Czech Technical University in Prague
Faculty of Information Technology

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Wireless Software System for Quiz Competition Game and Information System for Game Community
Student:	Vladimir Vorobyev
Supervisor:	Ing. Miroslav Balík, Ph.D.
Study Programme:	Informatics
Study Branch:	Information Systems and Management
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2016/17

Instructions

A) Design a software system (SWS) that supports a quiz competition game (a game between several players who are answering the same questions in the same time).
Requirements:
Must-have:
1. Instance of SWS consists of client (player) and server (moderator) components that communicate via WLAN using a communication protocol (CP).
2. SWS runs on PC (OS Ubuntu 14.04+).
Nice-to-have:
1. SWS runs on PC (OS Windows 7+, OSX 10.9+) and on mobile devices (OS Android 4.1+, iOS 8+).
2. SWS contains tools for customizing game process and tools for establishing wireless communication.
3. SWS is friendly to a user without advanced IT skills.
B) Design CP.
C) Implement a prototype of SWS that responds to must-have requirements.
D) Design an information system (IS) that supports the community of SWS users (deploying applications, shared database of questions, organizing games, etc.).
E) Analyze integration of SWS and/or IS with social networks (Facebook, Google+, etc.).
F) Analyze business applications of SWS and IS.

References

Will be provided by the supervisor.

L.S.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague February 8, 2016

Insert here your thesis' task.

Czech Technical University in Prague

Faculty of Information Technology

Department of Software Engineering

Bachelor's thesis

Wireless Software System for Quiz Competition Game and Information System for Game Community

Vladimir Vorobyev

Supervisor: Ing. Miroslav Balík, Ph.D.

16th May 2016

Acknowledgements

Firstly, I would like to thank Ing. Miroslav Balík, Ph.D. for his patience, attention and valuable assistance while supervising this thesis. Then, I appreciate English language advices from Snezhanna Markova. I am also thankful to Lilia Grozovski, the president of Verein Intellectus Helvetici (Bern, Switzerland) for her interview and to Alexei Golovko, Anna Lebedeva, Jevgenij Cist'akov, Maiia Grauden, Mikhail Titenko, Petr Polezhaev, Yevgeniya Chekh and Yulia Sheveleva for their time and patience while being while them being software testers. Last but not least, I am grateful to my friends and family for their priceless support.

Declaration

In Prague on 16th May 2016 . . . .

Citation of this thesis

Vorobyev, Vladimir. *Wireless Software System for Quiz Competition Game and Information System for Game Community.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

Hlavní autorská motivace je tvorba pohodlné a moderní platformy nutnou pro budování globální komunity kvizových her. Práce obsahuje výzkum komponent platformy, jejích integrace mezí sebou a s sociálními sítí a možné podnikové využití. A také byly provedeny implementace a uživatelské testování funkčního prototypu jedné z komponent.

Klíčová slova Informační systém, softwarový systém, soutěžní kvízová hra, bezdrátová lokální síť, herní komunita, integrace se sociálními sítí, Wi-Fi, Java

Abstract

The major author's motivation factor is creating a convenient and modern platform needed for building global quiz games community. The thesis contains studies of its components, their integration with each other and with social networks, and a possible business application of the platform. Also, implementation and user testing of a functioning prototype of one of the components was performed.

Keywords Information system, software system, quiz competition game, wireless local area network, game community, social network integration, Wi-Fi, Java

Contents

List of Figures

Introduction

According to the Merriam-Webster dictionary, a quiz game is “a form of entertainment ... in which the members of a panel compete in answering questions”[1].

One of the specific subsets of these games became an inspiration for this thesis. This subset is the games, popular in *intellectual games* (a.k.a. *Čto? Gde? Kogda?* [*What? Where? When?*]) community. In short, the main features of this phenomenon are competition in answering questions (that usually requires not only knowledge, but numerous other skills including analysing, intelligence, teamwork, etc.) and, more importantly, the special atmosphere in the game community (which mostly consists of students and intellectuals). This community began in the late 1980s from attempts playing an “off-line version” of the Soviet TV quiz game *Čto? Gde? Kogda?*.

The variety of games has increased as the community develops by borrowing ideas from television and inventing new types of games. Some of them require special means for estimating the results of competitive actions. One such game is *Svoâ igra*[*Their own game*][2] which is based on the American TV show *Jeopardy!*[3]. The author of this thesis proposes that creating such means through modern information technology is an interesting task.

To a lot of members of the game community, their hobby is not simply a game, but a way to meet new people of their type, to communicate and to make friends with them. It might be appropriate to say that this community is a specific subculture, based on the values of friendliness, intelligence and a sense of humour. Creating and implementing an information system and software might create an opportunity to modify this subculture and create a more advanced alternative, making these games a more popular and cosmopolitan phenomenon, rather than known only to a small number of Russian-speaking communities. This is another major motivation factor for the author.

Software and sources

During the work over this thesis the following software is used:

Kile [4] for creating LaTeXsource for some documents,

Netbeans [5] for developing software,

Lucidchart [6] for creating diagrams,

GIMP [7] for creating software icons and editing diagrams,

Kdenlive [8] for editing audio and video recordings,

LibreOffice [9] for creating some texts and processing survey results,

Kazam Screencaster [10] for making audio and video recordings,

Skype [11] for taking an interview,

Git [12] for versioning source codes of the developed software and of this document.

All photographic images used in this thesis are credited to the author.

All cited sources are presented in the Bibliography. The sources descriptions that are originally presented with non-Latin signs were transliterated according to *ČSN ISO 9*[13] as required by *ČSN ISO 690*[14]. Also, all transliteration of Russian words in the text (except personal names: their transliterations are given in the variation proffered by their owner) is made by *ČSN ISO 9* (for this [15] was used).

Chapter 1

Goals and Objectives

The main goal of this thesis is to design a new platform for quiz competitive games that is friendly to a greater number of users than the existing alternatives. This means that components of this new platform must combine strong aspects of state-of-the-art solutions and avoid their drawbacks. Also, the new solution will be more valuable if it contains new features and improves or extends the existing ones.

To achieve this main goal, the following objectives must be completed:

- **To design a Software system (SwS) that supports the game process of a competitive quiz game.** This system is meant to modify classical quiz games between human players who are physically present in the same location. The idea is to make the game process more comfortable and keep the advantage of live conversation, which is lost in distant online communication.

- **To design an Information system (IS) that supports the community of SwS users.** This part of the platform helps the community with game organization, communication and statistics.

- **To analyse options of integration of SwS and IS.** Games are the main reason for game communities to exist. SwS supports singular games and IS arranges the life of the game community, which is why a lot of information is transferred between them. This transfer is more effective and convenient if the systems are integrated.

- **To analyse options of integration of IS and/or SwS with social networks.** Social networks play a significant role in the life of today's individuals and organizations. And this role is increasing with the spread and development of mobile devices and the internet. That is why integration with social networks can make the platform more attractive for users.

- **To analyse options of business application of the designed platform.** Although the author suggests designing a great platform to be the main goal of this thesis, it is obvious that design is useless if the designed item is not supposed to be implemented. And implementation is useless if the item is useless itself. That is why the item needs to have business applications. This analysis not only provides practical information about using the platform in real life, but also answers a philosophical question: is there a reason for creating this project?

Chapter 2

Software system

SwS for a quiz competition game is a collection of programs which run and interact with each other to support the game process. This chapter contains an overview of existing software and equipment that is used for this purpose; analysis and design of SwS and realisation of the SwS prototype.

2.1 Introduction

The game, for which SwS is designed, is based on *Svoâ igra* (a Russian clone of the American TV show *Jeopardy!*). A popular name of this game is *Svoâk*. This game is chosen because it requires special equipment for reliable data estimation, which affects the competition.

2.1.1 Game process

The game process is described by the following algorithm:

The number of players is greater than or equal to two. Also, one moderator is needed for the game. At the beginning there is a set of questions that have a price and a topic.

1. Topics and prices of the questions are represented in a table. The player that gave the last correct answer (if such exists) or a random player chooses a question Q.

2. The chosen question Q is read by the moderator and subtracted from the set. If a player applies for an answer in this step, the price of the question is subtracted from his/her score and this player cannot apply until the next question; this issue is called a *false start*.

3. Players apply for an answer. If the player P applies before his/her opponents, this player gets the right to answer. If nobody has applied

in a certain period of time or no player can answer this question, the game proceeds from step 5.

4. The player P, who has the right to answer, answers. If the answer is correct, the moderator adds the price of the question to the P's score. Otherwise the moderator subtracts the price from the P's score and the game proceeds from step 3, while P cannot apply for an answer on the question Q.

5. If the set of questions is empty, or if the game time limit is reached, the game finishes. Otherwise the game proceeds from step 1.

The player with the highest score wins the game. This game has variations that differ in the details of the game process. For example, questions can be picked sequentially by price or topic, without players' choice. Another variation is to ignore a player's false start, without loss of score or the right to answer.

Further in this text the author uses the term *Game* meaning the quiz game described by the algorithm above (if another meaning is not obvious from the context).

2.1.2 General concept of the SwS

The most problematic part of the game process is handling competition for the right to answer. That means that main purpose of the SwS is estimating the result of this competition, i.e. which player has applied first. It also must correctly process the applying from the other players and help the moderator handle score changes. It must make the game process more flexible by allowing variations in a wide range of preferences. A feature that allows to organize games within larger competitions (like championships or tournaments) is also a useful addition. Above all, the system must be user-friendly and comfortable.

2.2 State-of-the-art

In this section the most remarkable software and non-digital solutions for supporting *Game* are reviewed.

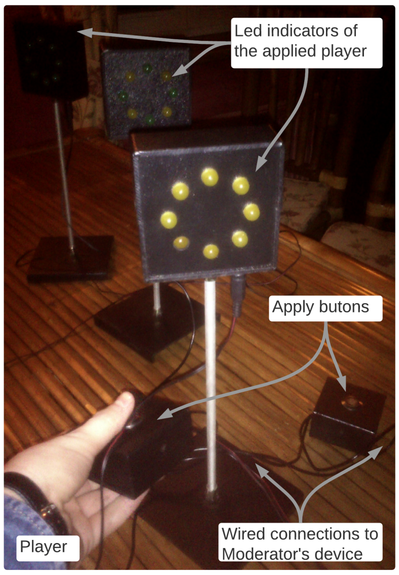
2.2.1 Non-digital alternatives

The easiest and obviously the most uncomfortable approach is applying for the answer with voice, flapping, gesture, etc.

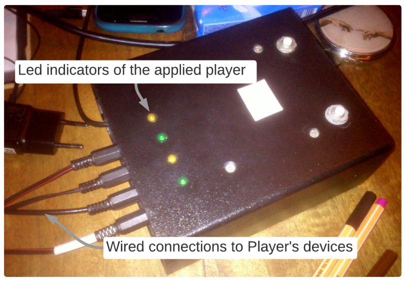
A more advanced and the most popular solution are analog systems (Figure 2.1) that usually consist of player's devices (Figure 2.1a), used for applying for an answer, which are connected with wires to a moderator's device (Figure 2.1b) which processes the applying (one of these systems is described in

[16]). These systems are usually assembled by small companies (like [17]) or by some game lovers themselves.

Despite their appropriate handling of competition for applying, these systems still have several major drawbacks. Firstly, most of them do not process the score. The moderator has to use external means (paper, applications like *MS Excel* , etc.) to count it. Secondly, the wires create various problems. They make setting up and tearing down the system uncomfortable and decrease performance stability due to the risk of accidental unplugging. Sometimes wires also make walking in the locations they are used in difficult or even dangerous. Finally these systems are usually rather heavy and large, and because of this, not very suitable for transporting.


Led indicators of the applied player
Apply butons
Player
Wired connections to Moderator's device

(a) Player's devices. Used by players for applying.


Led indicators of the applied player
Wired connections to Player's devices

(b) Moderator's device. Handles timing and competition. External sources like mobile devices or paper are used for placing questions and computing the score.

Figure 2.1: A typical analog system

2.2.2 Software

2.2.2.1 SImulâtor

SImulâtor (created by Vladimir Khil [18]) is a free of charge software simulator of *Svoâ Igra*. This application can be used together with analog systems, but it also supports emulating these systems with devices connected to a Wi-Fi network [19]. The concept of this solution is the following. The moderator launches the *SImulâtor* application on a device connected to a Wi-Fi network.

Then he or she announces the IP address of this device and the port used by the application for running the web-page with the player's interface. Then players connect their devices to the network and use their browsers to open the web-page (by addressing it with the provided IP and port) [19]. Please note that this feature does not work properly in the latest (25.3. 2015) version 2.5.1 [20], but it is fully functional in the version 2.4.6[21] (recommended for OS *Windows XP*, but runs properly on OS *Windows 10*).

An obvious weak point of this solution is the need for verbal or written announcement of data required for connection. Users are also forced to type this data into the address bars of their browsers. Sometimes the player interface web-pages have to be refreshed manually. These issues make usage of the system inconvenient. Furthermore, different browsers and devices can display web-pages differently, so non-equal conditions for players can occur. Using a browser can be also convenient for people who use the Internet for searching answers instead of fair play. Other drawbacks of this system include the platform dependency (*SImulâtor* is designed to run only on OS *Windows*[18]) and lack of option of language of interface (the only language is Russian)[21]. Also, a user-friendly tool for establishing the required network is missing in this solution.

However, the application handles applying competition and score managing well and can be successfully used in place of analog systems despite the mentioned drawbacks and some further minor problems. This solution allows to use computers and mobile devices connected to a wireless network, it solves the major weaknesses of analog systems. It also has some attractive features like multimedia content of questions, artificial intelligence that handles the moderator's job (works if answers are typed) and possible output of of the game information on a separate screen.

2.2.2.2 Other

A lot of implementations of quiz games exist in forms of software (like [22]) and web (like [23]) applications. They are not reviewed in this chapter because they are not direct alternatives of the analog system for *Game* and because of the lack of time. However, analysis of these competitors can bring valuable information for improving the concept and design of SwS or even the whole platform. This analysis can take place during further iterations of this project's lifecycle.

2.3 Analysis

Based on experience of state-of-art solutions and requirements from this thesis's task, the general concept of the structure of the system can be defined in the following way. SwS consists of two major components: the moderator component and the player component (which plays a role similar to buttons in an analog system). This division separates the requirements into three groups: to the whole system, to the moderator's component and to the player's component. Further specification of structure is presented in the Design section 2.4.1.

2.3.1 Functional and non-functional requirements

The following requirements are based on experience of state-of-art solutions and on requirements from the task of the thesis.

The reason for the requirements, aimed on increasing extensibility, is the variety of quiz games. It is obvious that using a large amount of applications for the same purpose is not convenient for a user. This means that the more games SwS is able to handle, the more the system is user-friendly. Because of this, it is important to design an easily-expandable SwS, in order to make adding new features and new supported games easier.

Flexibility of the game process also has vital importance because it is a feature that most analog systems cannot offer. And this feature allows to make the game process more adaptive which means more comfort for user. This is a field where a significant advantage against analog alternatives can be gained. A wide range of changeable preferences (see 2.3.1.2) makes the game process flexible.

Because of limits in time and other resources, the requirements are divided between Must-have and Nice-to-have. A system that responds to Must-have requirements is basically a wireless replacement for analog systems that handles score computing.

An implemented prototype of the SwS is supposed to meet Must-have requirements but it does not have to meet Nice-to-have requirements. However, the design of the SwS must let the complete implementation meet all the requirements (this complete implementation is not a part of this thesis).

2.3.1.1 Whole system

Functional requirements

Must-have

FS1 – Quiz game support
The SwS supports at least one variation of the game process (which is described in 2.1.1).

FS2 – Convenient network communication establishing
Being connected to the network, parts of the system establish communication between each other in a user-friendly way. The user is not required to provide information that can confuse him or her (IP-address, port,

etc.). If any information is required, it must be provided by a convenient user interface.

Nice-to-have

FS3 – Network establishing
The SwS contains a user-friendly program for establishing a wireless network needed for the game.

FS4 – Game Series
The SwS supports game series arrangement.

FS5 – Logging
The SwS is able to log the game (to record time of occurrence and other information about important events that happen during the game).

Non-functional requirements

Must-have

Composed
The System consists of a *Moderator* component and a *Player* component.

Nice-to-have

Multi-platform
The System works correctly if its components are running on different platforms.

Game capacity
The number of players who can take part in a game is greater or equal to 2 and less or equal to 10.

Game series capacity
The number of players who can take part in a game series must be greater or equal to 2 and at least less or equal to 30 (architecture of the system allows this number to increase in the future).

Extensibility
Architecture of the SwS allows to change the system and add new features relatively easily. In particular, this regards to future extending of the number of different quiz games, supported by the SwS.

