Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Diplomová práce

Projekt implementace a provozu webové hry Space Traffic

Plzeň, 2012 Bc. Richard Kocman

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 17. května 2012

Richard Kocman

Abstract

The subject of this diploma thesis is the students' web game development project Space Traffic. After completion the game will be used for promotion of the Department of Computer Science and Engineering of the University of West Bohemia among students of high schools. The motive of this project consists in an effort of providing a comprehensive view at the project progress in the course of three years, since its origins until the present time. In this essey we also summarize and analyse the acquired experience of the students, the software development methodology and the information on the IBM Jazz platform and its tools. We conclude that a students' software project of a greater extent is feasible, although it has many specifics originating from the academic background. The development is realized slowly, because its part is also presented by the process of gaining and sharing of experience among the students participating on the project.

Poděkování

Velice děkuji Ing. Přemyslu Bradovi, MSc., Ph.D. za cennou, ochotnou a trpělivou pomoc při psaní této práce a vedení projektu Space Traffic a všem studentům, kteří se na projektu podíleli a pomohli jeho vývoj posunout dále.

Zvláštní poděkování za podporu a umožnění studia na vysoké škole patří rodině Richarda Kocmana a jeho snoubence, která pro něj byla velkou oporou.

Obsah

1 Úvod

Než se budeme věnovat projektu Space Traffic, měli bychom nejprve říci, jak lze obecně definovat jakýkoli projekt: *„Časově omezené úsilí vynaložené na vytvoření unikátního produktu, služby či výstupu."* [RPIT/11]. Především nás budou zajímat takové, u kterých je výstupem software (počítačový program). Některé projekty mohou být malé s nevelkým počtem lidí, jiné naopak velké s několika týmy. Z časového hlediska trvají den, ale i roky. Space Traffic (dále ST) je softwarovým projektem vývoje webové hry.

Jeho výstupem má být webová hra pro studenty středních škol a propagace studia na katedře informatiky Západočeské univerzity pomocí herní komunity, která se okolo webových her vytváří. Předpokládáme, že studenti, kteří hru začnou hrát, ji budou popularizovat mezi svými vrstevníky, čímž se informace o hře i o katedře informatiky rozšíří.

Do projektu ST se na katedře informatiky Západočeské univerzity (dále jen KIV a ZČU) zapojuje stále více studentů, jejichž práce přináší celou řadu hmatatelných výsledků. Nyní se seznamme s jeho průběhem od počátku do současnosti z pohledu informatika a dlouholetého člena týmu.

1.1 O projektu webové hry Space Traffic

Studentský projekt Space Traffic byl zahájen v akademickém roce 2009-2010 pod záštitou Ing. MSc. Přemysla Brady Ph.D. a vedením studenta Ing. Zbyňka Neuderta. Pokračuje na KIV i v roce 2011-2012 a i v budoucnu lze očekávat jeho další vývoj. Zaměřuje se na tvorbu webové hry z prostředí vesmíru. Hlavním úkolem hráče je budování úspěšné společnosti zabývající se mezihvězdným obchodováním. Tohoto cíle bude možné dosáhnout pomocí flotil dopravních lodí efektivně řízených pomocí programovatelných instrukcí. Hráč se při hraní ST seznámí i s tématy informatiky. Výsledná hra tedy nemusí zaujmout středoškolské studenty pouze svou hratelností, ale část z nich může upoutat informatikou jako takovou. Takto oslovení žáci jsou chápáni jako potenciální skupina zájemců o studium na KIV ZČU.

Další záměr spočívá ve vývoji webové hry samotné. Pokud na něj nahlížíme jako na softwarový proces a uspořádanou množinu postupů vedoucí k vytvoření nového softwarového díla; hovoříme o činnosti zvané softwarové inženýrství. A tato činnost je i stěžejním oborem KIV. Diplomová práce hovoří i o zapojování již stávajících studentů katedry do projektu a o jejich možnostech získat tím zkušenosti a praktické dovednosti.

