Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Softwarová aplikace pro turnaj ve stolním tenisu

Bakalářská práce

Autor: Tomáš Zákravský, DiS.
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Mgr. Tomáš Kozel, Ph.D.

Odborný konzultant: doc. Ing. Václav Janeček, CSc.

Hradec Králové duben 2015

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 27.4.2015 Tomáš Zákravský

Poděkování:
Děkuji vedoucímu bakalářské práce doc. Mgr. Tomáši Kozlovi, Ph.D. za metodické vedení práce a doc. Ing. Václavu Janečkovi, CSc. za odbornou konzultaci.

Anotace

Cílem práce je vytvoření podpůrné aplikace pro organizaci průběhu, zpracování a zveřejnění výsledků turnaje.

Tomu předchází seznámení se s obecnými zásadami návrhu, s možnostmi postupů při vývoji aplikace, s analýzou průběhu turnaje a s požadavky na zpracovávaný software. Následuje výběr vhodných technologií, představení různých architektur a výběr vhodných programovacích jazyků s využitím frameworků pro zjednodušení programování.

Annotation

Goal is to create supportive application for managing, processing and publishing tournament results.

This precedes introduction to the general principles of design patterns and available approaches for developing the application, tournament analysis and software requirements. Following the selection of appropriate technologies, introduction to various architectures and selection of suitable programming languages with using frameworks for programming simplification.

Title: Software application for table tennis tournament

Obsah

1 Úvod ... 8

2 Obecné zásady návrhu a zpracování softwarových aplikací ... 9

2.1 Důležitost návrhu ... 9

2.2 Cíl návrhu ... 9

2.3 Teoretická východiska návrhu ... 9

2.4 Základní pohled na UML ... 10

2.5 Modelování případů užití ... 11

2.5.1 Hranice systému ... 11

2.5.2 Aktér ... 11

2.5.3 Případy užití ... 12

2.5.4 Scénář ... 12

2.5.5 Relace ... 12

2.6 Modelování diagramu tříd ... 13

2.6.1 Analytický model ... 13

2.6.2 Návrhový model ... 14

2.7 Způsoby vývoje software ... 15

2.7.1 Vodopádový model ... 15

2.7.2 Agilní vývoj ... 15

2.8 Závěrem k zásadám a návrhu aplikace ... 17

3 Analýza propozic a vlastního průběhu turnaje ... 18

4 Stanovení požadavků na zpracovávaný software ... 21

4.1 Uživatelské cíle a funkční požadavky ... 21

4.2 Non-funkční požadavky ... 23

5 Výběr vhodných technologií ... 24

5.1 Architektura aplikace ... 24

5.1.1 Centralizovaná architektura 24
5.1.2 Architektura klient-server 24
5.1.3 Třívrstvá architektura 25
5.1.4 Jednovrstvá architektura 25
5.2 Programovací jazyk 25
5.2.1 Java 25
5.2.2 C++ 26
6 Zpracování softwarové aplikace 28
6.1 Východiska aplikace 28
6.2 Kostra aplikace 28
6.3 Prezenční listina 30
6.4 Skupiny hráčů 32
6.5 Pavouk s hráči 34
6.6 Práce se soubory 36
6.7 Automatické aktualizace 37
6.8 Resources 38
6.9 Překlady jazyků 38
6.10 Styly uživatelského rozhraní 39
7 Shrnutí dosažených výsledků 40
8 Závěr 41
9 Seznam použité literatury 42
10 Přílohy 43

Seznam obrázků

Obr. 1 Ukázka aktéra 12
Obr. 2 Příklad případu užití 12
Obr. 3 Příklad jednoduchého diagramu případů užití 13
Obr. 4 Příklad diagramu tříd 14
Obr. 5 Hráči v pavouku. 20
Obr. 6 Diagram případů užití aplikace 21
Obr. 7 Aplikace s prezenční listinou 29
Obr. 8 Záložka skupin aplikace 33
Obr. 9 Diagramy vylučovacího systému hráčů 35

Seznam tabulek

Tabulka 1 Výsledky hráčů každý s každým. 18
Tabulka 2 Dva hráči proti sobě 19
Tabulka 3 Rozřazení skupin do pavouků. 19

1 Úvod

Cílem bakalářské práce je vytvoření podpůrné aplikace pro organizaci průběhu, zpracování a zveřejnění výsledků turnaje. Dále je třeba seznámit se se specifickým průběhem turnaje ve stolním tenisu pro mládež, s technologiemi pro vývoj podpůrné aplikace, která bude zaštiťovat potřebné kroky průběhu turnaje a navrhnout softwarovou aplikaci.

Druhá kapitola naznačuje důvody a správné postupy pro tvorbu objektově orientovaného návrhu. Seznámení se s modelováním diagramu případů užití a class diagramu. Dále zde budou popsány způsoby a metodiky pro organizaci práce při vývoji software.

Třetí kapitola obsahuje popis a analýzu průběhu turnaje ve stolním tenisu ve stávající podobě. Specifikuje navržený postup a návaznost kroků, které je třeba brát v úvahu v dalších krocích návrhu.

Čtvrtá kapitola se zabývá analýzou vytvářeného software a specifikací funkčních a non-funkčních požadavků na něj. Funkční požadavky jsou reprezentovány diagramem případů užití a non-funkční jejich soupisem.

V páté kapitole se nachází soupis vhodných architektur, mezi kterými je možné vybírat při návrhu a zpracování softwaru. Následuje výběr programovacího jazyka a představení multiplatformního vývoje.

Šestá kapitola popisuje jednotlivé kroky a oblasti vlastní implementace vyvíjeného programu. Představeny budou také možnosti začlenění ikon, postup překladu a začlenění těchto zdrojů do spustitelné aplikace.

2 Obecné zásady návrhu a zpracování softwarových aplikací

Tato kapitola popisuje důležitosti a teoretická východiska softwarové analýzy a návrhu. Dále zde bude popsán základní koncept modelovacího nástroje a notace UML a způsoby vývoje software.

2.1 Důležitost návrhu

Nespočet mladých, začínajících, ale i v dnešní době zkušených programátorů opomíjí důležitou část při vývoji aplikací. Touto částí je pečlivě promyslet a analyzovat potřebné funkce a chování jejich výsledných programů. Tím by předešli mnohým komplikacím.

2.2 Cíl návrhu

Účelem softwarového návrhu je seznámení se se správnými postupy, moderními nástroji, metodikami a správným postupem při návrhu a tvorbě objektově orientovaných aplikací, aby nedocházelo k programování ve zmatku a nepřehlednému nabalování kódu. Mělo by zde také být obsaženo ověření vhodnosti jazyka UML a procesu při analýze a modelování výsledného systému.

2.3 Teoretická východiska návrhu

Základním předpokladem je, že výstupem by měla být přehledná (na úrovni zdrojového kódu) objektově orientovaná aplikace. Jedná se o moderní trend vývoje aplikací, kde každý objekt reálného světa lze namapovat či abstrahovat do konkrétních objektů, respektive tzv. obálek neboli tříd, obsahujících vlastnosti a chování v počítačovém programování.

Dříve bylo pro návrh, specifikaci a nastínění problému možné použít pouze tužku a papír, což jistě řada lidí stále využívá pro osobní účely. I pro budoucí aplikaci je to velmi prospěšné. Avšak jedná se o již překonaný způsob a pro členy týmů nepříliš vhodné řešení. V dnešní době jsou dostupné specializované nástroje a metodiky, díky kterým je možné návrh zpřehlednit a zobecnit pro potřeby sdílení s dalšími návrháři a vývojáři.

Nejrozšířenějším vizuálním modelovacím nástrojem pro modelování systémů je jazyk UML (z anglického Unified Modeling Language - jednotný modelovací jazyk) a spolu s metodikou UP (z anglického Unified Process - jednotný proces) tvoří silný prvek pro počítačem podporované softwarové inženýrství CASE (Computer-Aided Software Engineering).

„Návrh aplikace může teoreticky vzniknout pouze za použití tužky, mozku a poznámkového bloku, s jejichž pomocí načrtnete jednoduchý model aplikace. Model ale bude používat jen vám známou syntaxi a sémantiku, takže budete jen těžko své záhadné omalovánky s někým sdílet. Dalším problémem je, že za těmito náčrty většinou nestojí žádný konzistentní logický metamodel ani formálně kodifikovaná sémantika jednotlivých elementů modelu, takže případná komunikace mezi návrháři připomíná situaci v IT Babylónu po zmatení jazyků a je protkána slovy asi, možná a jak to vlastně bylo myšleno.“ Popisuje Stein [2] archaický přístup.

2.4 Základní pohled na UML

Jak napsal Stein [2] „*Model v UML se skládá z různých diagramů, jež představují průhledy na různé části sémantického základu navrhované aplikace. Sémantickým základem je souhrn specifikací aplikace, který vymezuje teritorium, v němž se může návrh pohybovat. Diagram ve vizuální formě vypráví právě jeden konkrétní příběh o aplikaci. Žádný dvourozměrný diagram nemůže zachytit komplexní aplikaci v celku, ale soustředí se vždy právě na jeden důležitý aspekt.*“

UML rozeznává pět základních pohledů na systém, kterými jsou:

- *Pohled případů užití* - vyjadřuje základní požadavky kladené na systém. Všechny následující pohledy jsou vymezeny právě pohledem určeným pomocí případů užití.
- *Logický pohled* - zabývá se zejména pojmy z problémové domény zadavatele a jejich vzájemnými statickými vztahy.
- *Procesní pohled* - soustřeďuje se na chování systému, které musí splňovat požadavky a omezení z případů užití, které jsou kladeny na průběh procesů. Jedná se o procesně orientovaný doplněk logického pohledu.

