is-stimulus-of-scenario). Providing method for elicitation of these relationships allows for a more deep understanding of the description text, by linking the components added to the semi-structure with the tags and relationships elicited from the text.

And since this is a collaborative platform, it is expected that having many people working together in semi-structuring a software description will increase the quality of the final product. As students discuss and correct each others' work, the semi-structure extraction becomes more complete, and students also have an opportunity to learn from their mistakes - since sometimes it is not easy to relate the concepts learned in a classroom with real-life applications.

4.0.7 Ontology Learning

Research on Ontology learning from text has evolved over the years and there are several open challenges for this field [39]. Ontologies are defined as "effectively formal and explicit specifications in the form of concepts and relations of shared conceptualizations" [40], and can be thought of as a directed graph, with concepts as nodes and relations as edges.

Techniques for ontology learning can be classified in statistics-based, linguistics-based, logic-based or hybrid. Over the years there was also an increased interest in techniques for ontology learning from social data. Existant literature investigates the problem of ontology learning from user-defined tags [41], discusses the requirements for automatic ontology learning from social data [42], presents a tripartite ontology model [43], and describes an approach to complement corpus-based ontology learning with tags [44].

Regarding the context of the platform, all the concepts used to tag the document contents (the concepts of Software Architectures) will be defined a priori and organized according to the relations between them (IS-A relations and others).

Software descriptions analyzed in the platform do not contain these concepts. However, the students' motivation is to find parts of the text that correspond to the concepts predefined and tag them accordingly (for example, a single sentence of the text can identify a stakeholder, or a whole paragraph can identify

the stimulus of a scenario).

Since the relations between concepts are already defined, it should be possible for students to elicit these relations between the tagged parts of the text. This facilitates the comprehension of the text contents and the creation of the semi-structure as mentioned previously.

Chapter 5

Solution

To solve the problems elicited throughout this document, it was developed a Web Application. This kind of application facilitates collaboration between its users, and therefore was the more adequate choice for the system to develop. The developed application features:

- An authentication system, where users can login into the system. This authentication system provides an unique identity for a user inside the application, and also provides distinction between types of users, as there are users of type STUDENT and users of type TEACHER, with different permissions.

- Document Management, only available for teachers, where software description articles can be added or removed;

- Document parsing into a view;

- Creation of annotations in the document text. As the name says, an annotation is a part of the text that is marked and highlighted. The AnnotatorJS (REFER) library provides tools not only to mark up parts of text, but also to associate tags and user text to that selected text.

- Templates for a set of Software Architectures concepts. The parts of marked text described before can be associated to these templates, thus making it easier to co-relate the theoretical concepts with the practical applications. The concepts used in these templates are represented as entities of the domain model.

The next chapter will describe the developed system architecture.

Chapter 6

Software Architectures Concepts

Before describing the architecture of the developed solution, it is necessary to provide a small introduction to the most important concepts from the Software Architectures course.

6.1 Scenarios

A scenario is used to capture and express quality requirements of a system. It consists of six parts [45]:

- **Source of Stimulus:** Some entity (a human, a computer or any other actuator) that generates the stimulus;

- **Stimulus:** A condition that arrives at the system;

- **Environment:** The system condition when the stimulus occurs;

- **Artifact:** Part of the system that was stimulated;

- **Response:** Activity undertaken after the arrival of the stimulus;

- **Response Measure:** when the response occurs, it should be measurable in some way, so the requirement can be tested;

Tactics are design decisions used to achieve the quality requirements expressed by the scenarios.

6.2 Views

A view is a representation of a set of system elements and the relationships associated with them [46]. This set of elements and relationships is constrained by viewtypes.

A Viewtype defines the element types and relationship types used to describe the architecture of a software system from a particular perspective[46]. Viewtypes refine into styles.

An architectural style is a specialization of element and relation types, together with a set of constraints on how they can be used[46].

Views can fall into three viewtype categories[46]:

- **Module Viewtype:** document the system principal units of implementation.

  The elements of this viewtype are the *Modules*, which is are implementation units.

Relationships between modules can be of type *"Is-part-of"*, which defines a part-whole relationship, *"Depends on"*, which defines dependency relations, and *"Is-a"*, which defines a generalization/specialization relationship.

- **Component & Connector Viewtype:** document the system units of execution.

  The elements are the Components, which are the principal processing units and data stores, and the Connectors, which are pathways of interaction between components.

  The relationships can be of type *"Attachment"*, which associate components to connectors, and *"Interface"* Delegation, which associates component ports to other ports from an "internal architecture"- and similarly for the connector.

- **Allocation Viewtype:** document the relationships between a system's software and its development and execution environments.

  The elements are the *Software Element* and the *Environmental Element*.

  The relationships are of type *"Allocated-to"*, which means that a software element is allocated to an environmental element.

Chapter 7

Architecture

The application developed follows a Model-View-Controller architecture

7.1 Model

The Model describes the main entities of the system. Inserir figura.

The Document is one of the most important entities of the Domain Model, because it aggregates all the concepts elicited during text analysis. Annotations are first added to the document before being linked to other domain entities, and all SA concepts discovered through reading are connected either directly or indirectly to this entity.