1.2 Cíl práce

Následující pohledy na projekt ST tvoří hlavní osnovu diplomové práce a kopírují její zadání. Čtenáři umožňují vytvořit si jasnou představu o tom, co si pod pojmem studentský projekt ST představit, jak vypadala a vypadá jeho realizace, jakým způsobem se měnily vlastnosti výsledného produktu webové hry a jak je můžeme hodnotit.

V teoretické části práce se budeme věnovat problematice metodik vývoje softwaru. Představíme základní modely k vedení softwarového projektu i konkrétní metodiky. To nám umožní nahlédnout do softwarového inženýrství, abychom se v následujících kapitolách lépe orientovali. Dále se dozvíme, na jaká rizika se u studentského projektu soustředit, s čím počítat a co neopomenout, pokud uvažujeme o účasti v ST nebo nás tématika studentských projektů blíže zajímá. Tyto znalosti nejsou získány pouze studiem teoretických materiálů, ale vycházejí i ze zkušeností autora práce, který v projektu ST po dobu tří let působil. Zmíníme se i o platformě IBM Jazz a ukážeme si, čeho bylo z její nabídky použito.

1.3 Student a studentský projekt

Student je jako lidský zdroj pro softwarový vývoj velmi obtížně uchopitelný. Mívá celou řadu aktivit, velice proměnlivý časový plán a pracovní zápal i výkon, který když upadá, není snadné opětovně povzbudit. Pokud absolvent KIV a ZČU po studiu nastupuje do vývojářské společnosti, bývá zpravidla zařazen do rozběhnutého soukolí a pracovních postupů konkrétní společnosti. Jeho úkolem je především začlenění se do kolektivu a přizpůsobení se normám, rytmům a mechanismům, které v něm platí. Absolvent přichází do kontaktu s kolegy, z nichž mnozí jsou již velmi zkušení, softwarového projektů a metodiky jeho vývoje velice dobře znalí, a zároveň dostatečně rychle schopni nasměrovat tápajícího tou správnou cestou. Nováček tedy nejen že odvádí pracovní výkon, ale neustále získává praktické zkušenosti s metodikou vývoje softwaru.

Projekt ST neumožní studentovi získat takový rozsah znalostí jako získá absolvent během své praxe, ale může se v mnohém tomuto ideálu přiblížit. Teprve se seznamuje s metodologií vývoje softwaru, a to často jen studiem její terminologie. Získává informace o různých metodikách užívaných v softwarovém inženýrství a také o procesech nebo vlastnostech a parametrech, kterými se navzájem liší. I když nabyde tyto teoretické znalosti, teprve čeká na šanci uplatnit je v praxi. Příprava umělého cvičného projektu je často neefektivní a zapojení do projektů komerčních firem nebo projektů katedry nemusí být ideální. Úsilí, které garant předmětu vynakládá k nasmlouvání a organizaci praxe pro několik desítek studentů, je veliké. Těm navíc často chybí specifické znalosti pro pozici např. projekt manažera či softwarového architekta, protože řídit jen malý tým pracující na jednom konkrétním problému po dobu jednoho semestru nemusí být pro tyto pozice tak přínosné.

ST může představovat zajímavou alternativu, do níž posluchače KIV pozvolna začleňujeme během jejich studijních let. Lidé, kteří projekt vedli či v něm i jinak úspěšně působili, zastávají postupem času významnější pozice, ale v každém případě z univerzity po ukončení studia odejdou a uvolní místo dalším. Noví studenti musí pracovní postupy týkající se ST

v lepším případě opět uvést do chodu, v tom horším znovu stanovit. I vedoucí projektu jsou z řad studentů. Volí metodiku (seznamují se s ní), aplikují ji, zkouší (nebo si osvojují práci se stávající), a dle ní i projekt dále řídí. Parametry ST se navíc časem mohou zásadně měnit a předchozí postupy se stát zcela nevhodnými.