- *Implementační pohled* - zachycuje fyzické rozdělení aplikace na samostatné komponenty a jejich závislosti.

- *Pohled nasazení* - mapuje komponenty na množinu fyzických výpočetních uzlů v cílovém prostředí.

2.5 Modelování případů užití

Modelování případů užití UC (Use Case) se skládá z následujících aktivit Arlow [1]:

- Vymezení hranic systému
- Určení aktérů
- Nalezení případů užití
- Specifikování případů užití
- Určení scénářů
- Vymezení relací mezi aktéry a případy užití
- Opakování postupu až do ustálení diagramu

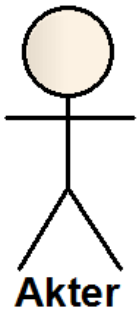
Prvotním předpokladem je stanovení, v jakém rozsahu bude na systém pohlíženo, tedy stanovení hranic systému. Co bude chápáno jako jeho součást a co už nikoliv. Toto východisko pak slouží ke správnému zaměření se na budoucí modelované aktivity odehrávající se uvnitř aplikace a ne vně. Tyto akce se nadále provádějí opakovaně s jinou úrovní abstrakce.

2.5.1 Hranice systému

Vymezení hranic systému, co patří vně a dovnitř, je důležité pro správné určení jeho chování, modelování případů užití a určení scénářů. Špatné vymezení hranic může mít za následek nutnost přepracovat celý model. Je nutné odstínit v danou chvíli nedůležité charakteristiky od charakteristik právě vznikajícího softwarového systému.

2.5.2 Aktér

Aktérem může být osoba nebo další systém postavený mimo hranice systému, který je právě modelován. Aktér iniciuje akce a pracuje se systémem. Píše se s velkým počátečním písmenem a je znázorněn viz Obr. 1.



Obr. 1 Ukázka aktéra
Zdroj: Vlastní zpracování

2.5.3 Případy užití

Případy užití jsou jednotlivé funkce či funkční celky, které reagují na určitý vstup aktéra zvenčí a poskytují nebo vykonávají specifické operace. Názvy se píší s velkým počátečním písmenem uvnitř elipsy viz Obr. 2.

Obr. 2 Příklad případu užití
Zdroj: Vlastní zpracování

2.5.4 Scénář

Scénář je definovaný postup úkonů a interakce případů užití s jednotlivými aktéry. Tímto je definováno chování systému a jeho interakce. Určuje, co daný případ užití vykonává za operace. Píše se v jednotlivých očíslovaných bodech v textové podobě.

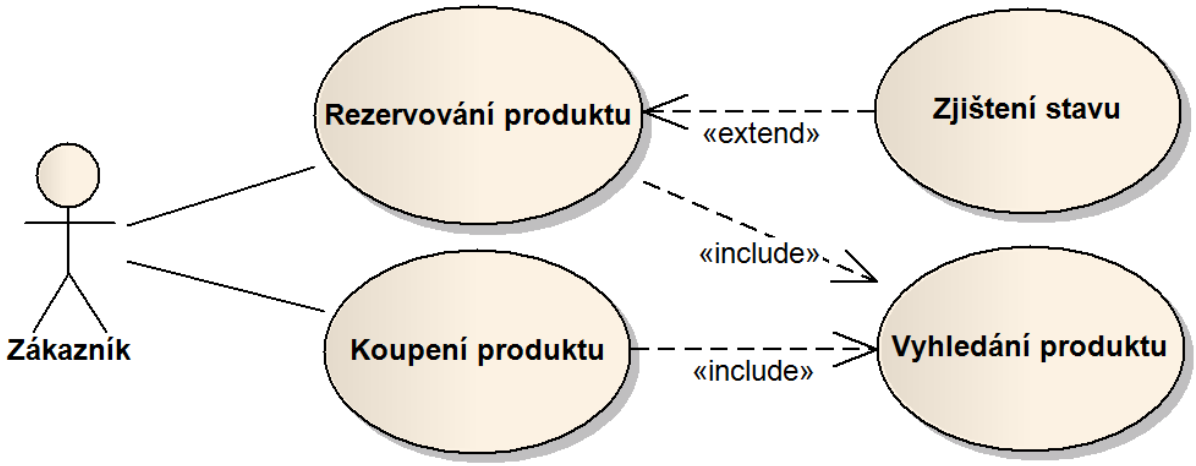
2.5.5 Relace

Relace je vztah mezi jednotlivými případy užití nebo mezi aktérem a případy užití.

Stručný přehled jednotlivých vazeb v UCD (Use Case Diagram):

- Use – základní vazba mezi aktéry a UC

- Include – vazba vyčlenění společné části scénářů do samostatného UC, provádí se vždy, jsou na sobě závislé.
- Extend – rozšíření UC o další chování, vykoná se pouze za určitých podmínek
- Inheritance – specializace jiného aktéra / UC, poděděním jeho vlastností / chování



Obr. 3 Příklad jednoduchého diagramu případů užití
Zdroj: Vlastní zpracování

UCD bývají hlavním zdrojem objektů, respektive vlastností a chování objektů pro následující diagram tříd a výsledné aplikace.

2.6 Modelování diagramu tříd

Názvy tříd, jakožto nedělitelných celků, se píší tzv. CamelCase neboli první písmeno každého slova velké, zbytek malá písmena bez mezer s tím, že jsou použita podstatná jména. Názvy vlastností a metod používají tzv. lower-CamelCase neboli pouze první písmeno celého názvu je malé, avšak u metod používáme slovesa.

2.6.1 Analytický model

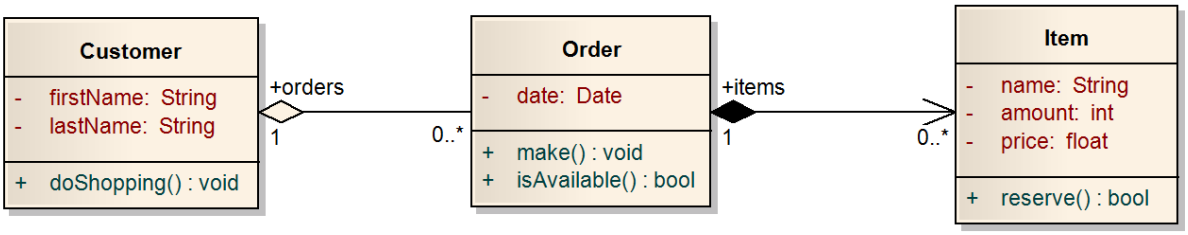
Analytický model mapuje pojmy reálného světa z obchodní oblasti, kterým rozumí i zákazník, na třídy, vlastnosti a metody. Zde ještě stále nezávisí na konkrétní implementaci, tedy na tom, jaká bude použita architektura a programovací jazyk. Důraz je zde kladen pouze na správný popis chování.

Obsahuje pouze vazbu asociace neboli souvislosti mezi třídami.

2.6.2 Návrhový model

Návrhový model rozšiřuje analytický model o podrobnější metody a vlastnosti. Je už také závislý na způsobu implementace. To znamená, že už musí být známa cílová architektura a programovací jazyk, v němž bude software vyvíjen.

Obrázek č. 4 znázorňuje třídy Zákazník, Objednávka, Položka a vazby mezi nimi. Parametry jsou uvozeny znakem - a metody znakem +. Mezi zákazníkem a objednávkou je volná vazba (agregace) – jedna třída bez druhé dává smysl a mezi objednávkou a položkou je silná vazba (kompozice) – třída Položka je zde vyžadována v souvislosti s třídou Objednávka.



Obr. 4 Příklad diagramu tříd
Zdroj: Vlastní zpracování

2.7 Způsoby vývoje software

Způsobů jak přistupovat k řízení a vývoji aplikací je velmi mnoho. Cílem této části je seznámit se se základními způsoby, kterými jsou tzv. Vodopádový (Waterfall) model a Agilní vývoj.

2.7.1 Vodopádový model

Ve vodopádovém modelu se vývoj software řídí striktně sekvenčním způsobem, což znamená postupně přecházet od jedné fáze k následující.

Roycův původní vodopádový model obsahuje sedm fází v následujícím pořadí [3]:

1. Specifikace požadavků systému
2. Specifikace požadavků software
3. Analýza
4. Návrh
5. Implementace
6. Testování a ladění (validace)
7. Instalace a údržba

Tento model klade velký důraz na propracovanost a bezchybnost splnění jednotlivých kroků. Je strukturovaný a tím i snadno pochopitelný. Největší využití je u stabilních projektů, kde se požadavky na systém a software v průběhu celého procesu nemění. Objeví-li se nějaká chyba například již v analýze a je odhalena až při testování nebo údržbě, je nutné přepracovat celý projekt a projít celým nebo téměř celým cyklem znovu. To platí i u následného požadavku na jistou změnu.

2.7.2 Agilní vývoj

Knesl [4] popisuje rozdíl mezi agilní metodikou a agilní technikou takto:

„*Přestože se tyto termíny často zaměňují, nebo dokonce považují za totéž, ve skutečnosti se jedná o rozdílné (jakkoliv propojené) oblasti s naprosto rozdílnými významy.*“

Agilní metodika je způsob rozvržení a ověřování práce. Cílem Agilní metodiky bývá lepší organizace práce. Odlišné Agilní metodiky vedou k odlišným produktům,

protože považují jiné aspekty produktu za klíčové. Liší se i přístup ke změně zadání. Možnost změny zadání je ale určující podmínkou agilního přístupu.

Agilní technika je naproti tomu konkrétní postup, který se většinou týká samotných vývojářů. Cílem těchto technik bývá zvýšení produktivity práce, odstranění chyb, kvalitnější software nebo přesnější dodržení specifikace. Agilní techniky jsou od metodik oddělitelné. Cílem Agilní techniky bývá kvalitnější produkt.