2.3.1.2 *Moderator* component

Functional requirements

Must-have

FM1 – Game state information displaying
The *Moderator* component provides up-to-date key information about the game (the price of the current question, the answering player, the state of the game process).

FM2 – Applying handling
The *Moderator* component processes the player's applying for answer according to the algorithm defined in 2.1.1, the current state of the game process and settings of the game.

FM3 – Competition handling
The *Moderator* component distinguishes the first player who has applied for an answer after the question is read.

FM4 – Score handling
The *Moderator* component allows to change the score of the player who answered the current question, according to the algorithm defined in 2.1.1 and settings of the game.

Nice-to-have

FM5 – Answers displaying
The *Moderator* component displays the correct answer to the current question.

FM6 – False start handling
The *Moderator* component processes false starts (see 2.1.1) according to the option chosen in preferences. A False start is applying before a question is completely read.

FM7 – Setting up the game
The *Moderator* component allows to set up preferences that vary game process.

FM8 – Extended answers displaying
The *Moderator* component displays the alternative answer and notes for the current question.

FM9 – Open information about game preferences
The information about preferences of the game is viewable for players' components.

FM10 – Player managing
The *Moderator* component allows to remove players from the game and ban them (not let a certain player attend games handled with this moderator component).

FM11 – Players' information managing

The *Moderator* component allows to specify which information is displayed by players' components (except information that is necessary for the game)

FM12 – Series of games

The *Moderator* component is able to arrange several games into a series (tournament, championships, etc.). There are various methods for arranging these series.

FM13 – Defence against cheating

The *Moderator* component is able to detect at least some cheating attempts committed by users of the player components (collapsing the component, for example).

Non-functional requirements

Must-have

Runs on *Ubuntu 14.04*

The component runs on a computer under Operation system (OS) *Ubuntu*, version 14.04 or greater.

Nice-to-have

Various platforms

The component runs properly on computers controlled by *Linux*, *Windows* or *OS X* and on mobile devices under control of *Android* and *iOS*.

Multi-language interface

A choice of the language of the interface and game logs varies between English, Czech and Russian. The architecture of the component allows further languages to be added easily in the future.

Game settings

At least these following preferences of the game process are mutable with moderator's component:

- number of players,
- package of questions,
- variation of question choice (player can choose topic and price, player can choose only topic and price chosen sequentially/randomly, player can choose only price – topic chosen sequentially/randomly, player's choice is disabled – questions occur sequentially/randomly),

- variation in selecting the player who starts the game (randomly, by choice of moderator, by results of previous games – best player or worst player first),
- time limit for the game and for its particular phases,
- consequences of false starts (applying before the question is read) (let or do not let the applicant answer, do nothing, subtract points from the player's score),
- arranging method of game series (specifying number of games, sets of players, etc.)

2.3.1.3 *Player* component

Functional requirements

Must-have

FP1 – Applying
The *Player* component lets the player apply for answer (in the appropriate state of game process, defined in 2.1.1).

FP2 – Displaying information
The *Player* component lets the player get information that is necessary for the game (identity of the user of the component). All information is up-to-date.

Nice-to-have

FP3 – Extended displaying information
The *Player* component lets the player get information about the game that is allowed by moderator (identity of the currently answering player, text of the question, time limit for different parts of game process, table of questions if player's choice of questions is enabled, etc.). All information is up-to-date.

FP4 – Player identity
It is possible to use the same player component with a different identity.

FP5 – Achievements
Achievements and the user's personal statistics are available in the player component.

FP6 – Preferences
The *Player* component lets the player get information about preferences of the current game.

FP7 – Informer
The *Player* component warns the moderator about cheating attempts (for example, actions that can mean a possible usage of another application for getting information for answering)

Non-functional requirements

Must-have

Runs on *Ubuntu 14.04*
The component runs on a computer under Operation system (OS) *Ubuntu*, version 14.04 or greater.

Nice-to-have

Different platforms
The component runs properly on computers controlled by OS *Linux*, *Windows* or *OS X* and on mobile devices controlled of *Android* and *iOS*.

Multi-language interface
A choice of the language of the interface and game logs varies between English, Czech and Russian. The architecture of the component allows further languages to be added easily in the future.

Number of identities
It is possible to use the same component under at least three different identities.

2.3.2 Use case model

Due to lack of time, this model is not complete. Only the use cases that are based on the most important functional requirements are presented. The text representation of the model is presented on the enclosed media in the folder `/SwS/UCmodel/BP_vorobvla_SwS_UC_model.pdf`

2.4 Design

2.4.1 Structure

The state-of-art system described in 2.2.2.1 consists of one application, operated by a moderator. The player interface is implemented as a web-page generated by this application. To access it, players have to use third-party software (web-browsers). This approach has a lot of drawbacks (the greatest of them being the inconvenient connection process to the interface).

It is possible to eliminate a lot of weaknesses of this approach by implementing a player application that is able to make establishing communication with moderator easier. Also, each player using the same application makes conditions more equal (third-party applications can differ in speed and displaying of web-pages). Because of this, both the moderator and player components are implemented as standalone applications: the *Moderator* application for moderator users and the *Player* application for players.

2.4.2 Communication between components

Since it is decided that the components are standalone applications, communication between them must be designed.

A possible option could be peer-to-peer (P2P) communication, which is a network system where "elements ... both provide services to other elements and request services from other elements" [24][p. 3]. Such features are redundant for the solution of the current problem. *Players'* components are not supposed to exchange data or to share resources with each other.

Competition handling, common settings of game process and common input (questions) require game process to run in a solid environment. Running this environment on one element is more reliable and easy than on a distributed network of elements. This is why an optimal solution seems to be client-server architecture[25]. In this implementation the *Moderator* component must be a server that handles the greater part of functionality of the system. The *Player* component is a thin client whose only functionality is to display the player interface and to transfer information gained from this interface (i.e. player's applying), to the server. The reason for this solution is the need for rapid performance of the player application, and the possibility that its users would use cheap and not very responsive devices. Also, that will make the system easier to extend, because the changes of logic and interface that will be implemented on the side of server (*Moderator*), will not affect clients and will not require their changes unless communication protocol is modified.

Network communication between server every particular client is based on TCP protocol [26] and TCP/IP technology[27]. This choice is made because TCP delivers data "reliably and in order at the destination"[26][p. 9], and it is advised for case "if you need a reliable stream delivery service"[27][p. 25]. Indeed, transfer of applying must be reliable and ordered if competition of these applying is supposed to be handled properly.

2.4.2.1 Communication protocol

Establishing network There are existing tools for establishing network (tools of opperating system, Connectify.me [28], etc.) It is possible to create specific scripts for specific platforms.

Establishing connection In classical client-server scheme, connection is initiated be client (cite trusty source) (*Player* application in this case). However, establishing of TCP connection stars with client sending certain data to server. For this client must “know” IP address and port number of server. In *SImulâtor* this problem was solved by announcement of this information to the users. This application must avoid such situation and must be able to establish connection with minimal participation of user in this process.

It is possible to pre-define IP address and port of moderator and set up this IP on the interface of device, which runs *Moderator* application. However, this would make setting up of network more difficult for users or require an advanced tool for establishing the network to be a part of the system. This solution might be implemented in the future, when the author will design a tool for establishing the proper network, which will run on all desired platforms (including mobile devices).

An easier way (also described on Figure 2.2) is to make *Moderator* application to broadcast the data, required for connection, to the players. Broadcast can be performed with UDP protocol. Since, UDP does not guarantee the proper delivery of data, connection information must be sent several times. The system must allow to commit broadcast on the user’s demand for the cases when new players connect to the system. A weak point of this decision is dependency on network topology, because, according to [29] “routers by default do not forward broadcast packets”. That is why a useful feature can be an option to set IP and port of moderator manually for cases when the game is held over a network with complex topology.

Player identity While initialising TCP connection, *Player* application sends username of the player that will be used in application for identifying the player. It is obvious that this username must be different from the names of the other players, connected to the system. Possible solution to avoid username collisions is implementing the following algorithm. After obtaining username, *Moderator* application looks for its collisions. If it founds any, it sends a message with text string `uname-num` (where `uname` is the obtained username and `num` is number of occurrences of this username, increased by one) to *Player* application and this text string is interpreted as new username for this player. Otherwise, it sends back the username without changes. The *Player* application obtains message with (new) username and changes the original username to the obtained one (so, if no collisions occurred, username is changed to identical).

Since this algorithm is rather trivial and originality of usernames does not belong to requirements, control of username collisions is not mentioned in design and implementation of SwS and protocol. This feature can be implemented in further improvements.

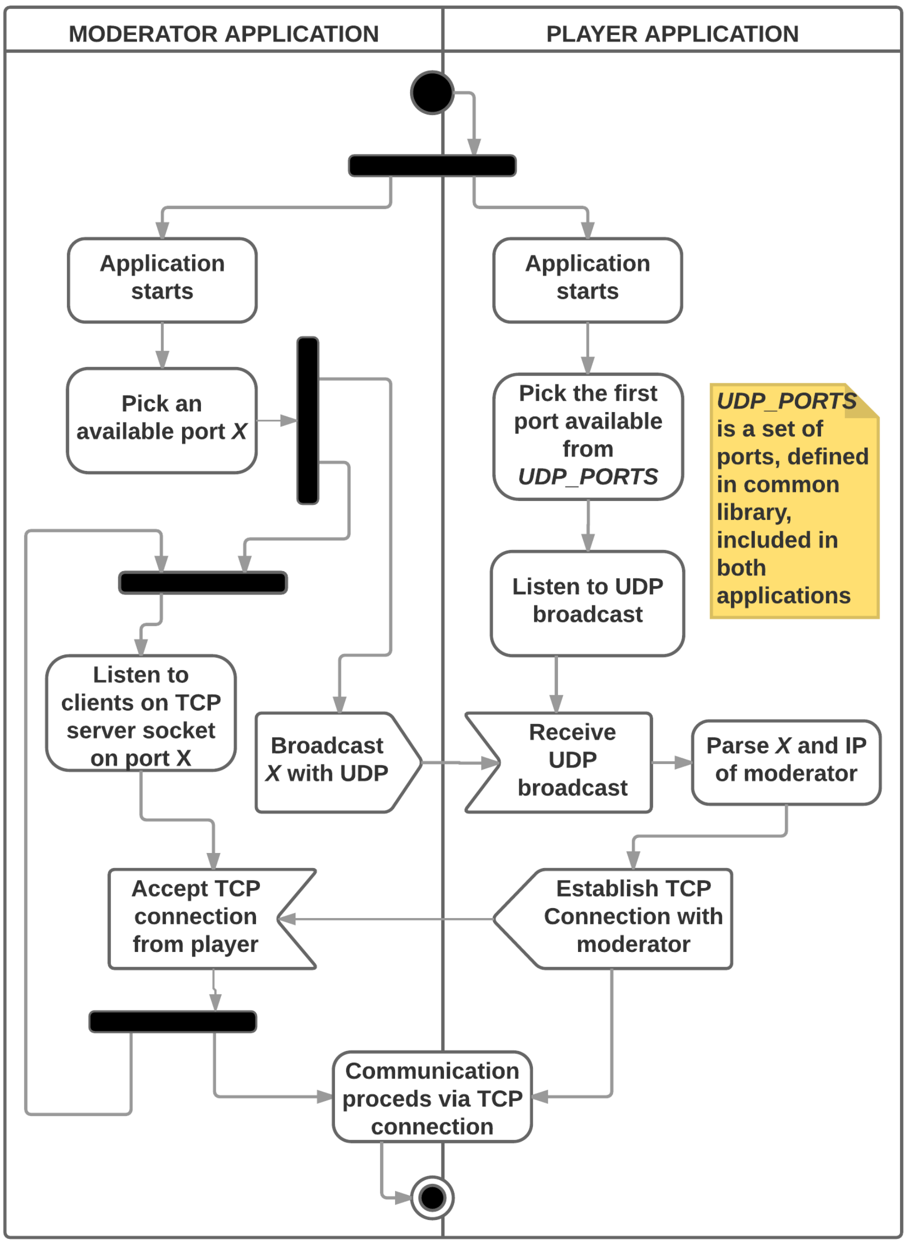

MODERATOR APPLICATION
PLAYER APPLICATION
Application starts
Application starts
Pick an available port *X*
Pick the first port available from *UDP_PORTS*
UDP_PORTS is a set of ports, defined in common library, included in both applications
Listen to UDP broadcast
Listen to clients on TCP server socket on port X
Broadcast *X* with UDP
Receive UDP broadcast
Parse *X* and IP of moderator
Accept TCP connection from player
Establish TCP Connection with moderator
Communication proceds via TCP connection

Figure 2.2: Establishing connection between *Moderator* and *Player* application

Time and competition Since players compete for time, the fair estimating of the moment of applying for answer has vital importance.

The easiest way to handle time competition is to save timestamps of player's applications and process them by *Moderator* application. These timestamps must be made by the device, where *Moderator* application runs to prevent possible players' cheating (they can set system time of their devices on smaller value than the real time). Drawback of this approach is time of processing player's application by his or her device and delay of network communication between *Player* and *Moderator* applications. It is clear that this period of time can be different and dependent on state and quality of network and player's device. However, human reaction time varies between 150 and 300 milliseconds [30]. An analysis of several wireless local area network (WLAN) standards shows that a mean delay is between 5 and 170 milliseconds [31]. That means that in most cases delay will not affect competition critically, because it is much shorter, than the time of human player's reaction. Because of this, the delay is ignored in implementation of the current project.

During further development, this problem can be partly solved in the following way. First, time between *Moderator* application and *Player* application is synchronized (by using protocol NTP [32] or SNTP [33], for example). Then, when player uses *Player* application to apply for answer the application saves timestamp of this applying and sends this timestamp to *Moderator* application together with applying. These applyings are stored in *Moderator* application and once per 100 ms they are sorted by timestamp and processed properly in their correct order. To make this solution implementable, the in-game communication protocol must transfer data that contains timestamp of applying together with data that represent this applying.

In-game communication In-game communication consists of sending information about game and player's user interface (UI) form *Moderator* server to *Player* and sending player's applying in the opposite direction. Player interface must be produced by *Moderator* application, *Player* application just receives this interface and sends back data that is required to detect user's interaction with this interface. That makes the system more extensible and modifiable (changes of player interface affect only moderator application, player application is changed only with protocol modification).

Information is exchanged as text messages in JSON format [34]. Message is represented as a pair of string with *Message key* (identifier of this type of message) and of a *Data* object with data. This data is represented as pairs of string tags that identify the data, and data itself, represented as a text string. Further and improved version of this protocol can be based on HTTP protocol using messages in further format. URI of the message is *Message key*, message body contains *Data* object. Switching to HTTP will be required if networking is implemented with with usage of a framework for Web applications. This solution can be useful because these frameworks can provide a more stable and effective solution. In this thesis the simpler version which is described in the beginning of this paragraph is used.

Game information is sent in the following format:

```
{"INFO": {"HTML": "<Html Representation>"}}
```

where `INFO` is the key that marks message with game information, `HTML` is the key for marking HTML representation of game information and player interface and `<Html Representation>` is HTML representation of game information and player interface itself. They are represented in single HTML-page, because keeping them together can be useful during their modification and representation. Elements of UI that are available for interaction with player are identified with specific keys. In HTML representation, these keys are stored in the `id` attribute of the element, identified with this key.

User's interactions with UI are sent in following format:

```
{"ACT": {
  "ELEM_ID": "<Action Element ID>",
  "TIMEST": "<Action timestamp>"}
}
```

where `ACT` is the key for marking this type of message, `<Action Element ID>` is `id` attribute of the HTML element, user interacted with (marked with `ELEM_ID`), `<Action timestamp>` is timestamp of this interaction (marked with `TIMEST`)

In case *Palyer* application detects activity, which can be a reason to suspect the user in cheating, it sends a message with following format:

```
{"CHEAT":
  {"INFO": "<Suspicious activity description>"},
  {"TIMEST": "<Suspicious activity timestamp>"}
}
```

where key `CHEAT` marks this type of message, `<Suspicious activity description>` is information about the suspicious activity (collapsing of application, for example; marked with `INFO`) and `<Suspicious activity timestamp>` is timestamp of the moment, when the action took place (marked with `TIMEST`).

2.4.3 *Moderator* application

2.4.3.1 Structure

Application structure is modular. The program consists of four modules that implement particular functionality: Model, GUI, Networking and Persistence.

2.4.3.2 Model

This module simulates game process itself. Game process, described in 2.1.1, can be simulated by a finite automaton (FA), with states that correspond to steps of this algorithm. This simulation also includes two more entities: *player* and *timer*.