Pozice, v níž se studenti nacházejí, není jednoduchá sama o sobě, ale ještě náročnější je, jedná - li se o jejich první kroky. Ti, kteří dokáží i bez předchozí přípravy nárokům ST vyhovět, získají velmi cennou zkušenost. A protože se nacházíme v akademickém prostředí, kde by jednou z nejvyšších priorit mělo být právě získání znalostí, a nikoli pouze úspěšné dovedení vývoje ST k dalším cílům, získají i ti, kteří jim plně nevyhoví, ale k jejich plnění se přibližují.

Tato práce zmiňuje celou řadu osvědčených pracovních postupů a specifikuje je v kontextu skutečných situací, ve kterých byly zachyceny a aplikovány. Snaží se samozřejmě zachytit i problémy, se kterými se studenti potýkali, a neúspěchy vzniklé špatnými postupy. Dále si klade za cíl být studentům dobrým průvodcem při prvních krocích v rámci ST.

2 Metodiky vývoje softwaru

V této kapitole se seznámíme s tím, co jsou metodiky (pracovní postupy) vývoje softwaru, a jak nám pomáhají při jeho tvorbě. Nejprve se seznámíme s těmi nejznámějšími, a poté zmíníme i několika představitelů tzv. agilních metodik.

Co je metodika? Jedná se o soubor postupů a návodů pro vývoj softwarové aplikace. Zahrnuje jednotlivé fáze životního cyklu vývoje softwaru. Řeší otázky proč, kdo, kdy a co a většinou se nezabývá podrobnějšími otázkami jako např. zda použít strukturální nebo objektový přístup a podobně.

Než začaly jednotlivé metodiky vznikat, byl software vyvíjen modelem ve stylu „napiš a oprav". Vývoj aplikace probíhal tak, že se rovnou provedla implementace, aplikace byla dodána a následovalo opravování chyb. Takový postup způsoboval mnoho problémů, a tak byl v roce 1957 definován Stagewise model životního cyklu, který byl založený na striktní posloupnosti fází. V roce 1970 přichází Vodopádový model, kterým se budeme více zabývat.