Fáze projektu využívající agilní metodiky

1. **Nultá iterace** – první krátká analýza a implementace základní činnosti. Výsledkem je hotový zlomek aplikace, který je možné předvést zákazníkovi.
2. **Analýza změny** – výběr, analýza a design změn nebo dalších prvků.
3. **Implementace změn nebo dalších prvků**.
4. **Prezentace výsledku zákazníkovi**.
5. **Opakovat stále od bodu 2, dokud není produkt dokončen**.
6. **Údržba a rozvoj** - v iteracích.

Agilní metodiky je vhodné využít především v projektech, kde se požadavky často mění nebo je není možné definovat všechny najednou na začátku.

„Code and Fix“

Styl tvorby software „Code and Fix“ zde slouží pouze jako připomenutí postupu s nejistým výsledkem a chaotickým přístupem, který by se měl v praxi omezit na minimum.

Tento přístup však není možné úplně vyloučit. Nicméně je používán převážně začínajícími kodéry a programátory, kteří se tak mohou zpočátku učit. V dalších krocích autor doporučuje přejít na jiný přístup.

2.8 Závěrem k zásadám a návrhu aplikace

Agilní přístup při vývoji softwarových aplikací se stává preferovanějším i velkými firmami, ne jen malými společnostmi, jak tomu bylo dříve. Přesto jej využívají převážně malé a střední firmy, které jsou pružnější než větší firmy.

Při použití jazyka UML a některé z metodik, je třeba dbát správné sémantiky a předem daných zásad, které přispívají k jednoznačnosti vytvářených diagramů. Je známo, že papír snese vše, nicméně i na něm je třeba kvůli přehlednosti dodržovat určitá pravidla, a proto není přechod k UML velká překážka.

Zapřísáhlí přívrženci metody „Code and Fix“ si snadněji najdou cestu k agilnímu přístupu. Díky jednotlivým iteracím metody je možné jednodušeji upravit svůj přístup. Jednalo by se pak o velmi krátké iterace, které mohou být postupem času prodlouženy.

3 Analýza propozic a vlastního průběhu turnaje

Turnaj pro mládež probíhá speciálním způsobem [5] rozdílným od pravidel stanovených Českou asociací stolního tenisu. Průběh turnaje lze pro přehlednost rozdělit do čtyř částí, které jsou popsány dále.

Prvním krokem turnaje je sepsání prezenční listiny hráčů, jejich jména, pořadí nasazení z předchozího umístění v turnajích, rok narození, oddíl a liga, do které budou hráči zařazeni. Ligy jsou zpravidla tři až čtyři a v každé lize musí být sudý počet hráčů. Každá liga zároveň představuje určitou úroveň a schopnosti hráčů.

Následně jsou hráči z jednotlivých lig rozděleni do dvou nebo čtyř skupin podle počtu hráčů v lize. Rozdělení probíhá zip-způsobem, tj. hráči s lichým pořadovým číslem jsou přiřazeni do skupiny A a se sudým pořadovým číslem do skupiny B. Pokud počet členů ve skupině přesáhne maximální stanovený počet (například 8), jsou hráči v rámci ligy rozděleni rovnoměrně ještě do skupin C a D.

V jednotlivých skupinách se hráči utkávají každý s každým a výsledky se zapisují do tabulky. Hodnocení probíhá tak, že se sečtou míčky hráče, poměr vyhraných a prohraných setů a jeden bod reprezentuje jednu výhru hráče. Z těchto údajů se vypočítá výsledné pořadí pro skupinu s nejmenší váhou míčků a největší váhou bodů.

Tabulka 1 Výsledky hráčů každý s každým.

	A	B	C	Míčky	Sety	Body	Pořadí
A	x	3:1	3:0		6:1	2	1.
B	1:3	x	2:3		3:3	0	3.
C	0:3	3:2	x		3:5	1	2.

Zdroj: vlastní zpracování

Hraje se na tři vítězné sety s tím, že v každém setu musí být odehráno minimálně 9 míčků a vítěz musí mít alespoň o 2 míčky víc. Body z každého zápasu jeden na jednoho se zaznamenávají do tabulky (viz příklad v Tabulce 1).

Tabulka 2 Dva hráči proti sobě.

Hráč	1	2	3	4		Body
A	11	11	10	11		3
B	9	9	12	9		1

Zdroj: vlastní zpracování

Rozřazení skupin do „pavouků“ probíhá tak, že se seřadí hráči podle pořadí od nejlepšího a postaví se proti sobě vždy lichý ze skupiny A proti sudému ze skupiny B a sudý ze skupiny A proti lichému ze skupiny B. Tito čtyři hráči mezi sebou pak soupeří o 1. - 4. místo a o další umístění.

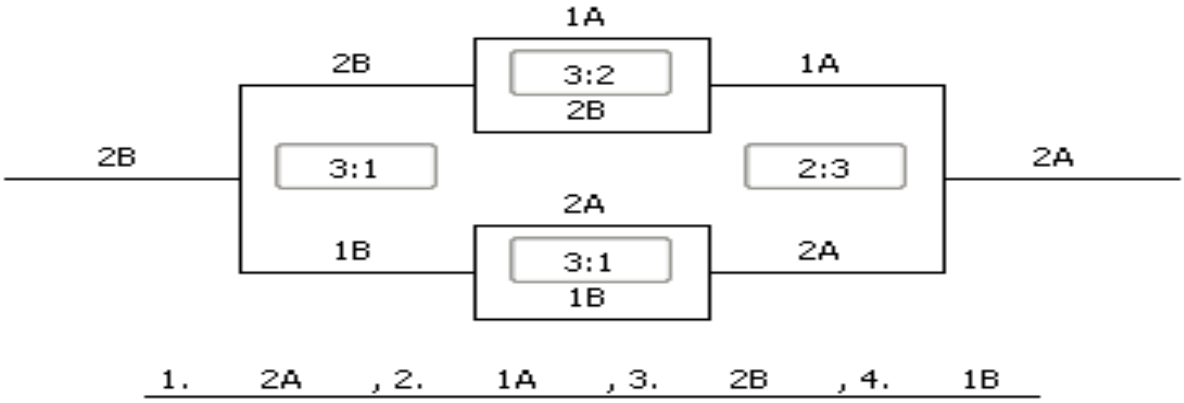
Tabulka 3 Rozřazení skupin do pavouků.

Liga	Skupina A	Skupina B	Pořadí
1.	1A	1B	1. - 4.
	2A	2B	
	3A	3B	5. - 8.
2.	1A	1B	
	2A	2B	9. - 12.
	3A	3B	

Zdroj: vlastní zpracování

Pavouk je zpracován tak, že vítěz postoupí na pravou stranu a poražený na levou stranu. Vítězové bojují o první a druhé místo, zatímco poražení o místo třetí a čtvrté viz obrázek Obr. 5.

Tímto způsobem budou odehrány všechny zápasy a na základě jejich výsledků je stanoveno výsledné pořadí hráčů celého turnaje.



Obr. 5 Hráči v pavouku.
Zdroj: vlastní zpracování

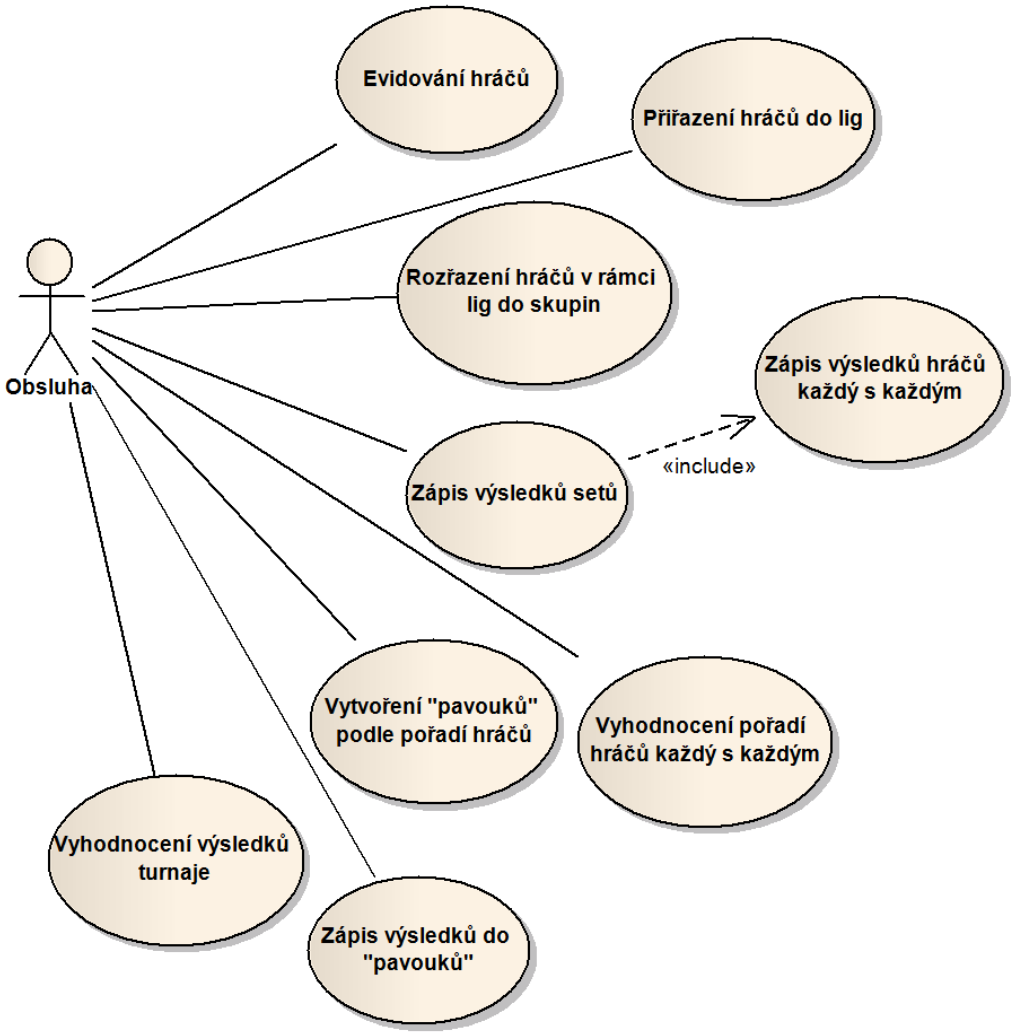
4 Stanovení požadavků na zpracovávaný software

Vhodným nástrojem pro účely organizace a průběhu turnaje se jeví software, který pokryje základní organizační a evidenční činnosti související s turnajem. V této kapitole bude provedena základní analýza požadavků na zpracovávaný software.