Annotations are constituted by a user comment, a tag and a selection of text from the document. Therefore, annotations are always linked to the document from which they were created.

7.2 Controllers

7.2.1 Annotation Controller

The Annotation Controller provides a way to manage the annotations added to a document. It communicates with the Domain Model to create, remove or update Annotations. The particularity of this controller is that it provides a REST API, which is used to retrieve a JSON representation of the annotations stored in the model and display it along with the document text. The endpoints provided by this controller are described in Table 7.1.

Name	Method	Endpoint	Description
Index	GET	../store/annotations	Returns the set of annotations associated with a specific document
Read	GET	../store/annotations/id	Returns the annotation with the specific id
Create	POST	../store/annotations	Creates a new annotation, stores it in the model, and redirects to the Read endpoint
Update	PUT	../store/annotations/id	Updates the annotation with the given id and redirects to the Read endpoint
Delete	DELETE	../store/annotations/id	Removes the annotation with the given id. The response is a HTTP/1.0 204 NO CONTENT.

Table 7.1: REST API provided by the Annotation Controller

7.2.2 Document Controller

The Document Controller handles the requests to view, add or remove a document from the system.

7.2.3 Scenario Controller

The Scenario Controller handles the construction of templates for Scenarios. All requests to create a new scenario, link document annotations to a specific scenario or delete a scenario are handled in this controller.

Chapter 8

Evaluation

Evaluation here...

Chapter 9

Conclusion

The success of social software is very dependent on the people that use it, since it is focused on the interactions and collaboration between their users.

The implemented platform will be tested in the context of Social Architectures classes. Students will be encouraged to use all the platform's features and provide feedback throughout the testing period.

After the testing period, a survey should be done to students in order to evaluate the platform in terms of Usability (ease of use) and Performance. Additionally, an assessment will be done on the quality and effectiveness of the knowledge synthesis.

Bibliography

[1] R. Pereira, M. C. C. Baranauskas, and S. R. P. da Silva, "Social software building blocks: Revisiting the honeycomb framework," in *Information Society (i-Society), 2010 International Conference on*, pp. 253–258, IEEE, 2010.

[2] C. Shirky, "A group is its own worst enemy," in *The Best Software Writing I*, pp. 183–209, Springer, 2005.

[3] R. Klamma, M. A. Chatti, E. Duval, H. Hummel, E. T. Hvannberg, M. Kravcik, E. Law, A. Naeve, and P. Scott, "Social software for life-long learning," 2007.

[4] B. E. Kolko, E. Johnson, and E. Rose, "Mobile social software for the developing world," in *Online Communities and Social Computing*, pp. 385–394, Springer, 2007.

[5] D. L. Wiley, "Tagging: People-powered metadata for the social web," 2008.

[6] M. A. Chatti, M. Jarke, and D. Frosch-Wilke, "The future of e-learning: a shift to knowledge networking and social software," *International journal of knowledge and learning*, vol. 3, no. 4, pp. 404–420, 2007.

[7] G. Smith, "Social software building blocks," *Retrieved November*, vol. 5, p. 2010, 2007.

[8] B. J. Fogg, "Persuasive technology: using computers to change what we think and do," *Ubiquity*, vol. 2002, no. December, p. 5, 2002.

[9] H. Oinas-Kukkonen and M. Harjumaa, "Persuasive systems design: Key issues, process model, and system features," *Communications of the Association for Information Systems*, vol. 24, no. 1, p. 28, 2009.

[10] H. Oinas-Kukkonen and M. Harjumaa, "Towards deeper understanding of persuasion in software and information systems," in *Advances in Computer-Human Interaction, 2008 First International Conference on*, pp. 200–205, IEEE, 2008.

[11] P. Mavrodiev, C. J. Tessone, and F. Schweitzer, "Quantifying the effects of social influence," *Scientific reports*, vol. 3, 2013.

[12] R. E. Guadagno and R. B. Cialdini, "Preference for consistency and social influence: A review of current research findings," *Social Influence*, vol. 5, no. 3, pp. 152–163, 2010.

[13] M. Oduor, T. Alahäivälä, and H. Oinas-Kukkonen, "Persuasive software design patterns for social influence," *Personal and Ubiquitous Computing*, vol. 18, no. 7, pp. 1689–1704, 2014.

[14] A. Bandura and D. C. McClelland, "Social learning theory," 1977.

[15] R. B. Zajonc *et al.*, *Social facilitation*. Research Center for Group Dynamics, Institute for Social Research, University of Michigan, 1965.

[16] R. B. Cialdini, "Influence: The psychology of persuasion," 1993.

[17] T. W. Malone and M. R. Lepper, "Making learning fun: A taxonomy of intrinsic motivations for learning," *Aptitude, learning, and instruction*, vol. 3, no. 1987, pp. 223–253, 1987.

[18] A. Stibe, H. Oinas-Kukkonen, and B. White, "Exploring the effects of social influence on user behavior targeted to feedback sharing," in *Proceedings of the IADIS WWW/Internet Conference (ICWI), Madrid, Spain*, 2012.