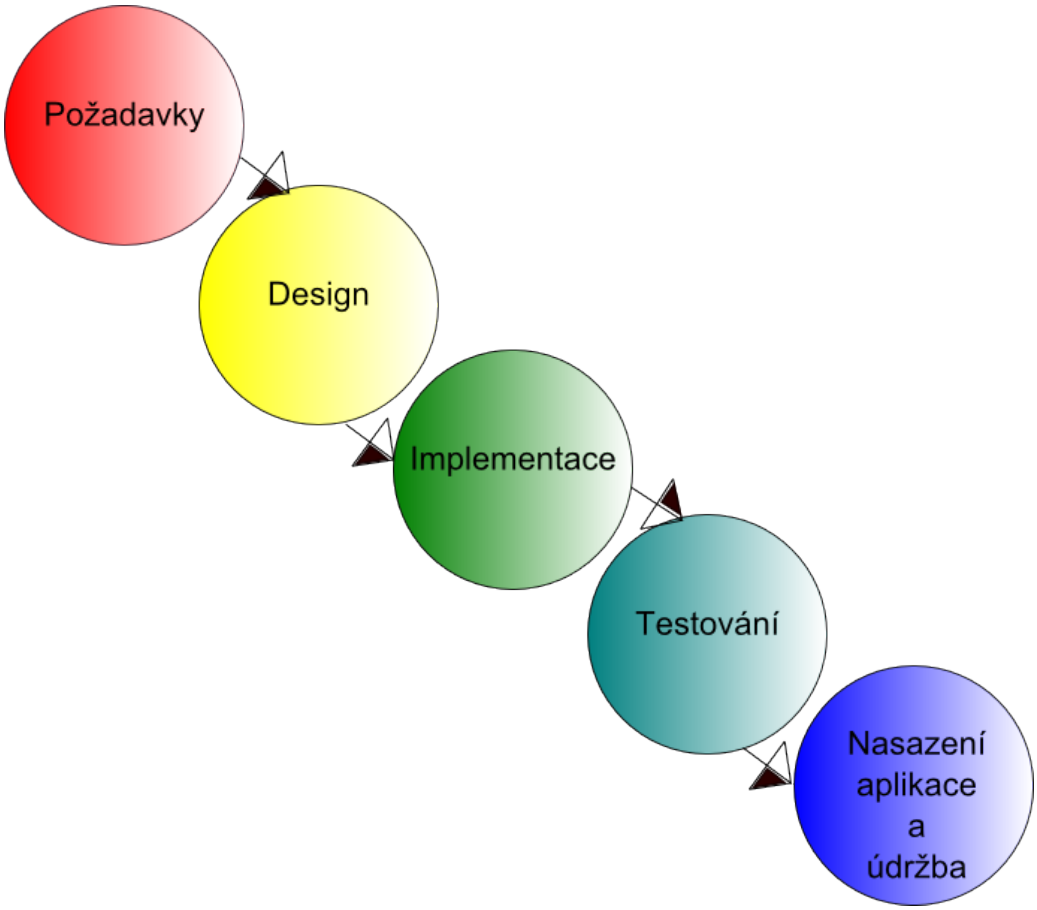
2.1 Vodopádový model

Za jednu z nejznámějších považujeme metodiku zvanou vodopád, vodopádový model (z angl. waterfall). Zakládá se na principu posloupnosti jednotlivých fází vývoje (viz obrázek 2.1). Skutečnost, že před započetím následující fáze musí být vždy ukončena fáze předchozí, má své výhody i nedostatky. Výhodou je, že v době, kdy například provádíme návrh programu, jsou již specifikovány veškeré požadavky a nemůže se stát, že by nás překvapilo „něco nového". Další předností je, že návrh důkladně rozmýšlíme, než začneme programovat. Pokud však v některé pozdější fázi (např. při testech) zjistíme, že jsme v návrhu udělali chybu, není možné učinit změnu.

Z hlediska jednotlivých částí trojúhelníku vývoje softwaru (cena, čas, rozsah) klademe u tohoto modelu důraz především na rozsah. Tím myslíme, že předem stanovíme množství funkcí, kterými má výsledný produkt disponovat, což neměníme. Pokud dojde v některé fázi k potížím, navyšuje se čas potřebný k jejímu dokončení, a s ním pochopitelně i náklady. Časový faktor můžeme regulovat. Například zapojením dalších vývojářů, nebo prodloužením pracovní doby. Ale k navýšení nákladů dojde v každém případě.

Vzhledem k charakteru jednotlivých fází považujeme za velice důležité dokumentovat všechny kroky a závěry. Vodopádový model definoval v roce 1970 Dr. Winston Royce. Metodiky v této době byly určeny pro vývoj velkých aplikací vyžadujících přesnost (například vesmírný program). Navíc nástroje pro vývoj softwaru byly na velmi nízké úrovni a dostatečné dokumentování nezbytností. Dnes velmi zatracovaná metoda Waterfall je prakticky využitelná jen pro menší projekty.

Autor původně tuto metodiku zamýšlel jako iterativní. Předpokládal, že k úspěšnému



Obrázek 2.1: Vodopádový model.

použití potřebujeme alespoň dvou průchodů, přičemž při druhém průchodu se lze poučit z průchodu prvního. Fáze vývoje patrné z obrázku 2.1 patří k základům, na nichž staví i další metodiky.

2.2 Spirálový model

Barry Boehm navrhl v roce 1988 Spirálový model vývoje softwaru [JTSM/09]. Jedná se o čtyřfázový iterativní proces určení cílů pro danou iteraci, vyhodnocení rizik, vývoje a plánování další fáze a plánování dalšího cyklu spirály. Ve fázi určování cílů identifikujeme konkrétní cíle pro danou úroveň. Ve fázi hodnocení rizik hledáme případná rizika a způsoby, jak je minimalizovat. Rizikem je myšleno například překročení nákladů. V této fázi jsou zavedeny tzv. prototypy. Prototyp vytváříme pro každou iteraci znovu. Ve fázi vývoje provádíme skutečnou práci. Tím není myšleno jen programování, ale na příslušné úrovni i specifikace požadavků, nebo návrh.