Player is virtual representation of a player, who takes part in the game. It can be in one of four states. This state defines how the game reacts to actions committed by this player. The states are following:

Passive Any action taken by player in this state is ignored.

Choosing question This player can choose a question, all it's other actions are ignored.

False active If player in this state applies for answer, a sanction, defined in settings, is applied to it (by default – score penalty and *Passive* state until next question). All other actions are ignored.

Active If player applies for answer, it gains the right to answer. Other actions are ignored.

Also, player has a counter that contains the number of its successful applyings (i.e. those, which gave this player right to answer).

Timer is the entity used by the author for describing an activity that happens by timeout, i.e. after a certain period of time passed from a certain moment. To start a timer means defining the moment, from which this period of time is counted. To do something on timer timeout means to make this something happen immediately after this period ends. To unset *Timer* means to cancel execution of the activity.

Functionality that implements steps of algorithm 2.1.1, is placed into the particular states of FA (represented as UML state diagram is on figure 2.3.) that simulates game process. These states are:

Start Game Game is initialised. All players are set to *Passive* state. *Game* timer starts.

Getting question Responds to step 1 of algorithm. If a new question cannot be chosen or obtained by defined order, *Game* transits to state *Finish Game*. If a question is supposed to be chosen, the state of the player, who chooses question is set to *Choosing question* on entering in this state. New question is chosen or obtained by defined order (depends on settings).

Reading question Responds to step 2 of algorithm. On entering to this state, the state of every player is set to *Falseactive*. Application displays question to moderator (optional).

Waiting for answer Responds to step 3 of algorithm. On entering to this state, the state of every player is set to *Active* and *Question* timer starts, if not started. Application processes applying from players. After the first valid applying [1] is obtained, all players' state is set to *Passive*, the applied player is set as answering player and its counter of attempts increments.

Player is answering Responds to step 4 of algorithm. *Answering* timer starts. The applied player is marked (saved in *Game* entity) as *answering* player. On *Answering* timer timeout or if moderator skips this state, game transits to *Processing answer.* If the transit committed by skip, timer is unset.

Processing answer Responds to step 4 of algorithm. Moderator accepts or denies answer. If answer is accepted, the *answering* player's score is incremented by the price of the current question, *answering* player is reset, *Question* timer is unset and *Game* transits to *Getting question* state. If answer is denied, the *answering* player's score is decremented by the price of the current question, *answering* player is reset and *Game* transits to *Waiting for answer* state.

Finish Game Total score of players is updated or saved by Persistence module, log files and other external resources are closed.

Functionality of this module is based on implementation of this automaton.

For implementation of FA a following way, which uses benefits of Object-Oriented Programming (OOP), is chosen. Automaton is represented by class `Game`(implemented as singleton to make this class easier accessible from other, mainly Networking, modules), its states are implemented as subclasses of abstract class `State`. This abstract base class declares the following methods:

`setup()` in subclass implementations contains functionality that is supposed to be performed on enter into this state (setting up timers, changing state of players).

`perform()` in subclass implementations performs functionality of the system in the defined state;

`terminate()` in subclass implementations finishes the performance started previously correctly; can be useful if parallel programming is used.

`Game` contains attribute `currentState`, containing object that expands `State` class and represent the current state of the system. Also, `Game` contains `goTo(State)` method, implementing transition to the next state. This

[1]applying from player, which is in *Active* state and which's counter is less than 1 (this number can be customized in settings)

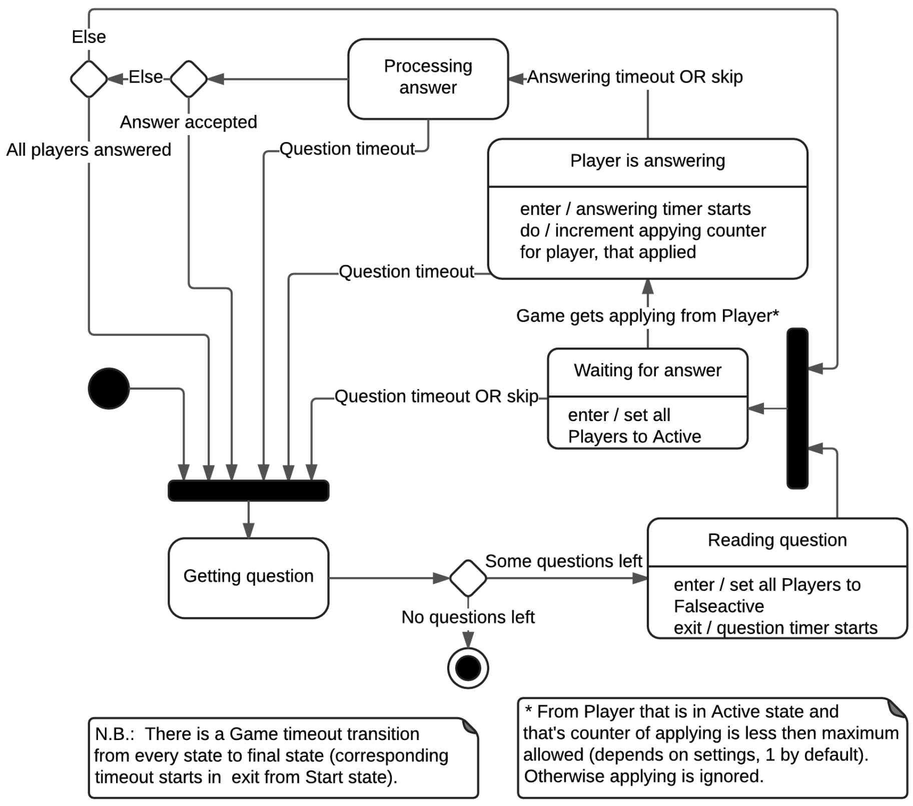

Else
Else
Processing answer
Answering timeout OR skip
Answer accepted
All players answered
Question timeout
Player is answering
enter / answering timer starts
do / increment appying counter for player, that applied
Question timeout
Game gets applying from Player*
Waiting for answer
enter / set all Players to Active
Question timeout OR skip
Getting question
Some questions left
Reading question
enter / set all Players to Falseactive
exit / question timer starts
No questions left
N.B.: There is a Game timeout transition from every state to final state (corresponding timeout starts in exit from Start state).
* From Player that is in Active state and that's counter of applying is less then maximum allowed (depends on settings, 1 by default). Otherwise applying is ignored.

Figure 2.3: Game process

method calls changes the `currentState` to next one and calls `setup()` and `terminate()` in proper order:

```
// Java-inspired pseudolanguage
  Singleton Class Game{
    State currentState;

    ...

    goTo(State nextState, bool terminate){
      if (terminate) { currentState.terminate();}
      nextState.setup();
      currentState = nextState;
      currentState.perform();
    }
  }
```

In this method, execution of `currentState.terminate()()` is optional, because some timers need to run during several states.

This module also contains classes that implement logging, question arranging, competition handling, settings, game information retrieving and user interface handling. The last one is remarkable, because it is used for communication with networking module and extends the abstract class `AbstractNetworkUI` from that module. The author took this design decision in order to keep all game logic in one module. This architecture seems to be useful for extending of this application in the future.

2.4.3.3 Networking

This module implements functionality needed for networking communication, like tools for establishing connection with *Player* applications, see 2.2 and class `PlayerPeer` that represents *Player* client in the context of networking. This class is able to handle network communication in the way described in the communication protocol 2.4.2.1. It parses incoming data that describes *Player* application user activity and uses an implementation of `AbstractNetworkUI` to transfer this activity to model. It also sends the UI in HTML format to *Player* application (for retrieving this data an implementation of `AbstractNetworkUI` is also used).

This module contains abstract class `AbstractNetworkUI`, which's implementation is in Model module. Implementations of this class in different models (in future extensions number of supported games is supposed to increase, which means adding new models) must implement UI in this particular model. `AbstractNetworkUI` defines two methods:

`String getUI()` generates User Interface as text string in HTML language format. This user interface contains control elements that allow user to interact with model. Every element are associated with an identifier of the model activity that is bound to this element. Design of an element allows the *Player* application to obtain this identifier when user interacts with this element (for example, by executing a simple JavaScript program that crates an alert with the identifier; this particular solution can change depending on choices, taken during implementation).

`processActon(String actId, long actTs)` performs Model action that is bound to HTML element with `id` attribute that is equal to `actId`. Attribute `actTs` contains timestamp of action. It is not used in this implementation, but can be used in further improvements with advanced competition handling.

An example of subclass implementation is following:

```
// Java-inspired pseudolanguage
```

```
class GameUI extends AbstractNetworkUI(){
  // key for marking UI control element 1
  private final String KEY1 = "key1";
  // key for marking UI control element 2
  private final String KEY2 = "key2";

  ...

  public String getUI(){
    // generates and returns HTML with game information and controlls.
    // the contolls' representation in HTML is something like
    // "<button onclick="alert('key1')">UI control element 1</button>"
    // "<button onclick="alert('key2')">UI control element 2</button>"
  }

  public void processActon(String actId, long actTs, Player player){
    switch(actId){
      case KEY1:
      // player calls model functionality
      // associated with UI control element 1
      break;
      case KEY2:
      // player calls model functionality
      // associated with UI control element 2
      break;
    }
  }
}
```

2.4.3.4 GUI

This model represents Graphical User Interface (GUI) for user of *Moderator* application. Contains views and controllers that allow local user (moderator) to interact with model.

2.4.3.5 Persistence

This model contains tools for work with database and file system (for storing questions, networking information and information, needed for series of games and users statistics). Implementation depends on the solutions chosen for realisation.

2.4.4 *Player* application

Although this application must not implement much logic, it also has modular structure, consisting of Networking, GUI and Persistence modules.

Networking model handles network communication under the designed protocol. It includes listening to broadcast of TCP connection information, establishing TCP connection with *Moderator* application and handling in-game communication, getting HTML representation of UI and sending proper data to server.

GUI handles representation of the obtained UI in HTML language. It is also able to detect the element user interacts with and get `id` attribute of this element in order to send it to Network module.

Persistence handles saving player and networking information.

2.5 Realisation

2.5.1 Technology

2.5.1.1 Applications

For implementation of both applications Java technology is chosen, because it is object-oriented, platform independent [35] and because Java is a very popular programming language (the most popular in the year 2015, according to [36]). It will make it easier for developers and users to mantain this project in the future, in case of its extension.

Platform independence of Java is useful for this project, because the components of SwS are supposed to run on different platforms (see non-functional requirements 2.3.1.1, 2.3.1.2 and 2.3.1.3). The chosen technology allows to adopt applications for different platforms with minor changes or without any.

Popular and wide-spread technology means that the project will be easy to maintain and support for developers and user community (in case the project will be open-source).

Furthermore, object-oriented language fits the modular structure of applications nicely.

Finally, Java is convenient and familiar to the author of thesis.

Communication Major feature of SwS is wireless communication between components that also has to be convenient for users. That is why choice of optimal technology for implementing communication is essential. In this section several approaches are reviewed and the best one of them is chosen.

Internet Components connect to server. Then the server processes data received from the components and sends results back or just transfers the data between the components. This solution requires Internet connection during the game and a permanently running server.

Local network based on Bluetooth technology According to [37], Bluetooth is a "wireless alternative to data cables by exchanging data using radio transmissions". This means that Bluetooth technology can be used for establishing a network that fulfills the needs of the system for a quiz game. This technology is also "low-energy" [37] that is a very positive feature for mobile devices. However this solution has some problems. Firstly this technology is not very familiar to ordinary user, so it will be probably difficult for him or her to establish the required network in case the application is not be able to do it for some reason. Furthermore, this technology appears to be rather complicated from the author's point of view, although some free of charge developer tools (for example, a Java library BlueCove [38]) for it exist.

Local network based on Wi-Fi technology The alternative is a well-known Wi-Fi technology. This technology is widely used for wireless networking and it is familiar and friendly to more users, than Bluetooth. Also, it is easier to find operating system tools (or other software) more managing local networks, based on Wi-Fi technology. Finally, the system will be able to use an existing Wi-Fi network if all the components of the system are connected to it (with some restrictions created by network topology, however).

Versioning The author performed versioning only on the project of *Moderator* prototype. The source code of *Player* is not versioned due to its relative simplicity.

For versioning Git VCS was used. In the beginning of work on the applications, the author used built-in interface if IDE to manage Git. This was a mistake that led to a significant loss in the future (an important part of the source code was lost and had to be written again). Also, that led to an unclear structure of the repository. That taught the author not to use VCS with an interface that is not clear enough (and to prefer command-line interface).

2.5.2 Implementation

An alpha-version of SwS prototype was implemented before the analysis and design were completed. This version was created mostly for testing the chosen networking solutions (this testing showed that the chosen solutions are functional and implementable). Because of that, this implementation contains some different solutions from the chosen in design. However, this implementation of SwS fulfils all must-have requirements and runs in a wireless network, created by devices under control of OS *Windows 10* and OS *Ubuntu 14.04* (testing was made on virtual machines, that used external USB Wi-Fi adapters).

Differences to design Some of the solutions, chosen during implementation of alpha-version, appeared to be less successful during more proper design, described in 2.4.

For instance, FA that is modelling game process in the moderator side (design described in 2.4.3.2), is implemented in another way. This solution is inspired by hard-coded implementation (presented in [39]), that was changed in order to make code more compact and, therefore, readable. Like in design, automaton is implemented as class `Game` in static attribute of `Game`, However, the states are implemented as methods of this class. Method `Game.proceed()` calls implementation of current state (with a `switch` statement). Every implementation of particular state change current state to the next one before calling `Game.proceed()`. This implementation seems readable enough, but it it still implies a lot of code in class `Game` that is not very handy and object-oriented. In general, this class is too solid and needs to be more composed and features of OOP must be used for increasing of variability of it's components that will lead to increasing variability of game process (example: polymorphysm – different style of question arranging. . .).

Other remarkable poor solution is an attempt to make an independent in-game UI for Player with broadcasting of game information form moderator in the form of JSON messages. After it is decided that another conception of in-game communication is more appropriate, it appears that changes of existing prototype are rather costly. These changes imply design and implementation of new in-game protocol based on transmitting of UI in HTML format.

Competition is handled in simplified way. However, that does not matter so much because of small delays in WLANs[31].

Different modules are rather dependent on each other (Networking calls methods from Model while processing messages). That decreases modularity of application and must be changed.

2.5.3 Testing

2.5.3.1 Development testing and debug

During the implementation, the author tested and debugged the system by running it's parts on local host and on virtual machines. Virtual machines used external Wi-Fi USB adapters for network communication via WLAN (that used to be created as an Ad-hoc Wi-Fi network on the host computer). That allowed to debug the system on a real WLAN, not on a software simulation of a network that is implemented on virtual machine players.

With this method, the author verified that the prototype implements basic functionality and that field and user tasting can be performed on it.

2.5.3.2 Field Testing

Field testing took place on a meeting of Prague community of intellectual games on April 25, 2016. The testing itself included several games with real *Svoâk* questions from the database of questions [40]. Each game was conducted by different moderator in order to allow the the testers to try both *Player* and *Moderator* applications.

Three members of Prague community of intellectual games took part in this testing: Alexei Golovko, Anna Lebedeva and Yulia Sheveleva. They kindly provided their devices for the testing. The author sincerely thanks them for their time and patience. All testers verbally agreed with video and audio recording of the testing before this recording started.

The testing was performed on three PCs under control of OS *Ubuntu 14.04*, OS *Windows 7* and *Mac OS X*. An ad-hoc WLAN created on the *Ubuntu* device was used for testing. It was planned to record it form the web-cameras of these devices and from a stand-alone camera, that filmed the whole testing area. However, due to stressing networking and applications set-up the author did not start the recording properly. That is why only one recording left as the evidence of this testing. Also, the author wrote down short notes with the most remarkable moments of the testing.

Testing was performed in Russian language because of empathy to the testers and simplicity of finding game questions in Russian.

This testing gained a lot of player's constructive criticism and allowed to find some bugs. In general, the users were not satisfied with the system, but they were able to conduct several games with it. "At least it works" was the final verdict of one of the testers.

Outputs Here is a review of the most important outputs:

- Network errors and exceptions are logged, but they are not presented in the GUI (fixed after testing).
- The *Player* application does not display any information about the game. It would be better for player to get information about the game form the state of the game.
- For some players, it would be more comfortable to apply for an answer with a key (for example, space bar), not with a button in the application.
- The *Moderator* application does not recognize Unicode characters on the name of player right if the *Player* application runs on *Windows 7*.
- The *Moderator* application game interface is "awful", "too technical" and "too close to backend". It must be more clear, abstract, intuitive and bright.