4.1 Uživatelské cíle a funkční požadavky

Jediným aktérem, který interaguje se systémem zvenčí, je obsluha.

Požadavky na software jsou znázorněny diagramem případů užití na Obr. 6 a jednotlivé případy užití jsou popsány níže.



Obr. 6 Diagram případů užití aplikace
Zdroj: Vlastní zpracování

UC Evidování hráčů

Evidovat se budou hráči (jméno, příjmení, předešlé pořadí, oddíl a rok narození) v pořadí z předchozího turnaje a nově příchozí podle žebříčku nasazení z předchozích turnajů. Pořadí hráčů bude možné měnit, bude možné specifikovat jejich prezenci na turnaji a sledovat počet zapsaných a příchozích hráčů.

UC Přiřazení hráčů do lig

Jednotliví hráči se přiřadí do lig zpravidla podle jejich schopností a předchozích výsledků. Zastoupení počtu účastníků v ligách bude možné sledovat v závislosti na jejich přítomnosti na turnaji.

UC Rozřazení hráčů v rámci lig do skupin

Hráči se v ligách rozřadí do menších skupin zpravidla sudého počtu dvou až čtyř zip-způsobem (viz kap. 3, odst. 3) podle stanoveného maximálního počtu hráčů ve skupině a obsluze se nabídne možnost toto pořadí změnit. Po odsouhlasení rozřazení hráčů do skupin je pro každou skupinu vytvořena tabulka každý s každým, s názvem složeným z čísla ligy a písmen A až D. Těchto skupin může být i více.

UC Zápis výsledků setů

Pro každý záznam dvou hráčů budou sledovány zápasy neboli sety a počty míčků. Na základě porovnání počtu míčků hráčů v jednotlivých setech se určí výsledné skóre zapsané v tabulce zápasů.

UC Zápis výsledků hráčů každý s každým

Výsledky zápasů vždy dvou hráčů se budou přepisovat do tabulky hráčů každý s každým.

UC Vyhodnocení pořadí hráčů každý s každým

Při přepisu jednotlivých skóre se zároveň budou průběžně sčítat počty míčků a setů. Na základě počtu vyhraných setů se stanoví body. Výsledné pořadí jednotlivých skupin je stanoveno podle počtu získaných bodů hráčů. V případě, že někteří hráči budou mít stejný počet bodů, rozhodují poměry setů a následně poměry míčků.

UC Vytvoření „pavouků“ podle pořadí hráčů

Rozřazení do „pavouků“ je provedeno po vyhodnocení pořadí všech zápasů na základě Tabulky 3 v předchozí kapitole. Na základě seznamů hráčů v tomto pořadí jsou vytvořeny diagramy pro oboustranný vylučovací systém pro 2, 4 nebo 8 hráčů.

UC Zápis výsledků do „pavouků“

Vždy dva hráči se mezi sebou utkají o příslušnou pozici viz Obr. 5. Po zapsání výsledků postupují hráči do dalšího kola. Na tomto základě se stanoví pořadí v rámci jednotlivých diagramů.

UC Vyhodnocení výsledků turnaje

Podle seznamu pořadí hráčů je sestaveno finální pořadí celého turnaje a vytvořen jednodušší duplikát prezenční listiny seřazený podle finálního pořadí.

4.2 Non-funkční požadavky

Výsledný software by měl být zpracován jako desktopová aplikace splňující požadavky průběhu turnaje a jednoduchost uživatelského rozhraní.

Aplikace by měla podporovat načítání a ukládání formátu XLS pro Excel a možnost uložení celého průběhu turnaje pro pozdější využití a kontrolu v libovolném formátu.

5 Výběr vhodných technologií

5.1 Architektura aplikace

Aplikace se skládají zpravidla ze tří vrstev a jejich rozčlenění určuje výslednou architekturu. Tyto vrstvy jsou [6]:

Prezentační vrstva poskytuje uživatelské rozhraní (grafické), poskytuje vstupy a výstupy a slouží pro interakci uživatele se systémem/aplikací.

Aplikační vrstva zajišťuje operace, zpracování vstupně-výstupních požadavků a výpočtů. Zde je implementována logika aplikace.

Datová vrstva slouží k práci s daty, jejich ukládání, načítání a manipulaci s nimi.

Rozdělení podle místa zpracování

Rozdělení architektur podle místa zpracování se dělí následovně [6]:

5.1.1 Centralizovaná architektura

Architektura skládající se ze serveru (centrálního počítače) a terminálu. Server provádí veškeré operace, výpočty a zpracovává výstup pro terminál, zatím co terminál slouží jen pro zobrazení dat uživateli a jejich vkládání.

5.1.2 Architektura klient-server

V tomto případě klientská aplikace zabezpečuje prezentační vrstvu, serverová aplikace slouží jako datová vrstva. Dále se rozlišují tři možnosti, kde bude implementována aplikační vrstva:

- celá na straně klienta
- celá na straně serveru
- část na straně klienta a část na straně serveru

5.1.3 Třívrstvá architektura

V této architektuře je každá z vrstev jako samostatná aplikace a zpravidla i rozmístěna na více počítačích nebo serverech.

5.1.4 Jednovrstvá architektura

V této variantě jsou všechny tři vrstvy svázány a implementovány v rámci jedné aplikace. Toto zpracování je možné rozdělit na tyto dvě skupiny [7]:

- **Monolitická** – výše popsané vrstvy nejsou v aplikaci nijak odděleny a není tedy na první pohled jasné, kterou část aplikace přiřadit jaké vrstvě.
- **MVC (Model View Controller)** – v tomto případě je každé vrstvě přidělena samostatná sekce a aplikace je logicky rozčleněna. Prezentační vrstva je přiřazena pojmu View, aplikační vrstva pojmu Controller a Datová vrstva pojmu Model. Existují i různé mutace, kdy v jednodušších aplikacích dvě vrstvy splynou v jednu.

5.2 Programovací jazyk

Pro výběr požadovaného programovacího jazyka je nutné vzít v potaz, že výsledná aplikace bude provozována na stolním počítači či notebooku. Na základě tohoto předpokladu a požadavku na objektově orientovaný programovací jazyk poskytující GUI (Grafické uživatelské rozhraní) se nabízejí převážně C++, C#, Delphi a Java.

Vzhledem k okrajovým zkušenostem s programovacími jazyky C# a Delphi pro desktop a rozhodnutí podporovat nejen platformu Microsoft Windows, která je domovskou platformou pro tyto dva zmíněné jazyky, zbývají ve výběru pouze Java a C++.

5.2.1 Java

Java je objektově orientovaný programovací jazyk, který se překládá do tzv. byte kódu, který je možné přenášet mezi platformami díky přítomnosti JVM (Java Virtual Machine) – virtuální stroj, který spouští a interpretuje byte kód do srozumitelné podoby dané platformě/OS. Lze v něm psát jak webové, serverové, mobilní, tak běžné konzolové a GUI aplikace.

5.2.2 C++

C++ vychází z jazyka C a je objektově orientovaný programovací jazyk, který se překládá (kompiluje) do nativní podoby jako samostatně spustitelný program interagující přímo s operačním systémem. Výsledná aplikace není kompilovaná za běhu jako Java nebo .NET a díky tomu by měla být rychlejší.

V čistém C++ by bylo velmi obtížné a složité vytvářet grafické uživatelské rozhraní a další specifické funkce pro každý OS zvlášť.

Proto vzniklo několik frameworků a knihoven, které poskytují GUI komponenty a některé v mnoha ohledech a oblastech osvobozují programátora i od konkrétní platformy, respektive operačního systému.

Nejrozšířenějšími GUI knihovnami jsou wxWidgets a GTK. Protože cílem je vyvinout multiplatformní aplikaci, tak tuto možnost nabízí až Qt Framework se svojí filozofií Qt Everywhere. Pro vývoj aplikace byla díky své vzrůstající popularitě a velké podpoře zvolena tato kombinace programovacího jazyka C++ a Qt.

Jeden zdrojový kód napsaný v C++ s využitím Qt Frameworku lze zkompilovat pro tyto platformy:

- Microsoft Windows XP, Vista, 7, 8/8,1 (MinGW nebo MSVC)
- Linux – 32bit a 64bit (Linux, HP-UX, Solaris, AIX, a další)
- Apple Mac OS X – 32bit a 64bit
- Embedded Linux
- Windows CE
- Symbian/S60
- Maemo
- Google Android
- IOS (iPhone, iPod, iPad, iPad 2)

Qt přichází s instalačním balíkem *„all in one“* (pro windows), který obsahuje kompletní prostředí pro vývoj aplikací, jehož součástí je MinGW (GCC), samotný framework, knihovny a Qt Creator, tj. multiplatformní IDE s ladícími nástroji. Pro operační systémy

Linux a Mac OS X jsou dostupné jednotlivé balíčky v systému, také instalátor nebo přímo zdrojové kódy pro kompilaci.