Naposled plánujeme další cyklus spirály. Vidíme, že každá úroveň spirály odpovídá jedné fázi vodopádového modelu. Spirálový model tedy v podstatě každou fázi vodopádo-

vého modelu rozdělil na čtyři podfáze a zavedl prototypy. Samozřejmostí po projití kompletní spirálou je návrat na začátek a opakovaný postup, tedy další iterace. K tomu by mělo docházet i u vodopádového modelu, což různé literární prameny zapomínají uvést.

2.3 Unifikovaný proces vývoje softwaru

V následující části se seznámíme s metodikou Unified Software Development Process [UML2/07]. Metodika USDP je průmyslovým standardem pro tvorbu softwaru. Na vývoji se podíleli autoři jazyka UML [1]. Metoda bývá běžně nazývána také Unified Process (dále jen UP). Metodika UP je procesní částí tvorby softwaru. UP na rozdíl od UML není standardem. Metodika UP vychází kromě jiných z metod jako je Ericsson (1967) a Rational (1996-1997).

2.3.1 Vznik UP

Historie UP sahá do roku 1967 a je spojena s Ivanem Jacobsonem, v té době zaměstnancem firmy Ericsson. Ze stejné doby pochází myšlenka, že je třeba složité systémy modelovat jako navzájem propojené bloky. Malé bloky jsou seskupovány do větších, a tak se postupně utváří celková podoba systému. Objevuje se i zásada „rozděl a panuj" související s vývojem komponentového systému.

Ericssonova metoda zaváděla tzv. „provozní případy", které se časem rozvinuly do dnes známých případů užití. Kromě těchto částí obsahovala i dynamiku v podobě popisu komunikace jednotlivých stavebních bloků. Všechny prvky této metody byly zahrnuty do UML.

Dalším historickým milníkem bylo zveřejnění specifikace SDL (Specification and Description Language). Jazyk byl navržen pro zachycování chování telekomunikačních systémů, přičemž se jednalo o množinu komponent komunikujících mezi sebou prostřednictvím zasílání signálů. Šlo o první objektově orientovaný standard.

V roce 1987 Jacobson založil firmu Objectory AB. Ta vytvořila a prodávala metodiku Objectory založenou na Ericssonově modelu. Za podstatný přínos považujeme skutečnost, že tato metodika byla modelována a vyvinuta jako softwarový systém. Metodika Objektory, podobně jako UP, vyžadovala před nasazením úpravy dle potřeb zákazníka.

V roce 1995 kupuje firmu Objektory AB firma Rational. V té době se Jacobson zabývá sjednocením metodiky s dalšími metodikami a zavádí architekturu pohledů 4+1 (logický, procesní, fyzický a vývojový + sjednocující pohled případů užití). Dále formalizuje iterativní vývoj složený ze čtyř fází (zahájení, rozpracování, konstrukce a zavedení). Došlo ke sloučení iterativního (opakování jednotlivých cyklů) a přírůstkového vývoje. Svůj podíl na tom měli i Walker Royce, Rich Reitmann, Grady Booch a Philippe Kruchten. Výsledkem prací se stala metodika ROP (Rational Objectory Process), která zaplňovala slabá

[1] Z angl. Unified Modeling Language používaného pro grafické modelování programových systémů v softwarovém inženýrství. Využití tohoto jazyka je velmi všestranné. Díky němu lze velmi přehledně a efektivně specifikovat složité a odlišné procesy, různé softwarové komponenty, nebo databázová schémata.

místa metodiky Objectory. Hlavním hnacím mechanizmem bylo použito riziko. V této době vzniká UML, které je i pro ROP používáno. V roce 1998 vzniká metodika RUP. V roce 1999 publikuje Jacobson knihu Unified Software Development Process [USDP/99], v níž důkladně popisuje metodiku UP.