- It would be good to add a sound signal after transition form reading question to waiting for an answer.
- Tips are unclear.
- The location of control panel is wrong. The user does not notice it.
- One (*Proceed*) button for all states is confusing. There must be several buttons with clear functions instead.
- Name of states confuse the user. For example, a moderator has to start game in the main menu to create game with the connected players. Then the moderator has to start a game again of game interface in order to to launch a game process.
- The idle state of choosing question confuses the moderator. He/she suggests that it is the reading state that leads to fatal consequences (moderator launches the reading state instead of waiting for answer and the players apply for answer with false starts).
- Switching between *Moderator* application and external file is very inconvenient. Questions (their text and correct answers) must be displayed in the application.
- Significant part of users don't have JRE on their devices.
- Some WLANs do not allow broadcasting because of settings of network equipment. There must be a tool to establish connection by IP directly.
- *Moderator's* interface blocking while wating for answer is confusing and useless (solved)

2.5.3.3 User Testing

User testing was conducted by user scenarios that cover Must-have and some Nice-to-have requirements. (FS1, FS2, FM2, FM3, FM4, FP1, FP2). The scenarios are written in Czech and Russian.

The author sincerely thanks Jevgenij Cist'akov, Maiia Grauden, Mikhail Titenko, Petr Polezhaev and Yevgeniya Chekh for taking part in this testing.

Each testing was performed by one tester and one test moderator (which was the author of this thesis). Each of the participants used a PC controlled by *Ubuntu*. The computers were connected to ad-hoc WLAN, created on of the PCs. Moderator used his PC to run the parts of SwS in order to create the proper situation for each scenario. The tester's computer performed a recording with *Kazam Screencaster*. The tester agreed to the recording verbally before the recording started.

Each test was held in the native language of the tester in order to make this procedure more empathic to him/her and to create the conditions for the player to provide as much information as possible. Author took advice for writing scenarios and for conducting the testing from [41] (ch. 2, 3, 5, 6, 7, p. 52-56, 95-103, 191-195, 220-226, 247-293). Basing on this advise, the author tried to conduct the testing basing on following rules:

- being empathic,
- informing the testers about the purpose of the thesis and user testing particularly,
- requiring neutral and objective position from the testers and creating conditions for that,
- asking and provoking the testers to express their thoughts,
- avoiding to give tips to the testers (this rule was not followed often enough).

Before the start of each test (or during the test in some cases), the testers were informed about the goals of the thesis and of the testing. They also learned about the domain of the application and read the user manual.

After the first test it became clear that some of scenarios were too long and they were split into smaller ones in the future versions of scenarios. The recording with the first test was cut by the latest version of scenarios.

After the testing, every tester took part in a survey Each survey file is set to read-only mode by a script immediately after the tester closed it. This measure does not make it impossible to forge results of the survey, but it prevents them from accidental change.

Scenarios, scripts, survey forms and completed surveys can be found on the enclosed media (path to folder: `/SwS/prototype/UserTesting`).

It is clear from the recordings that the testing was not performed perfectly by the guidelines from [41]. Some tasks in scenarios were were not short and clear enough. Also, lack of the author's soft skills was a problem sometimes.

Output The testers rated different aspects of the prototype from 0 (this aspect is totally missing) to 10 (perfect). Here is the average testers' rating calculated from the data gained from the survey:

Aspect	**Average rating of the testers**
Convenience of setting up the network communication (on the devices that are connected to WLAN)	——
from player side	5.6 (acceptable)
from moderator side	7.8 (acceptable)
Supporting of game process (how much does the game process responds to the Game rules)	6.8 (acceptable)
Convenience of game process	——
from player side	4.4 (acceptable)
from moderator side	7.2 (acceptable)
Completeness of the provided game information	——
from player	3.4 (awful)
from moderator side	6.8 (acceptable)
Clearness of representing of game information	——
from player	4.6 (acceptable)
from moderator	7.0 (acceptable)

The average ratings show that most testers suppose that the prototype is acceptable in most of it's aspects. Similar thought was experessed by a lot of testers. However, one of them suggested that a game with analog system is more convenient then a game with the prototype.

During the testing, valuable information was gained from interviewing the testers and from observing the test. For example, it is clear that one button (*Proceed*) button, that handles proceeding to the next state needs to be split into smaller buttons. It is not only confusing, it can ruin game process if the user clicks the button twice and unexpectedly skips a state of game process.

Also, most testers noticed a lack of game information provided. The information about false starts is not displayed properly in the *Moderator* application. The *Player* application does not provide any information about the game, too. For some testers that was a fatal drawback, for others it was an inconvenience, that can be tolerated because of verbal communication between participants of the game.

The testers made a lot of suggestions concerning presenting the information about game.

2.5.3.4 Testing outputs

A lot of information about drawbacks and possible improvements was gained. Based on this information tasks for the future iteration of the project were made. These tasks are presented in a form of tickets:

T1 – Improve in-game GUI

Making GUI more convenient and suitable for players has a major importance for the project. Most of the testers' complyings were caused by the drawbacks of UI. This is a complex task that can be divided into several subtasks:

- Improve the player GUI:
  - Implement the game information display (at least the name of the current player and game score table). That is asked by every tester.
  - Block or hide the buttons whose functionality does not make sense in the current situation.
  - Implement notifications about the result of applying to answer.
  - Map the applying action on a key (spacebar is suggested).
- Improve the moderator GUI:
  - Sort the score table by score or make the score table sortable by user.
  - Mark positive and negative scores with colour.
  - Implement display of question text and topic.
  - Split the *(*Proceed) button to several specified control elements.
  - Add sound signals for transition from reading game to waiting for applying and for false starts.
  - Add colours into game log.
  - Make a clear notification about false start.
  - Improve language. Use more simple terms, shorter sentences and avoid confusing similarities (like *Start game* button for creating new game with with the connected players and *Start game* state, when the user launches the game process inside the created game).

It would be reasonable to implement these improvements after changing the model according to the design of network UI, described in this thesis 2.4.3.3. This new network UI that is supposed to be implemented with HTML and CSS technologies must will fulfil the features above.

T2 – Improve networking

Testers were more satisfied with networking, then with GUI. However, they had some criticism about it. The player application does not display much information about network communication establishing process. For example, a player, who is connected to moderator does not know about that. That must be improved. Also, there must be a way to setup communication directly by IP in case the network that does not support broadcast is used. Finally, some testers found the *Player* networking setup interface not perfect and suggested replacing a radiobutton with more a common usual button and a label with the state of connection. Also, the field with player name in this application must be cleared automatically on click, if it contains the default name.

T3 – Improve the game process
Some problems in the implementation of game process occurred. For example, states of game, that do not imply moderator's or players' activity confuse the users and can ruin the game process (in the prototype such states are *Coosing questions* and *Reading question*). Freesing moderator's functionality during player's answer was a poor idea too (this freesing was removed soon after testing).

2.6 Summary

Prototype handles supporting of the *Game* and can be used as a replacement for analogue systems. It responds to must-have requirements. However, it does not implement all the designed features.

Because of mistakes that were made during naive design of alpha-version, architecture of the application must be changed significantly. That shows the importance of analysis and design. Since a lot of changes have to be made and a lot of code have to be rewritten or deleted, more work on alpha-version would be useless. It is good that this version with less successful design is not developed more, than it needed to. If more resources were invested in version, their waste would be unreasonable and disappointing.

The current version of SwS cannot be called a better alternative to the existing solutions. However, it is able it establish communication between components with minimal user's efforts. It is a unique feature, that was not found during the review of state-of-the art solutions.

Work on the project of SwS in the future iterations must must proceed in the following major directions:

- Improving the existing prototype by strict implementing of the design and executing the tickets, formed from testing output (2.5.3.4). A new GIT repository with a clear structure will be created for versioning this source code. That also implies a stricter discipline of versioning.
- Implementing the functionality from Nice-to-have requirements.
- Developing SwS components for mobile platforms. Most people don't bring computers on intellectual games. It is obvious that for wider population of the SwS these platforms are essential.

- Attempting to create a new version of network protocol and a took for establishing a network needed for game. That protocol would simply reserve one IP address for the game moderator. In this case the tool would create a network with static addressing, where moderator's device has the IP-address, specified by moderator. These options must be better reviewed during the analysis in the next iteration.

CHAPTER 3

Information system

3.1 Introduction

Since the SwS is designed for games that take place in one physical location, community of its users (game community) needs tools for organising the games and sharing information about them. Furthermore, the games require questions that are supposed to be created, arranged and distributed in an appropriate way. Finally, the SwS components also need to be distributed to the users.

All that implies a lot of information to be handled by members of the game community, especially by the users who organize games. Information system (IS), which is "an integrated set of components for collecting, storing, and processing data and for providing information, knowledge, and digital products"[42], can be used for supporting game community by making the information easier and more convenient to work with and share,

3.2 Analysis

Since SwS and the whole platform are inspired by the community of *intellectual games*, author supposes studies of this community to be a valuable source of information that can be used for arranging life of the community of that platform users (which is the domain of this IS) and for formulating requirements to the IS. Since it is possible to review the domain as a variation of *intellectual games* community (that is more cosmopolitan and bound on this platform), in this analysis life of both of these communities is understood as the domain, if other meaning is not specified.

Author admits that researching only *intellectual games* is not the perfect decision. Information about any other possible competitors can also be useful. These competitors do not have to bear resemblance to *intellectual games*, experience of other communities with any similarities in the domain can be useful [41](p. 37-38). Such researches are not presented in this thesis due to lack of time. However, it is strongly advised to perform them during analysis that will take place in further iterations of lifecycle of this project.

3.2.1 Methodology and structure

This analysis contains research of domain, state-of-the-art solutions and requirements. A user research, based on information from interviews and additional surveys was planned, but it was not performed due to lack of time. This research (that includes creating user profiles and persons) can be useful during further iterations of this project. Further wants&needs studies are also advised for the future.

Qualitative information about the domain and state-of-the-art solutions is collected from a participant observation (author's experience of being a member of *intellectual games* community for several years) and from an interview with Lilia Grozovski, president of Verein Intellectus Helvetici (Bern, Switzerland), game organizer and moderator[2]. This interview was prepared and conducted according to tips from [41](ch. 7, p. 247-293, 311). A contaminated form of interview was chosen with prepared questions that covered necessary topics but without a strict following to the plan. The information gathered from the interview is used as an input for the research.

The structure of these research is based on the following aspects of the domain:

Game organising
Refers to organising issues, like defining place and time of the game and sharing this information with the members of the game community. Organisation also includes all other the supportive processes, aimed on making game happen.

Shared database of questions
Refers to storing game questions in a database ("collection of data organized especially for rapid search and retrieval"[43]). These questions are shared with the members of the game community.

Deploying software
Refers to deploying software that supports games to the users' devices.

Other
Refers to other processes that are not mentioned above and that take place in the life of the game community.

3.2.2 Domain and state-of-the-art solutions

Main purpose of this research is to provide holist view on the domain.

[2] Taken via Skype on April 23, 2016. The recording and its English translation can be found on the enclosed media in the folder `/IS/Interview`

3.2.2.1 Game organising

Most games are organized by local communities of *intellectual games*.

There are different kinds of large competitions that usually involve more than one local community. Some of these competitions take place in one location and they are usually rather difficult to arrange (because of a lot of problems like finding a place for the games, accommodation for players, transport, etc.). Therefore, this type of games is not very popular and can be reviewed more closely in the further iterations of the project.

This review is focused on the most popular type of competitions, which are distributed (or, in the terms of the community, *synchronious*): local communities play the same games with the same questions in the same period of time, so that all teams from all local games compete with each other. Such games are usually arranged by one local community that plays the role of a coordinator: prepares questions and handles players' appeals (in case game disputes occur). Other local communities sign up to such competitions and handle game organising in their locality including retrieving the area, informing possible players about the game, arranging the game itself, checking the answers and calculating the score, collecting the game fee (if set) and player's appeals and, finally, sending local game results and game appeals to the coordinators. Usually these competitions are based on playing the most popular intellectual game – the "off-line version" (or *sport version*, as it is called in the community) of *What? Where? When?*. This game is actually a quiz of several teams (each of them contain up to 6 persons) that have the same time for answering (a minute) and express their answers in written form. This game needs a moderator that reads questions and collects the answers. Sometimes moderator's attendants are present, who help collecting answers and distributing so-called *supportive material* (pictures or text that are attached to some questions).

The *intellectual games* movement is coordinated by a non-profit organization called *MAK ČGK* or just *MAK* (*Meždunarodnaâ associaciâ klubov "Čto? Gde? Kogda?"* [*International Association of Clubs of "What? Where? When?"*])[44]. This organization licenses the local communities and competitions. Information about these licensed entities is collected by *MAK* and stored in their database available on their rating web-site [45] (this licensing is reviewed in more details in [46]). This database represents life of this community and contains information about any player and any game that was held in it. Also, this organization provides its IS (rating web-site) to the licensed entities as a platform for organising synchronous competitions.

3.2.2.1.1 Organising process In general, a process of organising a competition via *MAK* rating site is following:

1. The coordinator decides to create a synchronous competition. It orders

questions from their authors or editors, specifies the members of Game Jury and Appeal Jury and the fee for participation.

2. The coordinator sends an application for licensing (or registering in other terms) via *MAK* rating site. This application contains information about the game including [46] [3]:

   - name of the competition,
   - dates and place of the competition,
   - members of the Juries,
   - organizers (the coordinator),
   - scheme of the competition (type of competition, game, number of rounds and questions, etc.),
   - supposed number of participants,
   - financial issues (sponsors, participation fee),
   - mass-media that are supposed to attend the competition.

   This application must be sent not later, than two weeks before the start of the competition. The application is created with and interface on the web-site of *MAK* and sent as an e-mail to the member of *MAK*, who is in charge of licensing competitions.

3. If *MAK* representative agrees to license the competition, it appears in the *MAK* rating site and *MAK* sends an email with granting the licensing to the e-mail of coordinator of the game.

4. Local organizers apply to participate and start their processes of local game organising. Each of these process is described like this:

   a) Local organiser makes an application to participate in the competition in the *MAK* rating site. In this application the organiser specifies information about the local game (moderator, organiser, number of teams that will take part in the competition).

   b) Local organiser informs their community about the upcoming game sharing information about name, time and date, place and the about the competition itself. For this various means are used: social networks, web-sites or blogs of local communities, e-mail or even messengers. The organiser have remember which part of his or her audience is reachable by each of these options.

[3]N.B.: the information about this step differs in various sources (in the interview and in the specification in the web-page). In cases if collisions the author trusts more to the information from the interview, because it is more up-to date.

   c) The organiser downloads questions from IS, prints them and the moderator conducts the game.

   d) The organiser checks the answers and collects the game result (score, lists of team members, papers with answers), participation fee, appeals to the questions and remarks about the questions.

   e) The organiser sends the game results to the coordinator. The results (lists of team members and teams' answers, appeals and remarks). Also, an anonymous appeal can be sent by any player who took part in this game via the *MAK* rating site.

5. The Game Jury resolves remarks about the questions. These remarks contain suggestions about the questions and correct answer expressed in non-structured form. After this resolving coordinator notifies every local organiser about results of the resolving. The local organisers have to recheck the answers given on their games respecting the decisions of Game Jury (for example, accepting an answer that was not acceptable before, but the Jury decided to accept it) and update the results.

6. Appeal Jury resolves appeals that are structured requests for discharging a question (for example, if it contains a fact mistake) or for accepting and alternative question. After that, local organisers are informed about the decisions of Appeal Jury. They recheck answers and update game results the way similar to the previous step.

7. Local organisers transfer the payment in various ways. It can be bank transactions or passing the cash via the members of community, who travel to the events (some other tournaments), where they can meet the representative of the coordinator. The payments are usually transfered once per season (a time period from September to May).

3.2.2.2 Shared database of questions

Usually the questions are obtained on *Question database of What? Where? When?* [40]. Web-site of the database supports obtaining a random package of questions and a search of a package with specified parameters. It contains questions for *What? Where? When?*, *Their own game* and other *intellectual games*. The available questions are free of charge, but are distributed with obligatory conditions of reference to the database and non-commercial use [47]. These questions are often used by local communities for training. If more rare package is needed (for arranging a competition for example), the representatives of the database can help finding it. They can offer paid or free of charge (that are usually older) packages. This reference is performed via e-mail communication with the *Question database* team.

The web-site of the *Question database* allows to brows the questions on the web-page, without CSS (that is a better representation for printing), in fb2 or XML format (another XML-based format that is simpler, then fb2).

The exclusive packages can be written in the local game communities or ordered from known authors and editors. The author did not found a centralized platform, where the editors, authors and their possible customers meet each other.

3.2.2.3 Deploying software

Special software is not used very widely in the community of *intellectual games*. The only SW for *Their own game* known to the author is the *Simuliator*. It is distributed from web-pages of it's author [18] and from Windows Store[48].