Dobrým zvykem pro základ aplikace je vytvoření souboru *main.cpp*, který obsahuje převážně jen include třídy *QApplication*, hlavičkového souboru hlavní třídy a metodu *main*. V metodě *main* se instanciuje *QApplication*, hlavní třída a zavolá se metoda *exec*, která spouští tzv. Event loop pro danou aplikaci.

Hlavní třída může být nazvaná například *Window* dědící od třídy *QWidget*. V konstruktoru této třídy se nachází vytvoření tlačítka, které zavře aplikaci a vypadá takto:

```
QPushButton *btnClose = new QPushButton("Close");
```

Pro funkci tlačítka je nezbytné přiřadit jeho vyvolávanou akci (signál) *clicked* na spouštěnou akci (slot) takto:

```
connect(btnClose, SIGNAL(clicked()), this, SLOT(close()));
```

A jako poslední krok zbývá zasadit toto tlačítko do layoutu a okna aplikace:

```
QVBoxLayout *layout = new QVBoxLayout;
layout->addWidget(btnClose);
setLayout(layout);
```

Každá aplikace musí mít svůj project file, který lze napsat ručně nebo si jej nechat vygenerovat pomocí příkazu **qmake -project**.

Na výše zmíněných platformách je kompilace jednotných zdrojových kódů aplikace s využitím integrovaného nástroje stejná. Zavoláním příkazu **qmake** se vygeneruje tzv. Makefile sloužící jako „kuchařka“ pro kompilaci aplikace na dané platformě, tento soubor je tedy již platformě závislý. Samotná kompilace proběhne zadáním příkazu **make**. Je také možné zdržet se těchto příkazů a celý tento proces vyvolat stiskem tlačítka kompilace prostřednictvím Qt Creatoru.

6 Zpracování softwarové aplikace

6.1 Východiska aplikace

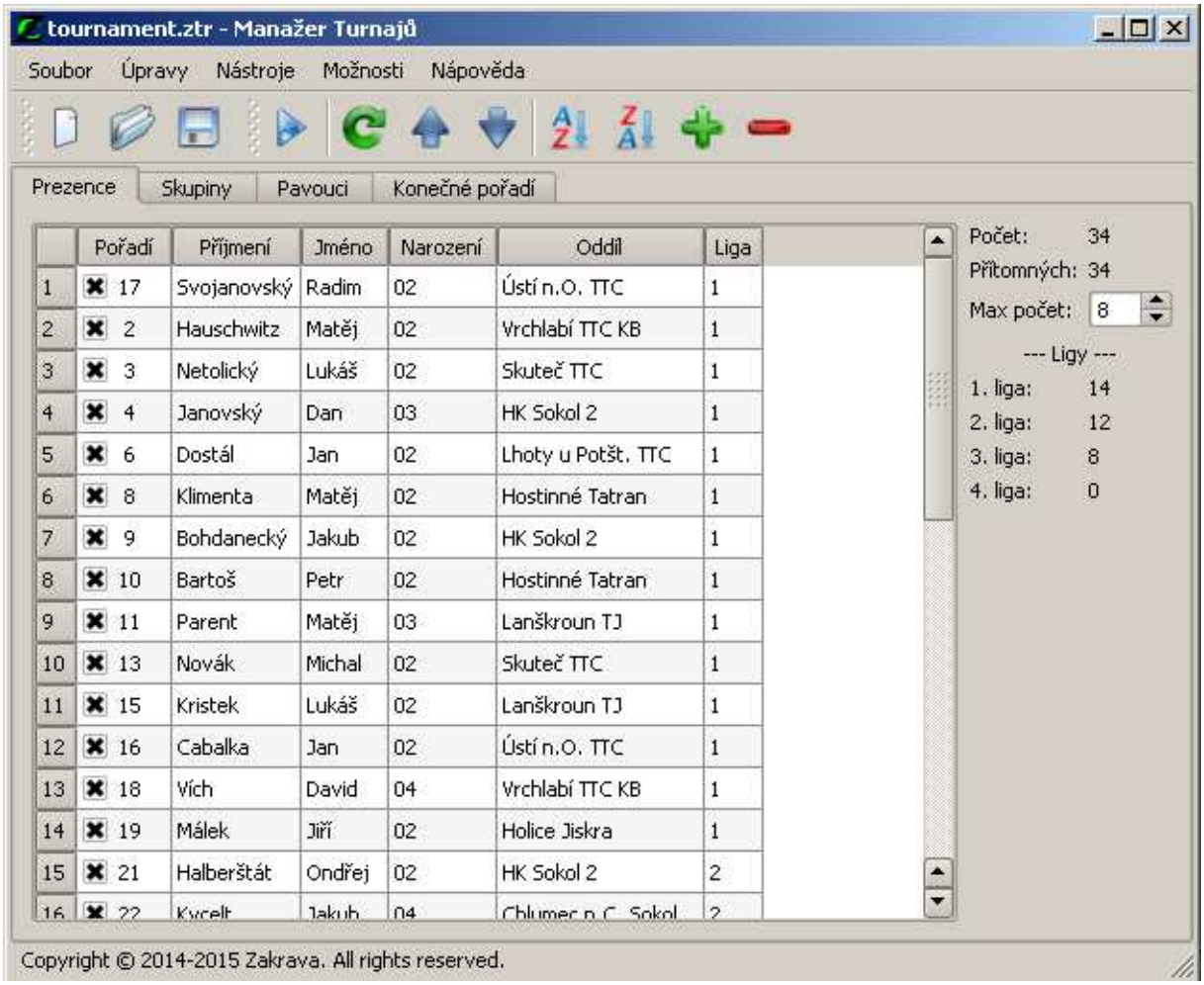
Z předchozích kapitol vyplývá, že výsledná aplikace bude desktopová s využitím programovacího jazyka C++ a Qt Frameworku.
Pro zpracování aplikace byl zvolen agilní způsob vývoje aplikace s iterací cca 14 dní.

6.2 Kostra aplikace

Nejprve je zapotřebí vytvořit základní kostru aplikace. Tu bude reprezentovat třída *ToMa* (Tournament Manager) dědící od třídy *QMainWindow*. Ta byla zvolena vzhledem k podpoře menu, lišty nástrojů a stavového řádku.

Menu je umístěno nahoře a slouží k vyvolání akcí zastupující manipulaci se soubory, jejich otevření a uložení. Dále obsahuje možnosti pro práci s prezenční listinou jako přidávání a mazání řádků, přesouvání pořadí řádků, vzestupného a sestupného abecedního uspořádání podle jednotlivých sloupců a tlačítka pro zahájení turnaje. Doprovodnými funkcemi jsou výběr jazyka aplikace, nastavení kódování souborů a výběr vzhledu aplikace. Pod menu je lišta nástrojů, kde jsou pro jednodušší dostupnost uživateli umístěny nejpoužívanější akce z menu. V hlavní části aplikace jsou záložky reprezentující části průběhu turnaje, které budou popsány dále. Dole je stavová lišta sloužící pro informaci o průběhu akcí a pro stručný popis akcí v menu a v liště nástrojů.

Aplikace je znázorněna na Obr. 7.



	Pořadí	Příjmení	Jméno	Narození	Oddíl	Liga
1	17	Svojanovský	Radim	02	Ústí n.O. TTC	1
2	2	Hauschwitz	Matěj	02	Vrchlabí TTC KB	1
3	3	Netolický	Lukáš	02	Skuteč TTC	1
4	4	Janovský	Dan	03	HK Sokol 2	1
5	6	Dostál	Jan	02	Lhoty u Potšt. TTC	1
6	8	Klimenta	Matěj	02	Hostinné Tatran	1
7	9	Bohdanecký	Jakub	02	HK Sokol 2	1
8	10	Bartoš	Petr	02	Hostinné Tatran	1
9	11	Parent	Matěj	03	Lanškroun TJ	1
10	13	Novák	Michal	02	Skuteč TTC	1
11	15	Kristek	Lukáš	02	Lanškroun TJ	1
12	16	Cabalka	Jan	02	Ústí n.O. TTC	1
13	18	Vích	David	04	Vrchlabí TTC KB	1
14	19	Málek	Jiří	02	Holice Jiskra	1
15	21	Halberštát	Ondřej	02	HK Sokol 2	2
16	22	Kycelt	Jakub	04	Chlumec n.C. Sokol	2

Obr. 7 Aplikace s prezenční listinou
Zdroj: Vlastní zpracování

Vytvoření lišty menu (*QMenu*) a jednotlivých položek se provádí velice podobně jako vytvoření lišty nástrojů (*QToolBar*) následujícím způsobem:

```
QMenu *fileMenu = menuBar()->addMenu(tr("&File"));
fileMenu->addAction(newAct);

fileToolBar = addToolBar(tr("&File"));
fileToolBar->addAction(newAct);
```

Každá položka menu je zde reprezentována tzv. akcí (*QAction*), která specifikuje název, ikonu, popřípadě klávesovou zkratku, která tuto akci spustí. Při spuštění akce je vyvolán signál *triggered* a ten je funkcí *connect* v tomto případě napojen na slot, respektive metodu *newFile* provádějící nastavení aplikace do výchozího stavu.

```
QAction *newAct = new QAction(tr("&New"), this);
newAct->setIcon(QIcon(":/images/new.png"));
newAct->setShortcut(tr("Ctrl+N"));
newAct->setStatusTip(tr("Create a new file"));
connect(newAct, SIGNAL(triggered()), this, SLOT(newFile()));
```

Jednotlivé části turnaje byly pro přehlednost rozčleněny na samostatné widgety (*QWidget*) do záložek pomocí *QTabWidget*, mezi kterými je možné následně přepínat. Procházet jimi směrem kupředu je možné pomocí tlačítka Play, které kontroluje stav a vytváří další kroky.

```
QTabWidget *tbwDock = new QTabWidget(this);
tbwDock->addTab(frmList, tr("&Main")); // Prezenční listina
```

QTabWidget je centrálním prvkem celé aplikace z pohledu *QMainWindow*, což je třeba nastavit voláním metody *setCentralWidget* s parametrem jeho vytvořené instance:

```
setCentralWidget(tbwDock);
```

6.3 Prezenční listina

Na první záložce, viz Obr. 7, se nachází prezenční listina hráčů, jejich jména, pořadí z minulého zápasu, datum narození, oddíl a liga, do které budou hráči zařazeni. Názvy sloupců tvoří seznam překladů hodnot následným voláním metody *tr*:

```
QStringList columnsList;
columnsList << tr("Position") << tr("Lastname") << tr("Firstname")
            << tr("Born") << tr("Club") << tr("League");
```

Tyto údaje reprezentuje tabulka zobrazená pomocí *QTableView* ve spolupráci s modelem dat *QStandardItemModel*.