2.3.2 UP a RUP

V roce 2003 převzala společnost Rational Corporation firma IBM. Metodika RUP je komerční verzí UP a obsahuje nástroje, které UP neobsahuje a které v případě jejího užití musíme opatřit odjinud. Metodika RUP navíc disponuje i uživatelským prostředím a dokumentací.

V roce 1999 byl RUP pouze komerční implementací UP. Od té doby byl UP v podobě RUP rozšířen v mnoha směrech. UP je tedy spíše otevřeným standardem a RUP je jeho specifickou komerční podtřídou, která některé vlastnosti překrývá a některé rozšiřuje.

Obě metodiky modelují aspekty „kdo, kdy a co" tvorby softwaru. Také se od sebe liší terminologií, nikoli významem. Rozdíl vidíme například v aspektu „kdo". UP používá slovo „dělník" (worker) RUP pojem „role". V obou případech se ale jedná o jednotlivce nebo vývojový tým uvnitř projektu. V případě aspektu „jak" se setkáváme s pojmy „aktivity" (úlohy přidělené jednotlivcům nebo týmům) a „artefakty" (vstupy a výstupy projektu – např. zdrojový kód, spustitelné programy, dokumentace). Aspekt „kdy" je v rámci UP modelován jako „pracovní postup" (workflow). V metodice RUP se obecnějším pracovním postupům dává zvláštní název „disciplína". Pracovní postupy se skládají z aktivit, rolí a artefaktů. RUP na rozdíl od UP kromě jména těmto detailům přiděluje i speciální symbol.

2.3.3 Nasazení UP na nový projekt

Na metodiku UP je třeba se dívat jako na obecnou metodu tvorby softwaru. Proto pro každou společnost vyvíjíme software a pro každý specifický projekt metodiku specifikujeme. Do tohoto procesu zároveň začleňujeme vnitropodnikové standardy, šablony dokumentů, nástroje, databáze a další náležitosti. Přestože je metodika RUP konkrétnější než UP, musíme ji upravit. Na rozdíl od UP je však zapotřebí úprav mnohem menších. V každém případě si uvědomujeme, že každá společnost a každý projekt jsou něčím specifické, a proto na úpravu metodiky vynakládáme dostatečný čas a prostředky.

2.3.4 Axiomy UP

Metodika UP je založena na třech základních axiomech. Prvním axiomem je zásada řízení na základě případů užití. Ty se používají k zachycení požadavků. Metodika UP je řízena požadavky. S požadavky jsou také úzce spjata rizika. K posouzení rizik slouží analýza rizik. To je úloha pro manažera projektu.

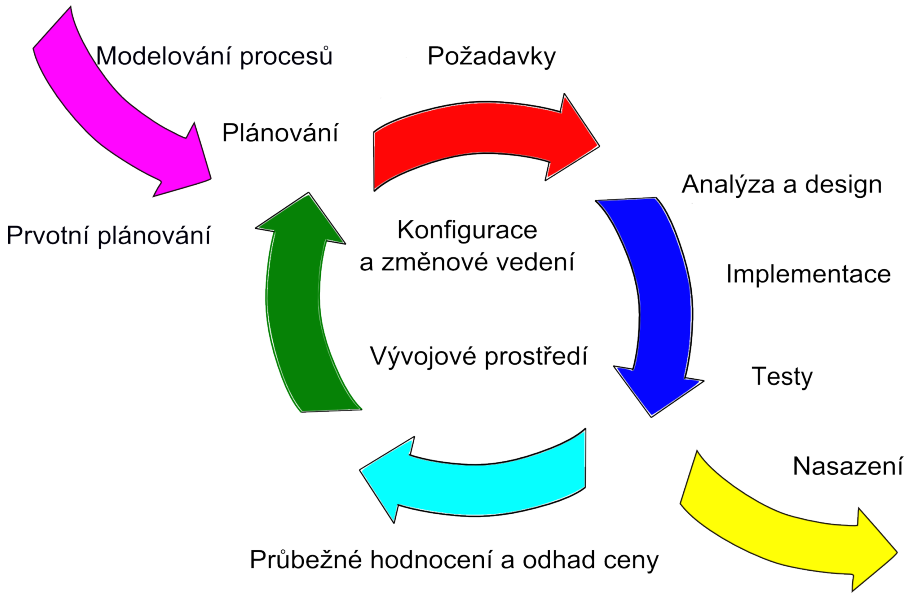
Druhým axiomem je zásada soustředění se na architekturu. UP si na návrhu a postupném vývoji robustní architektury zakládá. Ta v tomto případě popisuje rozklad na jednot-