3.2.2.4 Other

Area For some game communities the area for games is a problem. It is easily solved if the community is created basing on some organisation that can provide an area (for example, a community of some educational institution). Other communities manage to rent an area or use the places like pubs or bars for their games. In the last case, it can be difficult to find a place with appropriate prices, but sometimes these institutions have a financial interest in attracting the community because of guaranteed obtaining of certain number of clients in the certain time. There is no known unified platform for communication between game communities and the institutions.

Sponsorship The sponsorship of game communities or teams can be presented as providing areas or money for covering expenses on transport (if the team travels to a non-distributed competition that takes place in another locality), operating materials, prises and so on. On exchange for obtaining sponsorship, the community thanks and propagates the sponsor (places logo on the uniform, expresses acknowledgements on their web-site, etc.)

Summary From the holist view, retrieved in this study, it is clear that the life of the community of *intellectual games* is supported by two IS that are rather well-functioning. They are *MAK* rating site (handles game arranging) and Database of questions. Both tools handle their job well, although they still can be improved. Some aspects of the life of game community such as informing local community about the game are not supported by any uniform tool that can lead to complications.

The strongest aspect of *MAK* rating site is the database of players, and the process of arranging competition. The only weakness of this process is the need for the organiser to recheck the game results twice. Actually, the reason of division of the Juries to Game and Appeal (that causes this duplication) is not clear to the author. Maybe they should be joined into one instance. Also, it might be a good idea to pass the function of checking and rechecking of answers to the coordinators of the game because checking answers and uploading results before the deadline (the results must be uploaded before the deadline, set by coordinators) can be difficult for the organisers. Another drawback of this system is learned from the interview. The system requires to upload the game information in CSV format with specified encoding. It can be inconvenient and tricky for the new users, but they get used to it quick however.

The database of questions is great for obtaining free but obsolete questions that can be used on trainings. However, the author did not manage to learn about a centralized tool for ordering exclusive questions. Maybe using decentralized conversation satisfies the needs of community, but a more effective tool for it might make competition arranging more easy and increase the number of games by it.

A uniform tool for informing game community about the upcoming games does not exist. This job is made by the local community organisers with various ways. It might be good to find a way to centralize it.

Game area is a problem for game communities. A tool that helps to solve this problem (by simplifying communication with area owners, for example) might be valuable.

To sum up, the life of game community implies usage of a variety of organising tools. That means that a unified information system of a correct design and implementation can improve the life of this community, making it more convenient and better organized.

3.2.3 Requirements

It is possible to form functional and non-functional requirements basing on the research presented above.

3.2.3.1 Functional

Functional requirements are arrange into the same categories as domain review. Non-functional requirements are not separated to different groups, because they are common for all the IS.

Game organising

FG1 – Competition organising support
The IS supports competition organising process simplified variation of the organising process used by *MAK* (see 3.2.2.1.1). The simplicity is in using a single instance for resolving the appeals. Also, the organiser of a game that takes place in the frames of the competition do not collect

and send appeals in this process because any participant can make an anonymous appeal (see FG3).

FG2 – Game ratings
The IS contains a database of the players' game ratings (basing on the result of games played by the player).

FG3 – Competition feedback
The IS allows any participant of the game to make an anonymous appeal if he or she supposes that their answer has to be accepted or a question has to be discharged. Also, IS contains a tool for open discussion of the game.

FG4 – Game information sharing
The IS allows to share the information about the upcoming local game to the members of local game community. The information must be sent to various channels (e-mail, social networks, maybe Skype and messages).

FG5 – Game fee collecting
There must be a convenient way to collect game fee from local participants and transfer it to the game coordinators.

Database of questions

FQ1 – Free of charge obsolete questions database
The IS supports search in the database of obsolete questions. The use of these questions is free of charge. Only non-commercial use of these questions is allowed.

FQ2 – Paid and free of charge new questions store
The IS supports options for the authors of packages to sell new packages of questions and options for game coordinators to order or buy the packages. Also, authors and editors must be able to present free of charge questions. After being played once, the questions are available in the database of obsolete questions.

FQ3 – Question and author rating
The IS lets the users to rate questions. The ability to rate a question must be grant only to the players who played this question. The rating of an author of package is counted by the IS basing on the rating of questions written or edited by this person.

Deploying software

FSW1 – SwS components deploying
The IS allows the user to deploy SwS component on his or her device.

FSW2 – SwS components' extensions store
: The IS allows the user to deploy SwS component extensions.

FSW3 – Software feedback
: The IS allows the user to leave feedback about SwS component or their extensions.

Other

F1 – Game area finder
: The IS allows the communities to find a place for games and it allows the area owners to find new customers.

F2 – Local community communicating
: The IS contains a convenient tool for communication inside the community.

3.2.3.2 Non-Functional

Accessible via web
: The IS is available via WWW with current versions of most popular desktop mobile browsers (*Microsoft Edge*, *Mozilla Firefox*, *Google Chrome*, *Opera*, *Safari*).

Multi-language
: A choice of the language of the interface and game logs varies between English, Czech and Russian. The architecture of the component allows further languages to be added easily in the future. Default language is English.

3.2.4 Use cases

Due to lack of time, Use Case is created only in UML language without detailed scenarios (see Figure 3.1). The model is split into the same groups as the functional requirements: game organising, database of questions and deploying software. If any use case is related to a use case from another group, both use cases are presented in the both groups. The use cases from another groups are marked with grey on the schemes. The Use Cases that cover the other requirements are not presented and they are supposed to be modelled during next iterations of the project.

Here is a brief specification of actors:

User a person who uses the IS.

Coordinator a user that arranges competitions (sets of similar games that take part in a specified time period and that have common game rating).

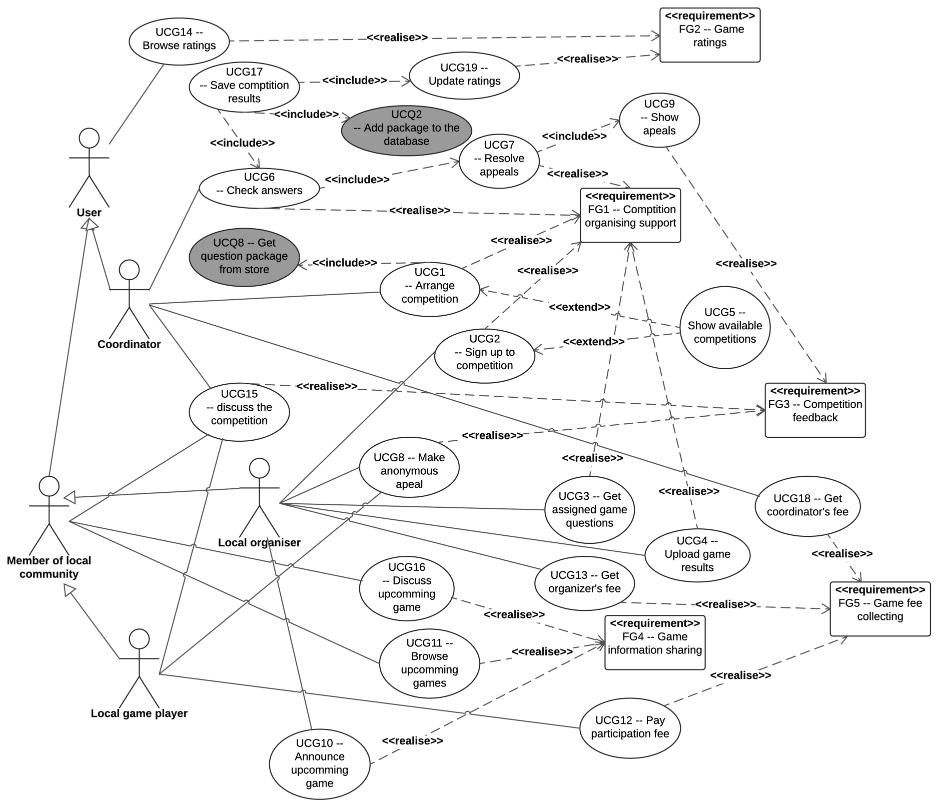
Member of local community a user that is associated to a certain local game community.

Local organiser a member of local community that can sign the community up to a competition and arrange the local game that is part of the competition.

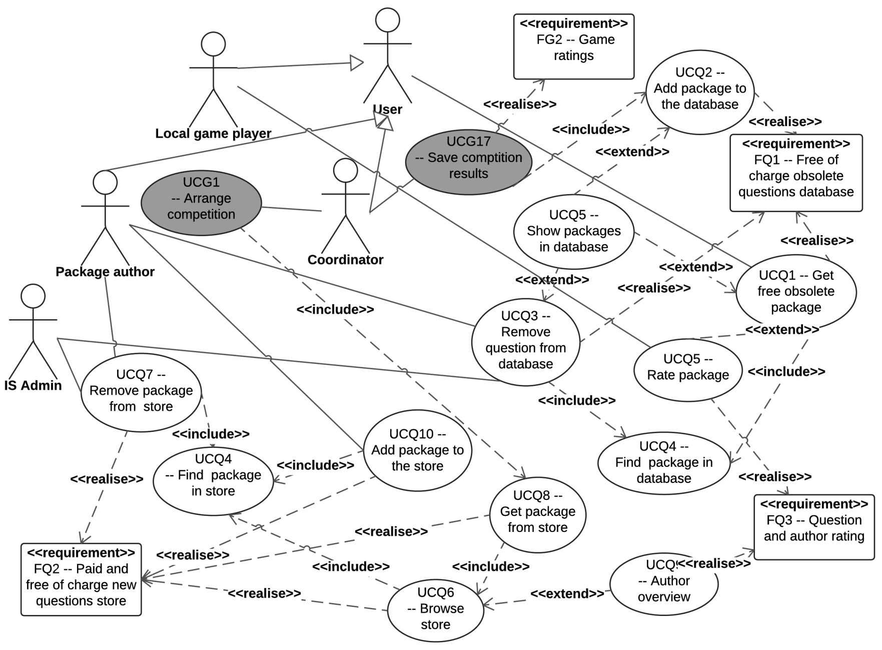
Local game player a member of local community that takes (took) place in the particular game.

Package author a user that creates new question packages.

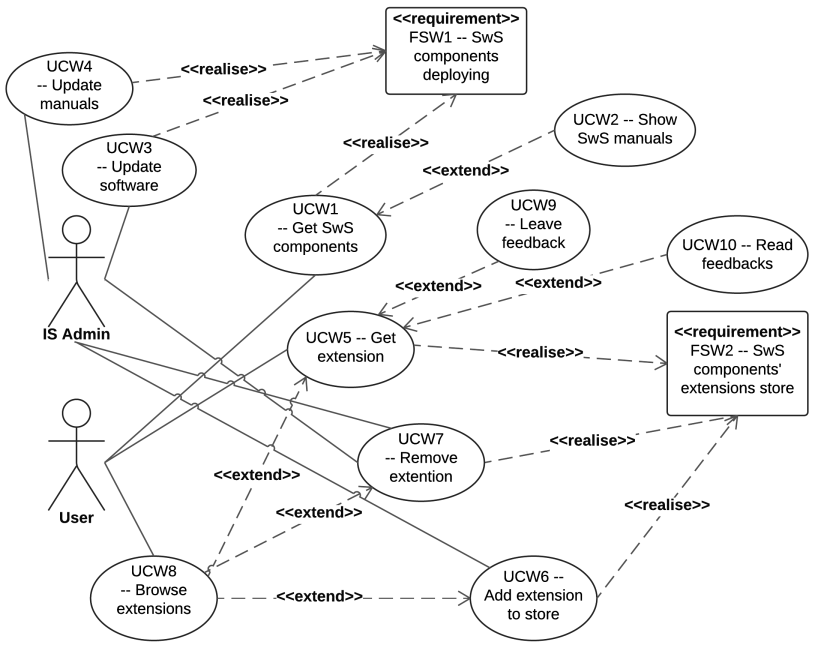
Admin administrator of the IS.


UCG14 -- Browse ratings
<<realise>>
<<requirement>> FG2 -- Game ratings
UCG17 -- Save comptition results
<<include>>
UCG19 -- Update ratings
<<realise>>
<<include>>
UCQ2 -- Add package to the database
UCG9 -- Show apeals
<<include>>
<<include>>
UCG7 -- Resolve appeals
User
UCG6 -- Check answers
<<include>>
<<realise>>
<<requirement>> FG1 -- Comptition organising support
<<realise>>
UCQ8 -- Get question package from store
<<include>>
<<realise>>
UCG1 -- Arrange competition
<<realise>>
<<realise>>
<<extend>>
UCG5 -- Show available competitions
Coordinator
UCG2 -- Sign up to competition
<<extend>>
UCG15 -- discuss the competition
<<realise>>
<<requirement>> FG3 -- Competition feedback
<<realise>>
UCG8 -- Make anonymous apeal
<<realise>>
UCG3 -- Get assigned game questions
<<realise>>
UCG18 -- Get coordinator's fee
Member of local community
Local organiser
UCG4 -- Upload game results
<<realise>>
UCG16 -- Discuss upcomming game
UCG13 -- Get organizer's fee
<<realise>>
<<requirement>> FG5 -- Game fee collecting
<<realise>>
<<requirement>> FG4 -- Game information sharing
UCG11 -- Browse upcomming games
<<realise>>
<<realise>>
Local game player
UCG12 -- Pay participation fee
UCG10 -- Announce upcomming game
<<realise>>

(a) Game organising


Local game player
User
<<requirement>> FG2 -- Game ratings
UCQ2 -- Add package to the database
<<realise>>
<<include>>
<<extend>>
<<requirement>> FQ1 -- Free of charge obsolete questions database
UCG17 -- Save comptition results
UCG1 -- Arrange competition
Package author
Coordinator
UCQ5 -- Show packages in database
<<extend>>
<<realise>>
UCQ1 -- Get free obsolete package
<<include>>
UCQ3 -- Remove question from database
UCQ5 -- Rate package
IS Admin
UCQ7 -- Remove package from store
UCQ10 -- Add package to the store
UCQ4 -- Find package in store
UCQ4 -- Find package in database
UCQ8 -- Get package from store
<<requirement>> FQ3 -- Question and author rating
<<requirement>> FQ2 -- Paid and free of charge new questions store
UCQ6 -- Browse store
UCQ -- Author overview

(b) Database of questions



(c) Deploying software

Figure 3.1: IS Use Case model

3.3 Design

There are prepared solutions that provide functionality covering the use cases. It would be reasonable to use them. However, all the solutions must be integrated and accessible with the interface that is single for the users in order to allow them settle all game issues in one place. This place is defined by the author as core IS.

For implementing core IS the Symfony PHP framework is suggested. This choice is made due to the its following features:

- popularity that implies best-practices, good documentation and wide community,
- development tools and ready solutions aimed on making development easier and faster (like ORM mapping handled by Doctrine, generic components handling functionality necessary for web-programming),
- authentication and authorization based on roles (implemented as ACL), that is useful for distinguishing of the functionality available to different actors,
- MVC architecture of the project that is convenient and known to the author from his experience with Java developing.

Also, one of the reason for this solution is the author's experience in using it during his studying on FIT (the subject BI-PWT).

The core IS consists of modules that implement game organising, shared database of questions, deploying software and further possible functionality. Modular structure implies better extensibility and easier modification of the core IS.

The design and suggested technologies of the most important modules (game organising, shared database of questions, deploying software) are reviewed further in this chapter.

3.3.1 Game organising

Game organising is supported by a custom-designed module because the author did not manage to find a proper ready solution for this problem. The simplified process is shown on the Figure 3.2.

For supporting this process, developing of a special web application is suggested. Conceptual model of this application in ontoUML language[49] is presented on the Figure 3.3.

Like the core system, this module is supposed to be implemented in Symfony.