[19] M. Forestier, A. Stavrianou, J. Velcin, and D. A. Zighed, "Roles in social networks: Methodologies and research issues," *Web Intelligence and Agent Systems*, vol. 10, no. 1, pp. 117–133, 2012.

[20] S. P. Borgatti and M. G. Everett, "Notions of position in social network analysis," *Sociological methodology*, vol. 22, no. 1, pp. 1–35, 1992.

[21] S. Nadel, "The theory of social structure.," 1957.

[22] S. P. Borgatti and M. G. Everett, "Two algorithms for computing regular equivalence," *Social Networks*, vol. 15, no. 4, pp. 361–376, 1993.

[23] R. L. Breiger, S. A. Boorman, and P. Arabie, "An algorithm for clustering relational data with applications to social network analysis and comparison with multidimensional scaling," *Journal of Mathematical Psychology*, vol. 12, no. 3, pp. 328–383, 1975.

[24] M. Steyvers, P. Smyth, M. Rosen-Zvi, and T. Griffiths, "Probabilistic author-topic models for information discovery," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 306–315, ACM, 2004.

[25] A. McCallum, X. Wang, and A. Corrada-Emmanuel, "Topic and role discovery in social networks with experiments on enron and academic email.," *J. Artif. Intell. Res.(JAIR)*, vol. 30, pp. 249–272, 2007.

[26] A. Daud, J. Li, L. Zhou, and F. Muhammad, "A generalized topic modeling approach for maven search," in *Advances in Data and Web Management*, pp. 138–149, Springer, 2009.

[27] J. Zhang, M. S. Ackerman, and L. Adamic, "Expertise networks in online communities: structure and algorithms," in *Proceedings of the 16th international conference on World Wide Web*, pp. 221–230, ACM, 2007.

[28] N. Agarwal, H. Liu, L. Tang, and P. S. Yu, "Identifying the influential bloggers in a community," in *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pp. 207–218, ACM, 2008.

[29] S. Vavilis, M. Petković, and N. Zannone, "A reference model for reputation systems," *Decision Support Systems*, vol. 61, pp. 147–154, 2014.

[30] L. Liu, M. Munro, and W. Song, "Evaluation of collecting reviews in centralized online reputation systems.," 2010.

[31] S. Dencheva, C. R. Prause, and W. Prinz, "Dynamic self-moderation in a corporate wiki to improve participation and contribution quality," in *ECSCW 2011: Proceedings of the 12th European Conference on Computer Supported Cooperative Work, 24-28 September 2011, Aarhus Denmark*, pp. 1–20, Springer, 2011.

[32] C. R. Prause and S. Apelt, "An approach for continuous inspection of source code," in *Proceedings of the 6th international workshop on Software quality*, pp. 17–22, ACM, 2008.

[33] L. Liu and M. Munro, "Systematic analysis of centralized online reputation systems," *Decision support systems*, vol. 52, no. 2, pp. 438–449, 2012.

[34] S. A. Golder and B. A. Huberman, "Usage patterns of collaborative tagging systems," *Journal of information science*, vol. 32, no. 2, pp. 198–208, 2006.

[35] J. W. Tanaka and M. Taylor, "Object categories and expertise: Is the basic level in the eye of the beholder?," *Cognitive psychology*, vol. 23, no. 3, pp. 457–482, 1991.

[36] T. Coates, "Two cultures of fauxonomies collide," *http://www. plasticbag. org/archives/2005/06/two_cultures_of_fauxonomies_collide/. Last access: May*, vol. 8, p. 2008, 2005.

[37] E. Hovy, R. Navigli, and S. P. Ponzetto, "Collaboratively built semi-structured content and artificial intelligence: The story so far," *Artificial Intelligence*, vol. 194, pp. 2–27, 2013.

[38] P. Domingos, "Toward knowledge-rich data mining," *Data Mining and Knowledge Discovery*, vol. 15, no. 1, pp. 21–28, 2007.

[39] W. Wong, W. Liu, and M. Bennamoun, "Ontology learning from text: A look back and into the future," *ACM Computing Surveys (CSUR)*, vol. 44, no. 4, p. 20, 2012.

[40] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[41] J. Tang, H.-f. Leung, Q. Luo, D. Chen, and J. Gong, "Towards ontology learning from folksonomies.," in *IJCAI*, vol. 9, pp. 2089–2094, 2009.

[42] K. Kotis and A. Papasalouros, "Automated learning of social ontologies," *Ontology Learning and Knowledge Discovery Using the Web: Challenges and Recent Advances*, 2011.

[43] P. Mika, "Ontologies are us: A unified model of social networks and semantics," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 1, pp. 5–15, 2007.

[44] A. Weichselbraun, G. Wohlgenannt, and A. Scharl, "Augmenting lightweight domain ontologies with social evidence sources," in *Database and Expert Systems Applications (DEXA), 2010 Workshop on*, pp. 193–197, IEEE, 2010.

[45] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. SEI series in software engineering, Addison-Wesley, 2003.

[46] P. Clements, *Documenting Software Architectures: Views and Beyond*. SEI series in software engineering, Addison-Wesley, 2003.

Appendix A

Appendix chapter