livé komponenty a zároveň ukazuje, jak se jednotlivé komponenty navzájem ovlivňují a jak spolupracují s konkrétním hardwarem. Pokud očekáváme kvalitní konečný software, musíme architekturu dobře promyslet a vytvářet systém psaním kódu s maximální přípravou.

Třetím axiomem metodiky UP je zásada iterace a přírůstku. Iterací myslíme, že projekt rozdělíme na menší podprojekty, které se pojmem iterace nazývají. V každé iteraci se do výsledného systému přidává určité množství funkcionality zvané přírůstek. Na rozdíl od případu jednoprůchodové podoby vodopádové metody tak postupně systém zdokonalujeme a jednotlivé aktivity (jako například analýzu či návrh) provádíme několikrát. Postupným iterováním se pak dostáváme k plně funkčnímu výslednému systému.

2.3.5 Pracovní postupy iterace

Jak jsme již zmínili v předchozí části, v rámci metodiky UP se snažíme projekt rozdělit na menší podprojekty. Jde nám především o to, aby jednotlivé části lépe zapadaly do sebe a abychom zvýšili šanci na dokončení. Důležité je, že každá iterace obsahuje všechny prvky normálního vývoje softwaru. Jsou jimi plánování, analýza a návrh, tvorba, integrace a testování a nasazení.



Obrázek 2.2: Zjednodušené schéma iterace.

V každé iteraci je cílem vytvořit funkční podmnožinu konečného systému včetně dokumentace. Jednotlivé iterace se seskupují do fází, kterými se budeme zabývat později. Metodika UP zavádí pět základních pracovních postupů (aktivit). Jsou jimi požadavky, analýza, návrh (design), implementace a testování.

Požadavky specifikují, co by měl systém po této iteraci dělat. Analýza se stará o strukturování požadavků. Návrh převádí požadavky do architektury systému a implementace

představuje tvorbu softwaru. Testování představuje ověření kvality, zda implementace dělá to, co se předpokládá. Viz obrázek 2.2.

2.3.6 Fáze projektu a milníky

Jednotlivé iterace jsou seskupovány do fází. V životním cyklu projektu rozlišujeme celkem čtyři fáze. Jsou jimi zahájení, rozpracování, konstrukce a zavedení. Jednotlivé fáze končí milníky. Milník představuje indikátor pokroku projektu. Jednotlivé fáze mohou obsahovat různý počet iterací. Počet je závislý na typu projektu. Délka iterace by neměla přesahovat tři měsíce.

Fáze zahájení představuje období plánování projektu, rozpracování především období architektury, konstrukce slouží k uvedení projektu do počátečních fází provozuschopnosti a zavedení umožňuje nasadit výsledný produkt do cílového prostředí. Z tohoto plyne, že v různých fázích bude podíl práce v jednotlivých aktivitách jinak rozdělen. Zatímco během zahájení chybí výstupy k testování, při nasazení jsou již požadavky a analýza prakticky uzavřené. V každé fázi se soustředíme na konkrétní cíl představovaný podmínkami definovanými milníkem. Ony podmínky se snažíme splnit.