Instance modelu je vytvořena pomocí konstruktoru s parametry počtu řádků, sloupců a rodiče a nastavení názvu sloupců. Instanci třídy *QTableView* je nastaven vytvořený model a chování následujícím způsobem:

```
QStandardItemModel *model = new QStandardItemModel(0, columnsList.coun
t(), this);
for(int column = 0; column < columnsList.count(); column++)
 model->setHeaderData(column, Qt::Horizontal, columnsList.at(column));
QTableView *tbvList = new QTableView(this);
```

```
tbvList->setModel(model);
tbvList->setShowGrid(true);
tbvList->setAlternatingRowColors(true);
tbvList->setSelectionMode(QAbstractItemView::SingleSelection);
tbvList-setSelectionBehavior(QAbstractItemView::SelectRows);
```

Aby bylo možné specifikovat, který z hráčů se dostavil, při vytváření nového řádku modelu se nastaví možnost zaškrtávacího políčka:

```
model->insertRow(model->rowCount());
QStandardItem *item = new QStandardItem();
item->setCheckable(true);
model->setItem(model->rowCount() - 1, 0, item);
```

Z důvodu přehlednosti je v této záložce doplněna statistika celkového počtu účastníku, počtu přítomných a počtu hráčů v jednotlivých ligách. Zobrazení probíhá přes *QLabel* s popisy a těmito počty organizované v *QFormLayout*. Přepočet je spouštěn změnou v modelu příslušného sloupce.

Výpočet přítomných hráčů se provádí podle stavu zaškrtávacího políčka:

```
int presented = 0;
for(int row = 0; row < model->rowCount(); row++) {
  if(model->item(row, 0)->checkState())
    presented++;
}
```

Následuje vypočítání zastoupení hráčů v jednotlivých ligách a jejich stanovení. Nejprve se projde celý sloupec vzhledem k zastoupeným ligám, které se uloží do seznamu a seřadí pomocí vestavěné metody *qSort*. Následně se pro každou ligu vypočítá zastoupení hráčů přes nastavený filtr pomocí *QSortFilterProxyModel*.

```
QSortFilterProxyModel *modelProxy = new QSortFilterProxyModel();
modelProxy->setSourceModel(model);
modelProxy->setSortLocaleAware(true);
```

Metoda nastavující filtr pro požadovanou ligu je následující:

```
setLeagueProxy(int league)
{
  modelProxy->setFilterKeyColumn(5);
  modelProxy->setFilterRegExp(QString("%1").arg(league));
}
```

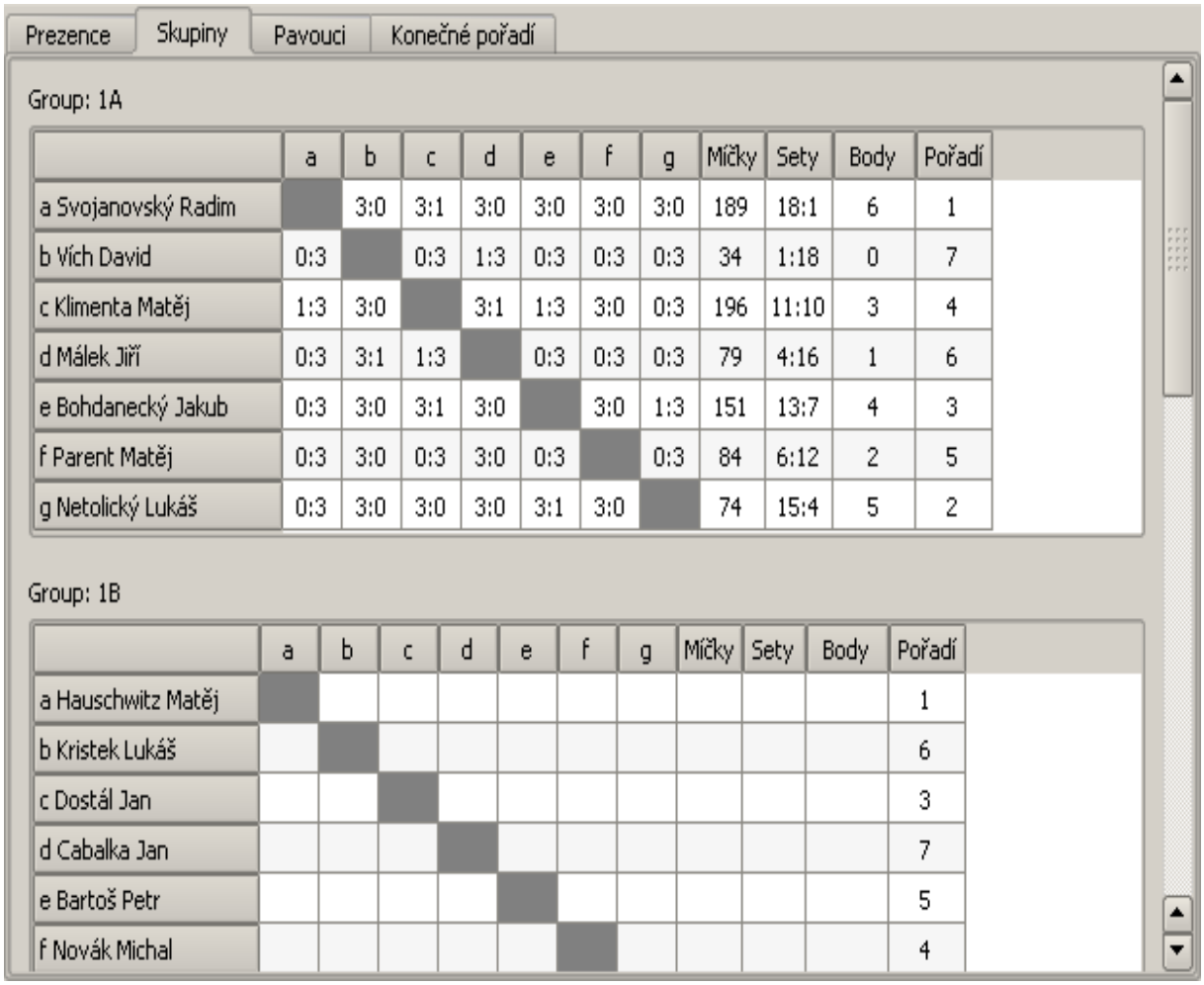
Následně se v rámci jednotlivých lig rovnoměrně zip-způsobem rozdělí hráči do skupin s maximálním počtem hráčů nastaveným pomocí *QSpinBox*. Pro každou skupinu je vytvořen model a pohled na tabulku bez možnosti editace umožňující Drag&Drop funkcionalitu, která se nastavuje takto:

```
tbvTeam->setEditTriggers(QAbstractItemView::NoEditTriggers);
tbvTeam->setAcceptDrops(true);
tbvTeam->setDefaultDropAction(Qt::MoveAction);
tbvTeam->setDragDropMode(QAbstractItemView::DragDrop);
tbvTeam->setDragDropOverwriteMode(false);
tbvTeam->setDragEnabled(true);
tbvTeam->setDropIndicatorShown(true);
```

Aby toto rozdělení do skupin bylo možné ovlivnit, či upravit obsluhou, zobrazí se tyto tabulky modálně v *QDialog* a po odsouhlasení se přejde na další záložku Skupiny.

6.4 Skupiny hráčů

Záložka skupin obsahuje tabulky podle dříve specifikovaného rozčlenění hráčů a hraje se tzv. každý s každým v rámci skupin, viz Obr. 8. Tyto tabulky, respektive modely, mají dvě části.



Obr. 8 Záložka skupin aplikace
Zdroj: Vlastní zpracování

První část obsahuje výsledky souboje vždy dvou hráčů mezi sebou nad hlavní diagonálou a zrcadlově obrácené výsledky pod ní. Jelikož se hraje na tři vítězné sety, není pro informace průběhu souboje dvou hráčů v jedné buňce tabulky místo.

Druhá část tedy sestává z vytvoření tzv. delegáta implementovaného třídou *ZTeamMatchesDelegate*, která dědí od třídy *QItemDelegate.* Instance delegáta je pak přiřazena každé buňce tabulky vyjímaje části pro vyhodnocení výsledků.

```
tbvTeams[team]->setItemDelegateForColumn(row,
 new ZteamMatchesDelegate(this, team));
```

Třída *ZTeamMatchesDelegate* implementuje chování dané buňky při její editaci. V tomto případě je vytvořena instance *QDialog* zobrazující jednoduchou tabulku dvou hráčů, kam

se zaznamenávají počty míčků. Při změně hodnoty se ověří minimální počet míčků, což je 9, a spočítá se jejich rozdíl v každém setu, který musí být alespoň dva a podle toho se připočte vítězství jednomu nebo druhému hráči. Při zavření delegáta se zavolá metoda *setModelData*, v níž se nastaví výsledek X:Y do rodičovského modelu na daném indexu.