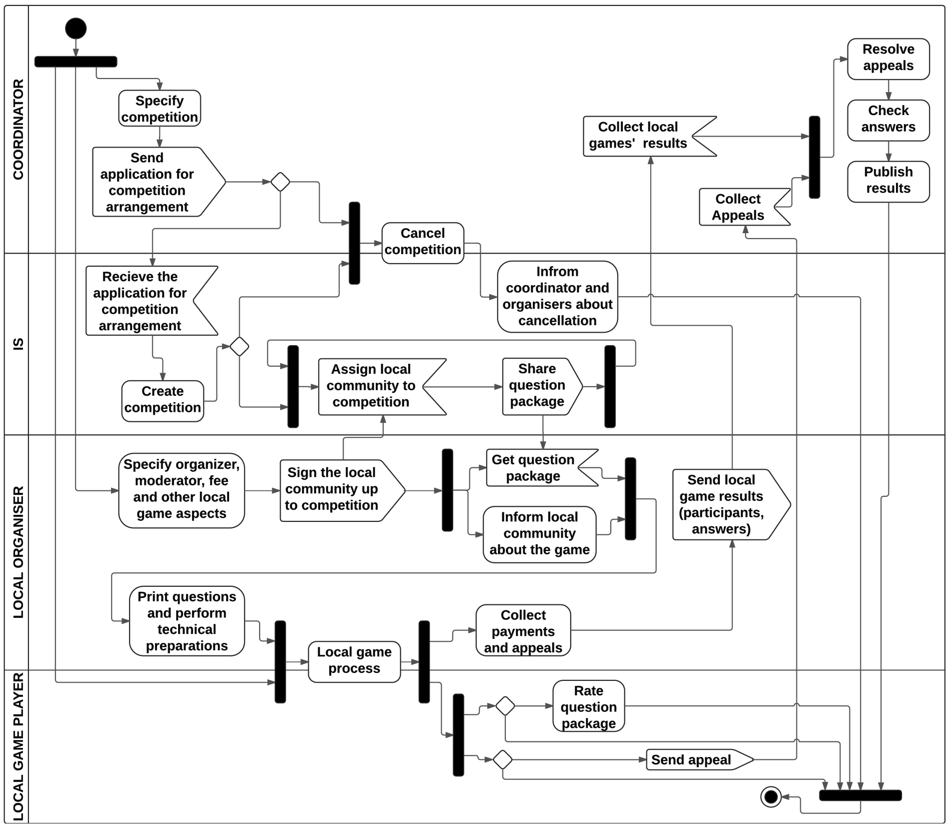

COORDINATOR
IS
LOCAL ORGANISER
LOCAL GAME PLAYER
Specify competition
Send application for competition arrangement
Cancel competition
Collect local games' results
Collect Appeals
Resolve appeals
Check answers
Publish results
Recieve the application for competition arrangement
Create competition
Infrom coordinator and organisers about cancellation
Assign local community to competition
Share question package
Specify organizer, moderator, fee and other local game aspects
Sign the local community up to competition
Get question package
Inform local community about the game
Send local game results (participants, answers)
Print questions and perform technical preparations
Local game process
Collect payments and appeals
Rate question package
Send appeal

Figure 3.2: Competition organising process

3.3.2 Shared database of questions

The database stores question packages as an achieves that contain:

- *Package manifest* that is a text file in JSON-based format that specifies text of the questions and meta-data (including relative paths to enclosed images),
- Enclosed images that can be a part of some questions.

The module contains a user-friendly package online editor for generating and editing the questions. It also supports downloading the packages as archives (that is needed for integration with SwS) and as generated PDF (in the form, that is convenient for printing – the images are paced in the separate pages so that it is easy to cut them out as *supportive material* if the game is handled without SwS). The packages could be presented just as JSON files containing images transferred into raw data, but the solution to present them as stand-alone files seems to be more appropriate for this system.

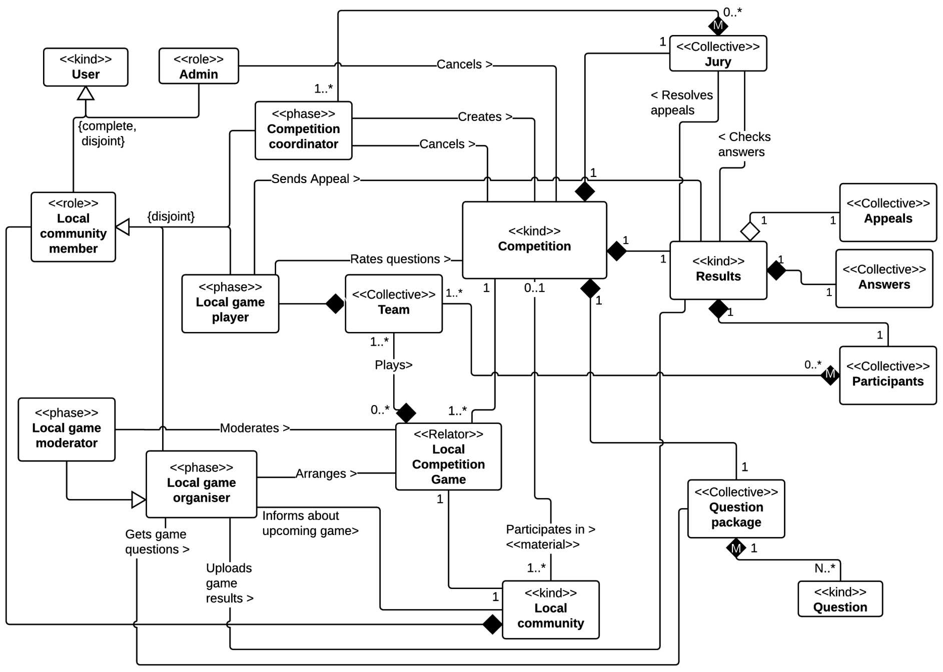


Figure 3.3: Conception of game organising IS module

The database consists of two parts: the store, where authors offer their paid or free of charge packages to game coordinators and the database containing obsolete packages that are available to every user for free. The conditions of transiting of a question package from the store to the database need to be specified after further wants&needs analysis that include interviewing possible coordinators, authors and common users. In general, it must take place after the package is played a certain number of times.

The packages are sortable by their author, type of game, level of difficulty, language, date and tags that specify their content (for example, common topic of questions or age restrictions).

This module is supposed to be implemented as a Content Management System (CMS). Although there exist ready implementation of such systems it seems to be reasonable to create a custom CMS with the usage of the *Symfony Content Management Framework* (CMF, [50]). A custom CMS would fit the requirements better than a ready solution and using a CMF will make the development easier and faster and will also increase quality of the CMS.

3.3.3 Deploying software

This module is supposed to be implemented as a wiki web site (a web site created with MediaWiki engine[51]), containing software manuals, documentation, feedback, support, and, above all, links for distributing the software itself.

Applications for OS *Android* are available on *Google Play* and applications for *iOS* are available on *Apple Store*. All other software (JAR files with desktop applications and scripts for network set-up) is available for download from the server of the IS. The links for downloading desktop applications, scripts and their dependencies (JRE, for example) are presented in the main page of the wiki together with *Google Play* and *Apple Store* badges for installing mobile applications.

3.3.4 Other

The most important module that implement the other functionality is the module of administering users and content. It must present in the prototype together with the basic modules mentioned above. Next importaint module is payment. Ideas for other modules can be picked during further analysis. Maybe modules for accounting (that can be used by users, who run their games activity in a form of an official organisation) or for finding places for games.

3.4 Summary

The analysis, design and choice of technology is performed in this chapter. This information is a valid input for implementing testing a prototype of IS. The further steps suggested for this project are:

- implementing and testing a prototype,
- analysis of testing outputs,
- user analysis,
- improving and completing domain analysis, design and implementation,
- deployment of the IS,
- support and maintenance of the IS.

In the next iteration of analysis the payment methods must be researched. That task has a vital importance for business application. A draft solution is to use virtual points for payments inside the IS (for question packages or participation fees). These points are attached to a user account. They can be bought by online payments via third-party systems (like PayPal). Then the fees are paid by transferring these points to the proper users (from coordinator to the author of package if the coordinator byes the package, for example). Finally, the user can transfer his or her points into real currency. It is possible to specify different price for the points in different currencies that allows to implement price discrimination (offering the goods for different price to different groups of customers) that appears to be a more effective policy than a fixed price. However this solution can seem too complicated to the users and it implementing a custom payment system (paying with virtual points) requires advanced testing of the security and reliability. Maybe an another solution would be better in this case.

Also, the administration module must be implemented.

Chapter 4

Integration

4.1 Software system and Information system

Integrating SwS and IS creates a joined platform that handles both arranging game competitions and game process of every single game. That makes the game process more convenient and reduces the influence of the human factor on the results of the games.

In this chapter the analysis of integration is performed in order to suggest additional functionality to the platform (SwS together with IS) that can be implemented with that integration.

The additional functionality is described in additional function requirements that are covered with this additional functionality. This approach may appear "upside-down", but it produces a structured and rather clear representation. The additional requirements are formulated in the following analysis.

4.1.1 Analysis

4.1.1.1 Requirements

Proper integration of the SwS and the IS can enable the functionality that is formulated by the following requirements to the platform consisting of the integrated systems:

FSIS1 – Direct game question download
The question package of an arranged game is downloaded to the *Moderator* application (component of SwS, see 2.4.3) that is associated with the IS user that is set as the moderator of this game.

FSIS2 – Direct game results upload [4]
The game results are uploaded to the IS during the game or directly after the game. The results are uploaded automatically, the information

[4]This requirement is based on the output of the interview from the previous chapter

is not transferred from one form to another (i.e. form raw game results to the IS) by man.

FSIS3 – Current global score overview
The currents results from the other local communities that run the same game can be displayed in the *Moderator* and *Player* (see 2.4.4) components of SwS.

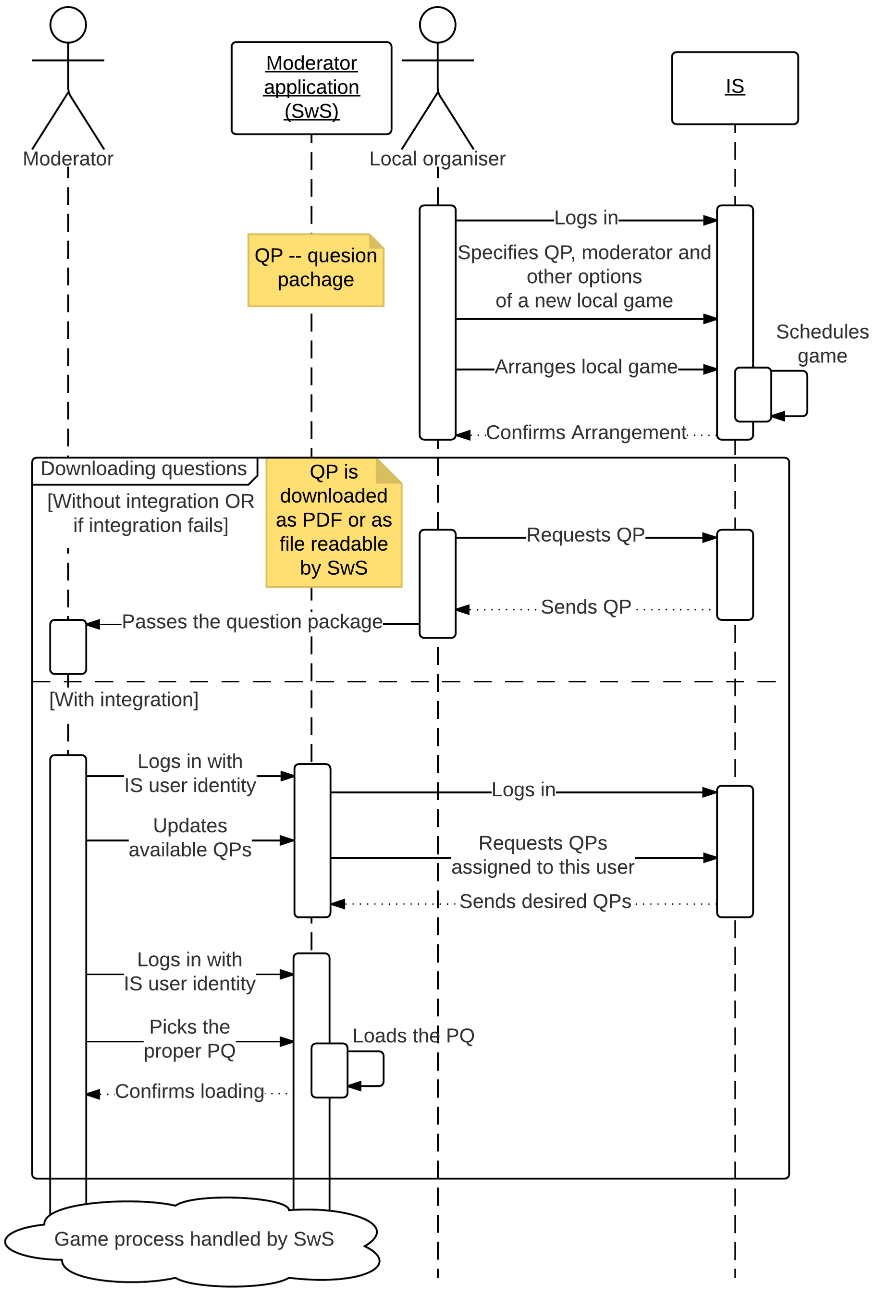
FSIS4 – Sharing WLAN configuration
The configuration of WLAN (SSID, password, authentication method) used for the arranged game can be defined in the IS. Then the *Moderator* and the *Player* applications that are associated with the users who take part in the this game obtain this information. After that, it is possible to establish the network with minimal user's efforts (without specifying the WLAN information with GUI).

FSIS5 – Deploying extensions
The *Moderator* application is supposed to be able to handle different types of games that are implemented in the application's extensions available on the IS.

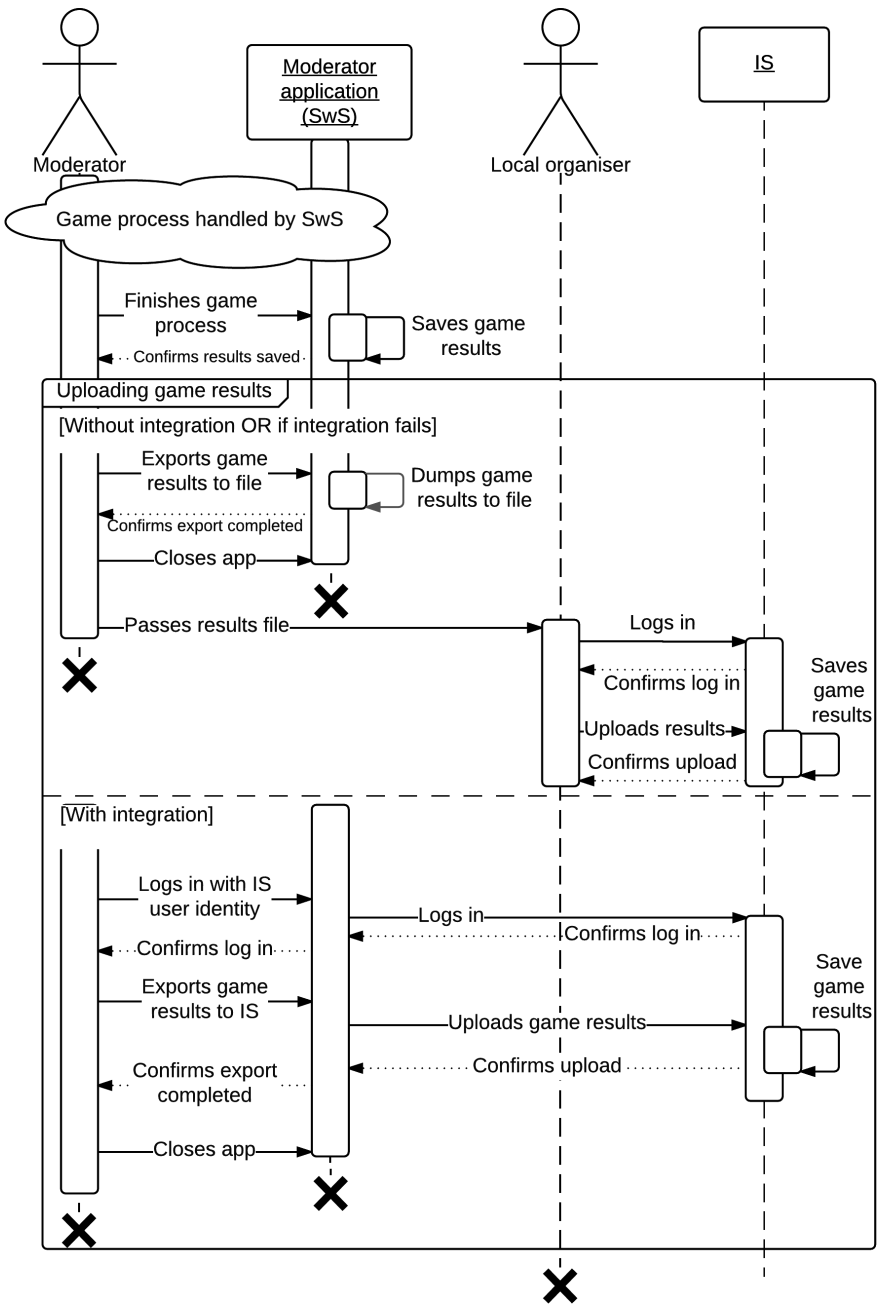
This functionality represents the options that are suggested to be the most valuable improvements of the platform.