Hlavička metody:

```
setModelData(QWidget *editor, QAbstractItemModel *model,
             const QModelIndex &index) const
```

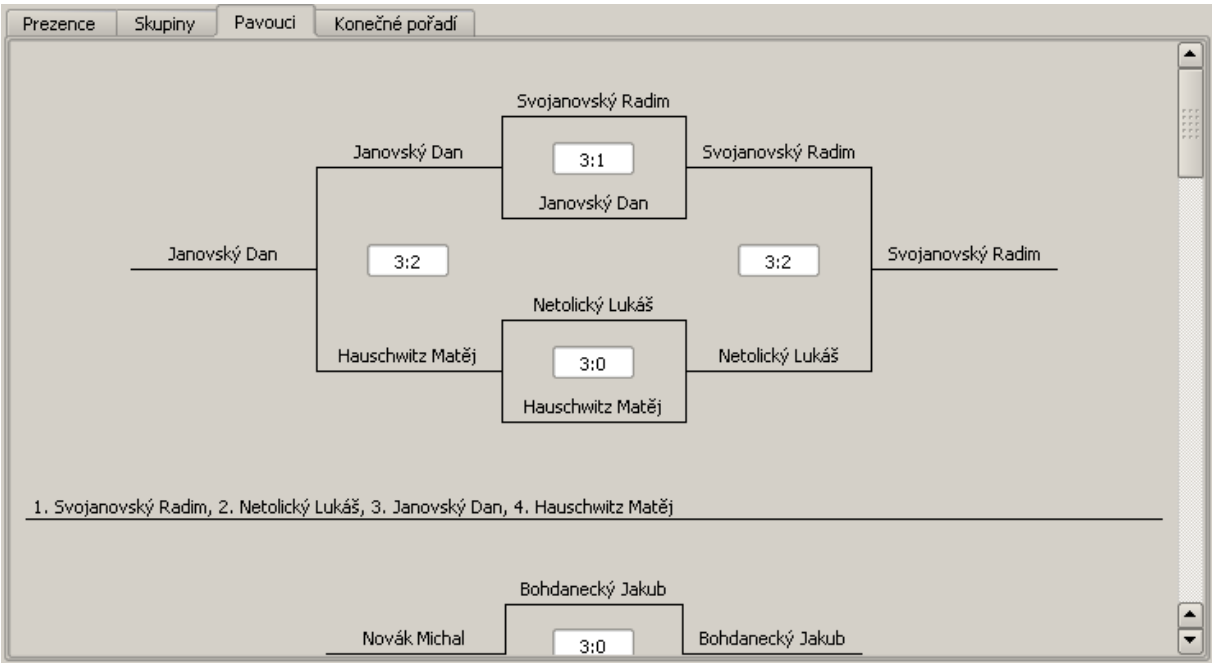
Tělo metody:

```
QAbstractItemModel *modit = tbvTeamMatches->model();
QModelIndex indexInv = model->index(index.column(), index.row());
QString setsA = modit->data(modit->index(0, columnCount - 1),
     Qt::DisplayRole).toString();
QString setsB = modit->data(modit->index(1, columnCount - 1),
     Qt::DisplayRole).toString();
model->setData(index, setsA + ":" + setsB, Qt::EditRole);
model->setData(indexInv, setsB + ":" + setsA, Qt::EditRole);
```

Každou změnou modelu se přepočítává počet míčků, poměr vyhraných setů ku prohraným a počet bodů vypočítaný podle počtu výher hráče. Jakmile jsou všechna políčka vyplněna, provede se seřazení podle počtu míčků, poměru setů a bodů a na tomto základě se stanoví výsledné pořadí hráčů pro vstup do dalšího kola.

6.5 Pavouk s hráči

V následujícím kole se rozřadí hráči do pavouků podle týmů a pořadí z předchozího kola tak, že první z liché skupiny a druhý ze sudé skupiny bojují proti sobě a naopak druhý z liché skupiny a první ze sudé skupiny bojují proti sobě a jsou umístěni do prvního pavouka proti sobě a tito hráči se utkají o 1. až 4. místo. A takto to pokračuje dále.
Ukázka diagramů s výsledky z aplikace viz Obr. 9.



Obr. 9 Diagramy vylučovacího systému hráčů
Zdroj: Vlastní zpracování

Každý pavouk je reprezentován třídou *ZSpiderWidget* a jako parametr konstruktoru přijímá *QStringList* se jmény v pořadí z předešlého kroku a pomocí volání metody *setPosition* je nastaveno číslo prvotní pozice.
Všechny instance jsou uchovány v dynamickém poli QVector:

```
QVector<ZSpiderWidget*> wdtSpiders;
wdtSpiders << new ZSpiderWidget(playersList);
wdtSpiders[pos]->setPosition(place);
```

ZSpiderWidget používá metodu *paintEvent(QPaintEvent *e)* pro nakreslení čar, jmen hráčů a rozmístění editačních polí *QLineEdit* pro vložení skóre mezi právě dvěma hráči.

Instanciace a inicializace třídy *QLineEdit* je nastavení vstupní masky do tvaru skóre X:Y, pojmenování objektu a navázání signálu editace dokončena vypadá následovně:

```
for(int i = 0; i < numberOfMatches; i++) {
  ledMatchSet << new QLineEdit(this);
  ledMatchSet[i]->setAlignment(Qt::AlignCenter);
  ledMatchSet[i]->resize(50, 20);
  ledMatchSet[i]->setObjectName(QString("%1").arg(i));
  ledMatchSet[i]->setInputMask("9:9");
```

```
    connect(ledMatchSet[i], SIGNAL(editingFinished()),
        this, SLOT(editingFinished()));
}
```

V metodě, respektive slotu *editingFinished*, dojde k přepočítání pořadí na základě zadaného výsledku zápasu mezi dvěma hráči a určení vítěze. Poté, jestliže jsou všechna editační pole vyplněna, se stanoví výsledné pořadí v rámci daného pavouka a za tohoto předpokladu se vyšle signál se seznamem hráčů seřazených podle pořadí.
Po přijetí signálů ze všech pavouků se podle pořadí jednotlivých hráčů a prezenční listiny sestaví tabulka finálního pořadí celého turnaje.

6.6 Práce se soubory

Aplikace podporuje tři typy formátu souborů pro čtení a zápis:

- CSV (Comma Separated Values) – uchovává pouze informace z prezenční listiny a výsledného pořadí hráčů,
- XLS (MS Excel) – uchovává pouze informace z prezenční listiny a finálního pořadí hráčů,
- ZTR (Zakrava Tournament Record) – uchovává informace z celého průběhu turnaje.

CSV je textový soubor oddělující hodnoty zpravidla čárkou nebo středníkem. Číst a zapisovat je možné s využitím kombinace otevřeného popisovače souboru *QFile* a proudem textu *QTextStream*.

XLS je binární soubor z dílny Microsoft Office pro prezentaci tabulek, není tedy možné s ním zacházet jako s textovým souborem, a proto byla využita pro práci s ním externí knihovna *BasicExcel* [9], která zaštiťuje práci s buňkami dokumentu.

ZTR je speciálně vytvořený formát pro účely zaznamenání veškerých dat z průběhu turnaje. Využívá třídy *QSettings* s výstupem podobným *ini* souboru se sekcemi.

```
QSettings settings(fileName, QSettings::IniFormat);
```

Načtení jedné proměnné typu *int* s výchozí hodnotou vypadá takto:

```
splitTeamMax = settings.value("splitTeamMax", splitTeamMax).toInt();
```

Načítání dvourozměrných hodnot zaštiťují metody *begitReadArray*, *setArrayIndex* a *endArray*, například načtení a otevření prezenční listiny ze sekce attendanceList:

```
int size = settings.beginReadArray("attendanceList");
model->setRowCount(0);
model->setRowCount(size);
for(int row = 0; row < size; row++) {
 settings.setArrayIndex(row);
 setCheckable(row);
 model->item(row, 0)->setCheckState(static_cast<Qt::CheckState>
     (settings.value("present").toUInt()));
 for(int column = 0; column < columnsList.count(); column++) {
  model->setData(model->index(row, column, QmodelIndex()),
     settings.value(columnsList[column]).toString());
 }
}
settings.endArray();
```

6.7 Automatické aktualizace

Aplikace také obsahuje možnost automatické aktualizace. Z webového serveru se stáhne textový soubor popisující dostupnou verzi a porovná se s aktuální verzí aplikace. Různí-li se tyto verze, proběhne stažení souboru aplikace a dojde k náhradě aplikace na straně počítače.

Inicializace probíhá přes *QNetworkAccessManager* pomocí instanciace *QNetworkRequest* s parametrem *QUrl* a zápisem do souboru přes *QFile*:

```
QUrl url(downloadPath + name);
QFile *file new QFile(name);
QNetworkAccessManager manager;
QNetworkReply *reply;
reply = manager.get(QNetworkRequest(url));
file->open(QIODevice::WriteOnly)
file->write(reply->readAll());
file->close();
```

6.8 Resources

Pro lepší přehlednost byly akcím přiřazeny obrázky, které je potřeba začlenit do aplikace. K tomu slouží soubor s příponou *qrc*. Jedná se o seznam zdrojů ve formátu XML s notací RCC. Tento soubor má následující strukturu:

```
<!DOCTYPE RCC><RCC version="1.0">
<qresource>
    <file>images/new.png</file>
    <file>images/open.png</file>
    <file>images/save.png</file>
</qresource>
</RCC>
```

Definice ikony spustitelné aplikace se vytváří ve speciálním souboru definice zdrojů s příponou *rc*.

```
IDI_ICON1 ICON DISCARDABLE "images/myapp.ico"
```

Tyto soubory je ještě třeba definovat v souboru projektu takto:

```
RESOURCES += toma.qrc
RC_FILE = toma.rc
```

Všechny soubory ať obrázky nebo jakékoliv statické soubory či read-only dokumenty lze začlenit do aplikace. Tento proces je dále automatický a odehrává se v rámci kompilace.

6.9 Překlady jazyků

Za povšimnutí stojí, že všechny zobrazované názvy byly obaleny voláním metody *tr*, která pomáhá s překladem aplikace do libovolných jazyků.

Každý jazyk má svůj vlastní soubor, který je třeba specifikovat v souboru projektu takto:

```
TRANSLATIONS = toma_en.ts toma_cs.ts
```

Takto definované soubory překladu s příponou ts se vytvoří spuštěním příkazu:

```
lupdate toma.pro
```

Tyto vytvořené soubory mají XML syntax a lze je otevřít aplikací linguist. Jedná se o nástroj umožnující snazší překlad textů aplikace do různých jazyků. Po skončení editace zdrojových textů jazyků je zapotřebí zkompilovat překlady do formátu *qm* takto:

```
lrelease toma.pro
```

Formátu *qm* pak rozumí třída *QTranslator*, která se stará o překlad textů v rámci aplikace. A jeho použití je následující:

```
QTranslator translator;
QApplication::installTranslator(&translator);
translator.load(":/toma_cs.gm");
```

Aby se staly tyto překlady součástí aplikace, používá se opět soubor se seznamem zdrojů a přidají se jako další záznamy:

```
<file>toma_cs.qm</file>
<file>toma_en.qm</file>
```

Zavoláním metody *load* s parametrem příslušného souboru překladu na instanci *QTranslator* a znovunačtení textů za chodu nebo při opětovném spuštění aplikace se zobrazí texty ve zvoleném jazyce.

6.10 Styly uživatelského rozhraní

Qt Framework poskytuje několik multiplatformních stylů, které zaručí, že aplikace bude vypadat na všech platformách stejně, kromě dekorace oken. Na každé platformě je dále k dispozici nativní styl charakteristický pro dané prostředí.

Mezi styly frameworku, které by měly být dostupné ve všech prostředích, patří zejména CDE, Motif, Plastique a Cleanlooks.

Například pro zavedení stylu Plastique jsou dvě možnosti:

```
QApplication::setStyle(new QPlastiqueStyle);
Qapplication::setStyle(QStyleFactory::create("Plastique"));
```

První možnost je přímo nastavení stylu vytvořením instance ze třídy a druhá je využít třídu *QStyleFactory*, která jako parametr metody *create* připouští název stylu.

7 Shrnutí dosažených výsledků

Propozice turnaje popsané ve 3. kapitole byly implementovány do vyvíjené aplikace. Jedná se o jediný známý program, který kopíruje tento specifický průběh turnaje ve stolním tenisu, a poskytuje tak nástroj pro organizaci turnaje bez nutnosti zaznamenávat vše na papír, ručního počítání výsledků a pořadí hráčů.

Aplikace pokrývá všechny analyzované kroky a funkční požadavky znázorněné diagramem případů užití ve 4. kapitole.
Zakomponovány byly i non-funkční požadavky s využitím knihovny pro manipulaci se soubory XLS, které slouží pro načítání a ukládání podkladů pro prezenční listinu a výsledky turnaje. Pro uchování prezence, výsledků a veškerých údajů v průběhu turnaje byla zvolena forma uložení do souboru.

Aplikace byla bez problémů zkompilována a otestována na distribuci Gentoo a Ubuntu operačního systému Linux a na operačním systému Windows XP, 7. Aplikace se na všech testovaných platformách chová stejně. Vzhledem k použitému předdefinovanému vzhledu a stylu vypadá aplikace na všech platformách jednotně.

Aplikaci je možné dále upravit nebo rozšířit. Přidat lze například další zdroje či výstupy dat nebo zvýšit počet účastníků ve skupinách. To by ovšem znamenalo rozšířit možnosti třídy pro vykreslování „pavouka“, který je implementován pro maximálně 8 hráčů najednou.

8 Závěr

Zvolený agilní přístup vývoje aplikace byl přínosný s ohledem na skutečnost, že zadání jednotlivých kroků nebylo předem známo a byl přínosný při organizaci práce a vývoji aplikace.

Vybraný programovací jazyk C++ v kombinaci s Qt Frameworkem poskytl produktivní prostředí a nástroje pro zpracování aplikace. Moduly a třídy frameworku pokrývají širokou škálu možností a poskytují velice dobrou podporu pro programování multiplatformních aplikací. Výsledná aplikace splňuje požadavky a kopíruje analyzovaný průběh turnaje ve stolním tenisu.

Dalším vývojem či rozšířením aplikace by mohla být podpora standardního průběhu turnaje a možnost přepínání jeho průběhu. Aplikace by mohla být rozšířena o interakci s webovým serverem prostřednictvím Web API nebo s databázovým serverem přes přímé spojení. Toto rozšíření by mohlo zajišťovat načítání hráčů do prezenční listiny a publikaci výsledků.

Výhodou zvoleného řešení je nezávislost programu na okolí. Nutnou a postačující podmínkou je počítač. Není zapotřebí připojení k internetu ani žádné nainstalované dodatečné softwarové vybavení. Na druhou stranu je nevýhodou, že nestačí jen zařízení s prohlížečem a přístupem k internetu, kde by se všechny údaje ukládaly přímo.

Výhodou samotného programu je, že vede obsluhu postupně celým turnajem krok za krokem a eliminuje chybu lidského faktoru.

9 Seznam použité literatury

[1] ARLOW, Jim a Ila NEUSTADT. UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky. Vyd. 1. Překlad Bogdan Kiszka. Brno: Computer Press, 2007, 567 s. ISBN 978-80-251-1503-9.

[2] TEIN, René. Návrh aplikací v jazyce UML - Unified Modeling Language. In: interval.cz [online]. 2003 [cit. 2014-08-21]. Dostupné z: http://interval.cz/clanky/navrh-aplikaci-v-jazyce-uml-unified-modeling-language/.

[3] ROYCE, Winston. Managing the Development of Large Software Systems. In: cs.umd.edu [online]. 1970 [cit. 2014-08-21]. Dostupné z: http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf.

[4] KNESL, Jiří. Agilní vývoj: Úvod. In: zdroják.cz [online]. 2009 [cit. 2014-08-22]. Dostupné z: http://www.zdrojak.cz/clanky/agilni-vyvoj-uvod.

[5] Ing. ŠANC, Přemysl. Informace č.5/Otevřené BT NEJML. ŽACTVA 2012/13 [listina]. 5.3.2013 [cit. 2014-10-15].

[6] Třívrstvá architektura. In: managementmania.com [online]. 2013 [cit. 2014-10-15]. Dostupné z: https://managementmania.com/cs/trivrstva-architektura-three-tier-architecture.

[7] Ing. Mlejnek, Jiří. Před. 5 – Architektura softwarových systémů. In: edux.fit.cvut.cz [online]. FIT ČVUT 2011 [cit. 2015-04-17]. Dostupné z: https://edux.fit.cvut.cz/oppa/BI-SI1/prednasky/BI-SI1-P05m.pdf.

[8] Qt 4.8 All Classes. In: doc.qt.io [online]. 2015 [cit. 2014-10-15]. Dostupné z: http://doc.qt.io/qt-4.8/classes.html.

[9] BasicExcel - A Class to Read and Write to Microsoft Excel. In: codeproject.com [online]. 2015 [cit. 2014-10-15]. Dostupné z: http://www.codeproject.com/Articles/13852/BasicExcel-A-Class-to-Read-and-Write-to-Microsoft.

[10] Qt Documentation. In: doc.qt.io [online]. 2015 [cit. 2015-04-08]. Dostupné z: http://doc.qt.io/qt-4.8/index.html

[11] Qt Examples. In: doc.qt.io [online]. 2015 [cit. 2015-04-08]. Dostupné z: http://doc.qt.io/qt-4.8/all-examples.html

[12] Qt Tutorials. In: doc.qt.io [online]. 2015 [cit. 2015-04-08]. Dostupné z: http://doc.qt.io/qt-4.8/tutorials.html

[13] Qt4 Tutorial. In: zetcode.com [online]. 2015 [cit. 2015-04-08]. Dostupné z: http://zetcode.com/gui/qt4.

10 Přílohy

Oskenované zadání práce

Zdrojové kódy a spustitelná aplikace

FIM UHK
UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Rokitanského 62, 500 03 Hradec Králové, tel: 493 331 111, fax: 493 332 235

Zadání k závěrečné práci

Jméno a příjmení studenta: **Tomáš Zákravský**

Obor studia: Aplikovaná informatika
Jméno a příjmení vedoucího práce: **Tomáš Kozel**

Název práce:
Softwarová aplikace pro turnaj ve stolním tenisu

Název práce v AJ:
Software application for table tennis tournament

Podtitul práce:

Podtitul práce v AJ:

Cíl práce: Cílem práce je vytvoření podpůrné aplikace pro organizaci průběhu, zpracování a zveřejnění výsledků turnaje.

Osnova práce:
1. Úvod
2. Obecné zásady návrhu a zpracování aplikace
3. Analýza propozic a vlastního průběhu turnaje
4. Stanovení požadavků na zpracovávaný software
5. Výběr vhodných technologií
6. Zpracování softwarové aplikace
7. Závěr

Projednáno dne: 15.10.14

Podpis studenta Zákravský

Podpis vedoucího práce