FSIS1 and FSIS2 are supposed to make the job of moderator and organiser easier, reduce the amount of boring work and minimise the chance of mistakes on their side. Also FSIS2 implies almost immediate delivery of game results to the coordinator. According to the interview (see 3.2.1), problematic deadlines for this process and technical details of uploading (setting proper format and encoding for the file with results) can bother the user. FSIS3 might make the game process more exiting, by sharing the global information about the current game online. FSIS4 must be mostly appreciated by the users of mobile devices because these devices' interface is usually less inconvenient for connecting to WLAN (typing a password into a touch-screen is challenging to the author sometimes).

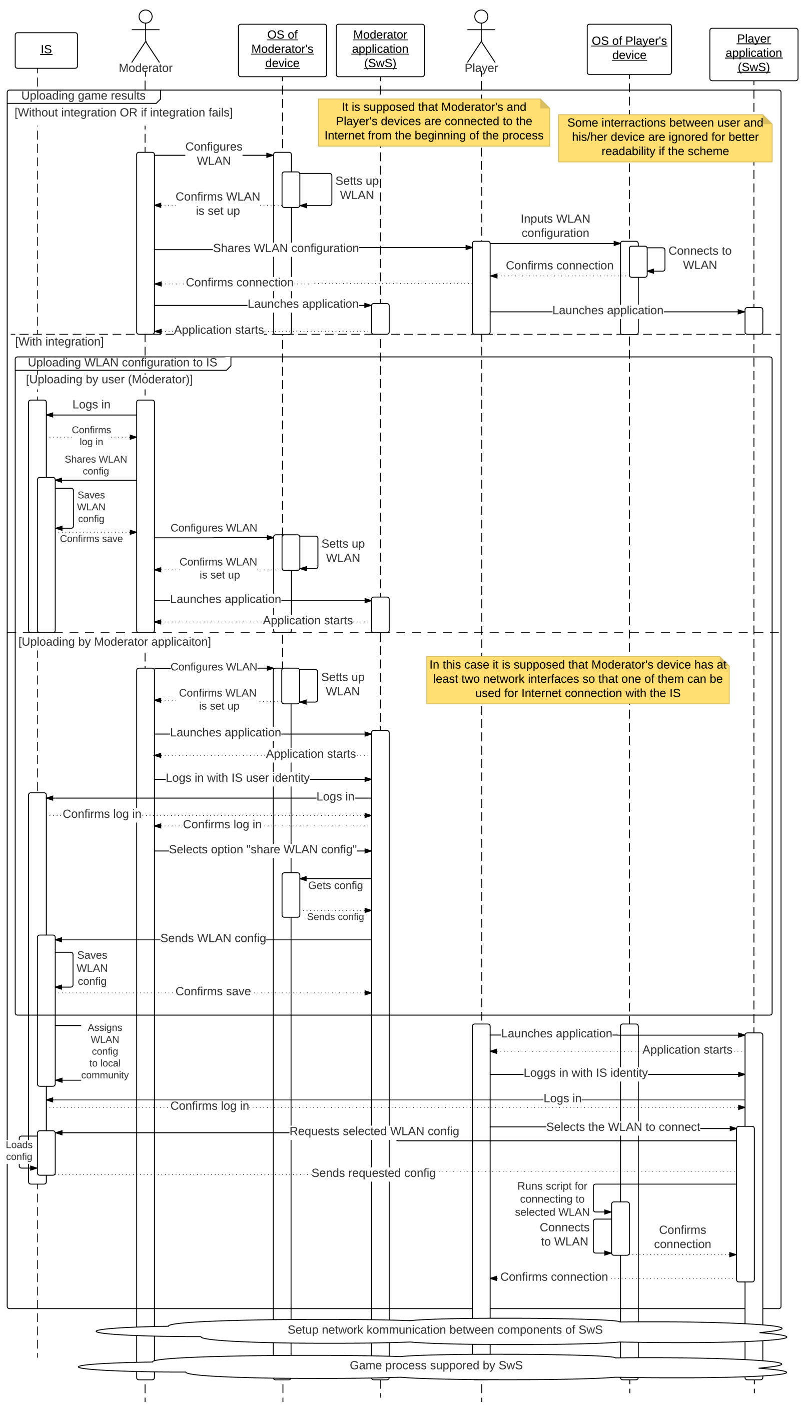
The requirements FSIS1, FSIS2 and FSIS3 are suggested to be the most important. On the Figure 4.1 the processes that can be improved by implementing these requirements are displayed with UML sequence diagrams. These diagrams contain variations of each process with and without integration (FSIS1 implementation is on Figure 4.1a, FSIS2 is on Figure 4.0b and FSIS3 is on Figure 4.-1c).


Moderator
Moderator application (SwS)
Local organiser
IS
QP -- quesion pachage
Logs in
Specifies QP, moderator and other options of a new local game
Schedules game
Arranges local game
Confirms Arrangement
Downloading questions
[Without integration OR if integration fails]
QP is downloaded as PDF or as file readable by SwS
Requests QP
Sends QP
Passes the question package
[With integration]
Logs in with IS user identity
Logs in
Updates available QPs
Requests QPs assigned to this user
Sends desired QPs
Logs in with IS user identity
Picks the proper PQ
Loads the PQ
Confirms loading
Game process handled by SwS

(a) Additional integration finctionality: Question package download



(b) Additional integration finctionality: Game results upload


IS
Moderator
OS of Moderator's device
Moderator application (SwS)
Player
OS of Player's device
Player application (SwS)
Uploading game results
[Without integration OR if integration fails]
It is supposed that Moderator's and Player's devices are connected to the Internet from the beginning of the process
Some interractions between user and his/her device are ignored for better readability if the scheme
Configures WLAN
Setts up WLAN
Confirms WLAN is set up
Inputs WLAN configuration
Shares WLAN configuration
Connects to WLAN
Confirms connection
Confirms connection
Launches application
Launches application
Application starts
[With integration]
Uploading WLAN configuration to IS
[Uploading by user (Moderator)]
Logs in
Confirms log in
Shares WLAN config
Saves WLAN config
Confirms save
Configures WLAN
Setts up WLAN
Confirms WLAN is set up
Launches application
Application starts
[Uploading by Moderator applicaiton]
In this case it is supposed that Moderator's device has at least two network interfaces so that one of them can be used for Internet connection with the IS
Configures WLAN
Setts up WLAN
Confirms WLAN is set up
Launches application
Application starts
Logs in with IS user identity
Logs in
Confirms log in
Confirms log in
Selects option "share WLAN config"
Gets config
Sends config
Sends WLAN config
Saves WLAN config
Confirms save
Assigns WLAN config to local community
Launches application
Application starts
Loggs in with IS identity
Logs in
Confirms log in
Selects the WLAN to connect
Requests selected WLAN config
Loads config
Sends requested config
Runs script for connecting to selected WLAN
Connects to WLAN
Confirms connection
Confirms connection
Setup network kommunication between components of SwS
Game process supported by SwS

(c) Additional integration finctionality: Sharing WLAN settings

Figure 4.1: The processes improved by integration

4.1.2 Design suggestions

This integration can be implemented by using a custom API for the IS that supports:

- login,
- download of the current question package in the proper representation,
- upload of game results,
- upload and download of WLAN configuration,
- download of software extensions.

Also, also the corresponding changes in design of the IS must be made, for example representation of the WLAN configuration must be added to local game.

4.2 Social networks

Social networks are an important source of information for many people in today's world. According to the the author's present observations and interview (see 3.2.1), they are widely used by *intellectual games* communities for keeping in touch with their members and attracting new people. However, this communication is usually performed manually by the organisers (by copying and pasting the same announcements in different online media, for example). It can be a good idea to use the community's experience of applying social networks for sharing information and performing feedback, but the solution implemented in the platform must reduce the amount of the "copy and paste" job.

4.2.1 Analysis

The author has reviewed several options of integration IS with social networks (integrating SwS with social networks was not reviewed because of lack of time and because proper integration of IS with social networks and IS with SwS transitively makes SwS integrated with social networks). The suggested options implement functionality that covers the following functional requirements to the IS:

FSN1 – Log in and sign up with social networks' accounts
This functionality is widely spread on modern web-sites. It implies an option of creating a user account or authenticating with a user account of a social network (such as *Facebook*, *Goodle+*, *VK*, etc.). That brings more convenience to the user: everything is done by a single click on a

corresponding button (no need to fill any fields, waiting for an e-mail conformation, etc.).

FSN2 – Centralised sharing game information

Sharing of information about upcoming games includes publishing the same information with various tools such as e-mail, web-sites of the communities and above all social networks. This creates a lot of tiresome routine game organiser's work that must be reduced. However, the information must be shared to the largest number of people possible and the sender of information must be able to find out which information has reached the audience and which has not. The perfect solution is one button that posts the defined post with announcement on every social media account associated with the certain local game community. The same button sends the information by email and uses other possible media for reaching the members of community.

This requirement partly duplicates the IS requirement FG4, but it is reasonable to specify this functionality once more in the context of integration with social networks.

FSN3 – Centralised feedback

The other side of functionality required in FSN2. It is clear that the persons who get information from different sources use different means of feedback. For example, if Alice sees a game announcement post on her *Facebook* feed, she most probably will use comments of the feed to inform the organiser about her absence, for instance. If Bob gets e-mail with game announcement, he will express his feedback in an response e-mail. The game organiser must be able to browse all feedbacks in the one place.

4.2.2 Design suggestions

Based on information form the IS analysis (see **??**), the author can suggest that *Facebook* [52] and *VK*[53] are the most popular social networks among members of this community, so the greatest attention must be paid to them. Tools for work with the *Google* account can be also important mostly because of the popularity of Gmail service.

A brief review of these platforms' tools that can be used for implementing the requirements from the analysis (in this section implementing of the functionality defined in a requirement is meant by phrase "implementing requirement") is given below.

Facebook offers a range of developer tools for implementing the requirements [54]. FSN1 can be implemented with *Account Kit.* For FSN2 *Staring* tools that allow adding *Facebook* sharing interfaces to the web-page can be used and *Social plugins* like *Share button* or *Page plugin* might help too. *Pages*

API can be another useful tool that allows to "manage Facebook Pages from app"[55]. For FSN3 the same tools as for FSN2 can be used.

VK API for web sites [56] can be used for integrating with *VK*. It handles authorization (needed for FSN1), posting button and community module that can be used for implementing FSN2 and FSN3.

Google Identity Platform[57] can be used for implementing FSN1 allowing login and sign up with *Google* account.

In the later analysis further options of integration are suggested for review. For instance, solutions for automated sharing of game information via Skype and messengers like WhatsApp and Telegram can be used.

4.3 Summary

In this chapter additional requirements are specified and technologies for their implementation are reviewed. In the next iteration of the project this analysis is useful for improvements in design and implementation of the SwS and the IS. If these improvements are performed correctly, this integrated software platform will successfully play its role of an essential part of the new community of quiz games.

Chapter 5

Business application

In this chapter a review of possible business application of the platform is given, consisting of the vision of the project, roadmap, risk and SWOT analysis.

5.1 Specification of business

5.1.1 Form of business organisation

Factors that motivated the author to bring the ideas of the thesis to life are not only commercial. His more ambitious aim is an attempt to create an alternative community of intellectual games not limited by Russian speakers and therefore truly international. However, this community is supposed to generate some income because the project needs to be maintained and developed. Therefore, the project is supposed to be implemented by a commercial organisation. This organisation is a Společnost s ručením omezeným (s.r.o.)[5]. Thit type of organization is chosen because it is more safer (the owner risks only with the property or organization). If the commercial application of the system is commercially unsuccessful, an attempt to run the project as a non-profit business might be made.

5.1.2 Business strategy

The project itself consists of providing two major products: the SwS components for handling local games and the IS which is a platform that can be used for arranging games and that can also bring some money to game organisers and question package authors. After adding a module for owners of the platform, it will be also a tool for managers of pubs and bars to attract clients regularly.

[5]this term is not translated to English because its meaning is a type of organisation that is defined by Czech law under this term.

Important part of the project is offering free of charge versions of software. Free usage of *Player* is available to all customers. Small game communities and educational institutions are able to use a lite version of *Moderator* application for free. These steps are aimed at increasing the popularity of the games which implies higher demand on their complementary goods that are are the paid versions of *Moderator* application (that are used to run large competitions and implement some specific functions for commercial game organisers) or some extra user IS functionality.

The increase of the games' popularity also implies growing of the amount of games organised and question packages sold. Their organisers and the authors can set a fee for their efforts and in this case they pay a small part of that fee for using the platform. Making all question packages available for free after they are played several times is likely to increase popularity of the project among ordinary users and stimulate the authors to create new questions.

The marketing strategy uses the approach of price discrimination that is offering the products to the different customer groups for a different price. For students, small game communities and educational institutions special discount is valid (in addifion to free and lite products, designed for them). Also, the price is dependent on the quality if life in the client's region. These measures should make the marketing policy more effective.

5.1.3 Target groups

Most of the users are supposed to be people in the age group from 16 to 40 who are interested in educational and cultural events. In the beginning it will be mostly used by Russian-speaking users, but the design of the project and its philosophy should attract more people of different cultures.

5.1.4 Technical realisation

The run of IS plays essential role for the whole business, since it handles all processes that generate income. That makes technical its realisation vitally important. Maintaining of a server appears to be complicated enough to the author, he suggests delegating part of these problems to web hosting.

Establishing a the business's own server might happen in several years, if the business is relatively successful and less risky, which implies more reasons for greater investments.

5.1.5 Values and policy

The major motivation factor of this project is creating a cosmopolitan *intellectual games* community. From a business point of view, a loyal community of users is a great asset for this project (*Apple* and its fans is a bright example of it). The community can be built by various means including the project's values and policy.

The basic values that define the ethics and policy of the project are believed to be open-mindedness, sense of humour, friendship, equality, neutrality and tolerance. These values should be attractive to a great part of the target group and they do not differ much from the values of *intellectual games*, that must make the project attractive to a lot of them too.

Both values and policy mush be completed and improved before thestart of the project.

5.2 Roadmap

1. Implement new the SwS based on the analysis, design and testing outputs, presented in this thesis 2. A new repository will be initiated for versioning the code.

2. Test the SwS (user testing). Publish the free versions, if the testing is successful.

3. Complete analysis and design of the IS,

4. Implement the IS.

5. Test the IS (user and stress testing).

6. Formulate the user agreement for IS and licences for SwS.

7. If all previous steps are successful pick a name for the project, create s.r.o., register Internet domain. Otherwise, cancel project continue it as a non-profit hobby.

8. Improve the IS by the outputs of testing.

9. Implement the IS integration with the SwS.

10. Implement and publish the basic extensions for the SwS (supporting of *What? Where? When?* and several other games)

11. Select a hosting and start the business part of project (Support and mantain the IS and SwS that are available to the public).

12. Launch the life of community by filling the question database with some packages. The authors of the packages can be found among authors from *intellectual games* community. They can be appreciated financially or by exclusively beneficial conditions of using the platform.

13. Order English questions or English translation of some original packages for launching the English-speaking part of community. Perform the same action with other languages that have a potential (speakers of which express interest in the community).

14. Implement the IS integration with social networks.
15. Support, mantain and expand the platform.

N.B.: every stage that contains implementation implies unit testing and regression testing of the implemented item.

5.3 Risks

5.3.1 Low popularity and high competition

There is a lot of alternatives to quiz games and the project will face a great competition from their side. Since the phenomenon of *intellectual games* in not known to a lot of people, obtaining their attention is difficult without large marketing investments.

Impact Low demand, failure of the attempt of making the community break boarders of the Russian-speaking community.

Mitigation High quality and affordability of the platform component, offering the platform for popular public places (pubs, bars, cultural and educational institutions) on exclusively beneficial conditions.

5.3.2 The users' distrust

A part of the Russian-speaking community can take a foreign platform negatively. On the other hand, the community that is inspired by a Soviet phenomenon of *intellectual games* and created by a Russian citizen can be suspicious to the rest of the world.

Impact Low demand, loss of users.

Mitigation Following the policy of openess, neutrality and globalism, avoiding partnership with any suspicious political, cultural, religious or other organisations and persons. In case of a strict choice between sympathy of Russian-speaking and international parts of the game community, the international popularity must prevail, since the goal of the project is bringing the variation subculture of *intellectual games* outside of the Russian-speaking environment.

5.3.3 Payment methods

Some members of *intellectual games* community got used to pay in cash and transfer with other members of the community who travel to the desired destination. Such people may dislike the platform because it requires online payments.

Impact Loss of client.

Mitigation Offer diverse, user-friendly and cheap payment methods. Make the platform attractive by improving it's quality.

5.3.4 Dependence on questions

If new question packages stop to appear, there is less games and there are no online competitions. The problem can be very usual in the situation of starting a new language community, when there is not much authors of questions who write in that new language.

Impact Slowing or blocking of the life of the project.

Mitigation Ordering new packages of questions or translations of old packages to new languages. Making author's activity more beneficial (reducing the payments, collected form them, letting sell the same package for more time, before it becomes available for free).

5.4 Summary

The project's overview is summarised with SWOT analysis.

Strengths

- Integrated platform that makes game organisation and game process easier and more convenient.
- Free products that create demand for paid ones.
- Little investmets
- Multi-language community and software.
- Attractive community values.
- Opportunity to earn money (for organisers and authors).
- If a language community is rather developed (it contains a proper number of the authors who write regularly), it generates question packages itself.

Weaknesses

- This thesis is accessible to the public and its license allows anyone to utilise it (BSD-like license). It might be useful to possible competitors, although this thesis does not provide enough information to start an alternative project.
- The SwS and the IS require a lot of further analysis, design and development.
- The project is dependent on the amount of question packages and their authors.
- Possible distrust from different parts of target group.
- Need for creating question packages when the community can not generate it itself.
- The global orientation of the project means that the project must support languages and respect laws of various countries.

Opportunities

- A new kind of entertainment for most people. Might be curious to those who have not had and experience of *intellectual games*.
- Can be attractive to a part of existing *intellectual games* community as a simulations project that offers more interesting global audience, convenient technical solutions and more liberal policy.

Threats

- A lot of members of *intellectual games* community prefer to fees for tournament participation in cash. They probably would not like to pay online (beacuse of possible commissions and lack of trust).
- Great amount of indirect competitors. Generally, the project competes with any business that offers leisure activities.
- Usual risks of an internet business: DDoS attacks, security hazards (personal data, online payments), etc.

The project can be characterised as a rather risky one, but it does not require much investments at the beginning. That is why the author suggests that it if worth implementing. Even if it the the business does not grow or is nit profitable enough, supporting a small cosmopolitan *intellectual games*-like community can be an interesting hobby for the author.

Conclusion

The answer to the question if the main goal (designing a new platform for quiz competitive games that is friendly to a greater number of users than the existing alternatives) is achieved is ambiguous. It is difficult to judge about that without the prototype of the whole platform. On the one hand, the author managed to create a usable prototype and the designed systems appear to be centralised, well-structured and avoid some drawbacks of existing alternatives, keeping their strong sides. However, the design is not detailed enough. The result of the thesis is not a product ready for its business application, it requires several iterations of software process to produce the platform that can be presented to the public.

However, the author gained a lot of important experience while working on this thesis: from developing usable software to conducting an interview and processing its results. Some valuable things were learnt from mistakes (poor time management, erasing part of the code or loosing recordings from field testing "by accident"). The most important lessons learnt during the work over this thesis are: "keep the tools in order" and "respect the scheduling". Also, a remarkable problem was the wrong priority of the tasks (for example, creating diagrams sometimes took much more time than it deserved – and not all diagrams were useful later).

Finally, some day the full-scale implementation of the project might become the true conclusion of this thesis.

Bibliography

[1] Merriam-Webster, Incorporated. Quiz game. [online]. [Cited 2.4.2016]. Available from: http://www.merriam-webster.com/dictionary/quiz%20game

[2] Studiâ 2V. Svoâ igra [television program]. 1994 –, [Cited 14.5.2016]. Available from: http://www.studio2v.ru/projects/programmy/svoya-igra/

[3] Sony Pictures Entertainment Inc. Jeopardy! [television program]. [Cited 14.5.2016]. Available from: https://www.jeopardy.com/

[4] Ludwig, M.; et al. Kile [software]. Version 2.1.3. [Cited 14.5.2016]. Available from: http://kile.sourceforge.net

[5] Oracle Corporation and/or its affiliates. NetBeans IDE 8.1 [software]. 2016, [Cited 14.5.2016]. Available from: https://netbeans.org

[6] Lucid Software Inc. Lucidchart [online]. 2016, [Cited 14.5.2016]. Available from: https://www.lucidchart.com/

[7] Kimball, S.; Mattis, P.; the GIMP Development Team. GIMP 2.8.10. GNU Image Manipulation Program [software]. 2016, [Cited 14.5.2016]. Available from: http://www.gimp.org/

[8] Mardelle, J.-B.; et al. Kdenlive [software]. Version 0.9.6. 2007-2012, [Cited 14.5.2016]. Available from: http://kdenlive.org

[9] LibreOffice contributors. LibreOffice [software]. Version 4.2.8.2. 2000-2014, [Cited 14.5.2016]. Available from: http://www.libreoffice.org/

[10] Higginson, A.; Klasinc, D. Kazam Screencaster [software]. Version 1.5.3. [Cited 2.5.2016]. Available from: https://launchpad.net/kazam

[11] Skype and/or Microsoft. Skype^{TM} [software]. Version 4.3.0.37. 2014, [Cited 14.5.2016]. Available from: `https://www.skype.com/en/`

[12] Software Freedom Conservancy. Git [software]. Version 1.9.1. [Cited 14.5.2016]. Available from: `https://git-scm.com/`

[13] *ČSN ISO 9 (01 0185) Informace a dokumentace - Transliterace cyrilice do latinky - slovanské a neslovanské jazyky.* Praha: Český normalizační institut, 2002, [Cited 13.5.2016].

[14] *ČSN ISO 690 (01 0197) Informace a dokumentace - Pravidla pro bibliografické odkazy a citace informačních zdrojů.* Praha: Český normalizační institut, 2011, §6.1, p. 10, [Cited 13.5.2016].

[15] Cestovatelské stránky www.cesty.in. Transkripce a transliterace textů z azbuky do latinky ONLINE (podle české normy ČSN ISO 9) [online]. 1997-2015, [Cited 14.5.2016]. Available from: `http://cesty.in/transliterace_azbuky_podle_csn_iso_9`

[16] Serezha_zp. Sistema dlâ ≪Svoej igry≫ [online, post on Geektimes]. October 2014, [Cited 24.3.2016]. Available from: `https://geektimes.ru/post/258850/`

[17] OOO TK ≪TORUM≫. Brèjn Sistema [online]. [Cited 1.11.2014]. Available from: `http://xn--1-9sb2c.xn--p1ai/`

[18] Vladimir KHIL. Ličnyj sajt Vladimira Hilâ [online]. [Cited 25.10.2014]. Available from: `http://vladimirkhil.com/si/simulator`

[19] Vladimir(a.k.a ur-quan1986). SImulâtor 2.3: mobil'nyj telefon v kačestve knopki [online, post on LIVEJOURNAL]. [Cited 25.10.2014]. Available from: `http://ur-quan1986.livejournal.com/304044.html`

[20] Vladimir KHIL. SImulâtor [software]. Version 2.5.1. [Accessed 22.3.2016]. Available from: `http://vladimirkhil.com/si/simulator`

[21] Vladimir KHIL. SImulâtor [software]. Version 2.4.6. [Accessed 22.3.2016]. Available from: `http://vladimirkhil.com/si/simulator`

[22] AdlexApps. Mozkovna [software]. Version 1.8.0. [Accessed 3.4.2016]. Available from: `http://hramozkovna.cz/`

[23] THX Games. Triviador [online]. [Accessed 3.4.2016]. Available from: `https://world.triviador.net/`

[24] Camarillo, G. *Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, Examples, and Applicability [online].* Internet Engineering Task Force (IETF), November 2009, [Cited 26.3.2016]. Available from: `https://tools.ietf.org/html/rfc5694#page-3`

[25] Online, E. B. Client-server architecture [online]. *Encyclopædia Britannica Online*, 2016, [Cited 9.4.2016]. Available from: http://www.britannica.com/technology/client-server-architecture

[26] Information Sciences Institute. *TRANSMISSION CONTROL PROTOCOL. DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION [online]*. University of Southern California, 4676 Admiralty Way, Marina del Rey, California 90291, September 1981, [Cited 26.3.2016]. Available from: https://tools.ietf.org/html/rfc793

[27] Socolofsky, T.; Kale, C. *A TCP/IP Tutorial [online]*. Internet Engineering Task Force (IETF), January 1991, [Cited 26.3.2016]. Available from: https://tools.ietf.org/html/rfc1180

[28] Connectify. Connectify Hotspot 2016 [software]. [Cited 3.4.2016]. Available from: http://www.connectify.me/hotspot/

[29] Cisco Systems, Inc. *Configuring IPv4 Broadcast Packet Handling [online]*. November 2011, [Cited 3.4.2016]. Available from: http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipapp/configuration/12-4/iap-12-4-book/iap-bph.pdf

[30] Yuhas, D. Speedy Science: How Fast Can You React? [online]. *SCIENTIFIC AMERICAN, A DIVISION OF NATURE AMERICA, INC.*, May 2012, [Cited 25.3.2016]. Available from: http://www.scientificamerican.com/article/bring-science-home-reaction-time/

[31] Dhomeja, L. D.; et al. PERFORMANCE ANALYSIS OF WLAN STANDARDS FOR VIDEO CONFERENCING APPLICATIONS [online]. *International Journal of Wireless & Mobile Networks (IJWMN)*, volume 3, no. 6, December 2011: p. 63, [Cited 25.3.2016]. Available from: http://airccse.org/journal/jwmn/1211wmn05.pdf

[32] Mills, D.; et al. *Network Time Protocol. Version 4: Protocol and Algorithms Specification [online]*. Internet Engineering Task Force (IETF), June 2010, [Cited 26.3.2016]. Available from: https://tools.ietf.org/html/rfc5905

[33] Mills, D. *Simple Network Time Protocol (SNTP). Version 4 for IPv4, IPv6 and OSI [online]*. Internet Engineering Task Force (IETF), January 2006, [Cited 26.3.2016]. Available from: https://tools.ietf.org/html/rfc4330

[34] Ecma International. *ECMA-404. The JSON Data Interchange Format [online]*. First edition, October 2013, [Cited 4.4.2016]. Available from: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[35] Oracle and/or its affiliates. The Java Technology Phenomenon. About the Java Technology [online]. [Cited 24.3.2016]. Available from: `https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html`

[36] IEEE Spectrum. The 2015 Top Ten Programming Languages [online]. [Cited 24.3.2016]. Available from: `http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages`

[37] BLUETOOTH SIG, Inc. What is Bluetooth Technology? [online]. [Cited 4.4.2016]. Available from: `https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth`

[38] BLUECOVE TEAM. BlueCove [online]. [Cited 4.4.2016]. Available from: `http://bluecove.org/`

[39] Holub, J.; Plicka, M. *Automaty a gramatiky (BI-AAG). Programová a obvodová realizace NKA a DKA [presentation from public lections]*. Katedra teoretické informatiky, Fakulta informačních technologií ČVUT v Praze, 2015, slide 5; [Cited 30.3.2016].

[40] Baza voprosov ≪Čto? Gde? Kogda?≫. Baza voprosov ≪Čto? Gde? Kogda?≫ [online]. 2016, [Cited 18.4.2016]. Available from: `http://db.chgk.info/`

[41] COURAGE, C.; BAXTER, K. *Understanding your users: a practical guide to user requirements methods, tools, and techniques.* The Morgan Kaufmann series in interactive technologies, San Francisco: Morgan Kaufmann, 2005, ISBN 1-55860-935-0.

[42] Zwass, V. Information system [online]. *Encyclopædia Britannica Online*, 2016, [Cited 9.4.2016]. Available from: `http://www.britannica.com/topic/information-system`

[43] Merriam-Webster, Incorporated. Database. [online]. [Cited 16.4.2016]. Available from: `http://www.merriam-webster.com/dictionary/database`

[44] Meždunarodnaâ associaciâ klubov "Čto? Gde? Kogda?". [Official website] [online]. [Cited 18.4.2016]. Available from: `http://mak-chgk.ru/`

[45] Sajt rejtinga sportivnogo ČGK. "Čto? Gde? Kogda?"(sportivnaâ versiâ). Oficial'nyj rejting MAK [online]. 2016, [Cited 7.5.2016]. Available from: `http://rating.chgk.info/`

[46] Meždunarodnaâ associaciâ klubov "Čto? Gde? Kogda?". PORÂDOK POLUČENIÂ LICENZIJ NA PROVEDENIE SOREVNOVANIJ PO

IGRE “ČTO? GDE? KOGDA?” (S 2014) [online]. Available from: http://mak-chgk.ru/komissii/sertlic/licenz/licenztur2014/

[47] Baza voprosov ≪Čto? Gde? Kogda?≫. Licenziâ na ispol’zovanie voprosov iz Bazy Voprosov Internet-kluba ”Čto? Gde? Kogda?” [online]. 2016, [Cited 6.5.2016]. Available from: http://db.chgk.info/copyright

[48] Vladimir KHIL. SImulâtor: Svoâ igra [...].

[49] Ontology & Conceptual Modeling Research Group NEMO. *OntoUML Specification 1.0*. September 2015, [Cited 14.5.2016].

[50] Potencier, F. SymfonyCMF [software]. 2016, [Cited 10.5.2016]. Available from: http://cmf.symfony.com/

[51] Wikimedia Foundation, Inc. MediaWiki [software]. 2015, [Cited 10.5.2016]. Available from: https://www.mediawiki.org/wiki/MediaWiki

[52] Facebook. Facebook [online]. 2016, [Cited 11.5.2016]. Available from: https://www.facebook.com/

[53] V Kontakte Ltd. VK [online]. 2016, [Cited 11.5.2016]. Available from: https://vk.com/

[54] Facebook. Facebook for developers. Documentation [online]. 2016, [Cited 12.5.2016]. Available from: https://developers.facebook.com/docs/

[55] Facebook. Facebook for developers. Documentation. Pages API [online]. 2016, [Cited 12.5.2016]. Available from: https://developers.facebook.com/docs/pages

[56] V Kontakte Ltd. VK Developers [online]. 2016, [Cited 11.5.2016]. Available from: https://vk.com/dev/sites

[57] Google Inc. Google Identity Platform [online]. [Cited 11.5.2016]. Available from: https://developers.google.com/identity/

Appendix A

Acronyms

TV	Television
GIMP	GNU Image Manipulation Program
ČSN	Česká technická norma
ISO	International Organization for Standardization
SwS	Software system
IS	Information system
OS	Operating system
MS	Microsoft
IP	Internet Protocol
P2P	Peer-to-peer
TCP	Transmission control protocol
UDP	User Datagram Protocol
WLAN	Wireless local area network
NTP	Network Time Protocol
SNTP	Simple Network Time Protocol
UI	User interface
JSON	JavaScript Object Notation
HTTP	HyperText Transfer Protocol
URI	Uniform Resource Identifier

GUI	Graphical user interface
FA	Finite automaton
UML	Unified Modeling Language
OOP	Object-Oriented Programming
VCS	Version Control System
IDE	Integrated development environment,
USB	Universal Serial Bus
ID	Identifier
JRE	Java Runtime Environment
PC	Personal computer
MAK (ČGK)	Meždunarodnaâ associaciâ klubov ("Čto? Gde? Kogda?") [International Association of Clubs (of "What? Where? When?")])
CSS	Cascading Style Sheets
fb2	FictionBook
XML	Extensible Markup Language
CSV	Comma-separated values
WWW	World wide web
PHP	PHP: Hypertext Preprocessor
ORM	Object-relational mapping
ACL	Access control list
MVC	Model–view–controller
FIT	Fakulta informačních technologií
BI-PWT	Podnikové webové technologie
PDF	Portable Document Format
CMS	Content Management System
CMF	Content Management Framework
SSID	Service set identification

API	Application programming interface
SWOT	Strengths, weaknesses, opportunities, threats
s.r.o.	Společnost s ručením omezeným

Appendix **B**

Contents of enclosed media

- `readme.txt` .... the file with this media contents description
- `Thesis` .... thesis text and its LaTeXsource
  - `src` .... source of the thesis text in LaTeXformat
    - `images` .... images used in the thesis
  - `text` .... the thesis text
    - `BP_Vorobyev_Vladimir_2016.pdf` ... the thesis text in PDF format
- `SwS` .... SwS prototype and Use Case model
  - `UCmodel` .... SwS Use Case model (LaTeXsource and PDF)
  - `Prototype` .... SwS prototype implementation
    - `Sources` . zip archives of *NetBeans* projects (including source code)
    - `Distribution` .... distributions of SwS components
      - `Networking` . simple scripts for setting-up network, needed for a test run of the system
    - `UserTesting` .... user testing materials and outputs
      - `Outputs` .... outputs of testers' survey
        - `Video` .... video recordings of testing
- `IS` .... analysis and design of IS
  - `Interview` .... recording and translation of the interview
  - `Design` .... design schemes