2.3.7 Zahájení

První fází projektu je fáze zahájení. V této fázi je zapotřebí specifikovat požadavky a provést jejich analýzu. Zvažují se obchodní přínosy a připadaná rizika. Vytváříme různé prototypy usnadňující komunikaci se zákazníky a pomáhající vyjasnit případné nedostatky. V této fázi nedochází k testování, protože u prototypů se předpokládá, že nebudou součástí výsledného produktu a odstraní se. Jak jsme se již zmínili dříve, každá fáze má milníky, které určují cíle, jichž musí být dosaženo. U fáze zahájení jsou to následující:

- Dokumenty požadavků na funkcionalitu a podmínky.
- Případy užití.
- Slovník projektu.
- Počáteční plán projektu.
- Dokument odhadu rizik.
- Obchodní případ.
- Základ dokumentu architektury systému a případně prototypy.
- Dokument o vizi.

2.3.8 Rozpracování

Další fází je rozpracování a hlavním výstupem spustitelný architektonický základ. Na rozdíl od předchozí fáze se již nejedná o prototyp, ale o spustitelný systém ve finálním jazyce[2]. Tento základ bude následně rozšiřován o další funkcionalitu. Dále pracujeme na vylepšování odhadu rizik, definování atributů kvality a tvorbě plánu pro konstrukční fázi. Případy užití zde pokrývají 80 procent funkčních požadavků. Již se zapojuje i testování architektonického základu, a tak má tato fáze všech pět aktivit (požadavky, analýza, návrh, implementace, testování). Její milník vedle zmiňovaného spustitelného architektonického základu definuje následující cíle:

- UML model (statický, dynamický a případy užití).
- Aktualizovaný obchodní případ.
- Aktualizovaný plán projektu.
- Dokument se záznamem dohody s uživateli a zainteresovanými osobami.

2.3.9 Konstrukce

Následující fází je fáze implementace. V ní se všechny požadavky specifikované analýzou a návrhem zabudovávají do spustitelného základu vytvořeného dříve. Nesmíme podlehnout časové tísni, aby nedošlo k narušení původní vize. Dbáme na zachování integrity architektury systému. V případě nedodržení může dojít ke snížení kvality systému a následným navýšením nákladů na údržbu. V prvních třech aktivitách (požadavky, analýza a návrh) dochází už jen k dolaďování a dokončování modelů. Nejvíce práce je třeba udělat na implementaci. Nesmíme ale opomíjet ani testování. Hlavním cílem milníku této fáze je:

- Softwarový produkt, který je dostatečně stabilní, aby jej bylo možné nasadit u uživatele.
- Model UML.
- Testovací sada.
- Uživatelské příručky.
- Popis verze.
- Plán projektu.

[2]Prototypy se často vytvářejí v jiném než cílovém jazyce nebo třeba jen v grafickém editoru.

2.3.10 Zavedení

Zavedení začíná ve chvíli, kdy byl softwarový produkt vytvořen, otestován a připraven k nasazení u uživatele. Soustředíme se hlavně na opravy chyb odhalených během provozu, přípravu prostředí pro běh softwaru[3], řešení neočekávaných problémů, tvorbu manuálů a dalších dokumentů i konzultace s uživateli. V této fázi se již požadavky a analýza nijak nemění. V případě, že dojde při testování k nalezení chyb v návrhu, je možné jej upravit. Práce se provádějí hlavně v implementaci a testování. Cílem posledního milníku je:

- Hotový, úspěšně nasazený softwarový produkt.
- Plán uživatelské podpory.
- Uživatelské příručky.

2.3.11 Požadavky

První aktivitou v rámci fází je získávání požadavků. Většina prací probíhá během zahájení a rozpracovaní. Pochopitelně, než začneme vytvářet objektově orientovanou analýzu, snažíme se získat alespoň rámcový přehled požadavků. Specifikaci systémových požadavků dělíme na model požadavků a model případů užití. Model požadavků se dále dělí na funkční a nefunkční požadavky. U případů užití systémový analytik nalézá aktéry a případy užití. Architekt na základě toho stanovuje priority případů užití. Osoba odpovědná za specifikaci případů užití pracuje na jejich upřesnění. Na základě toho systémový analytik vyhotoví strukturovaný model případů užití a návrhář uživatelského rozhraní vytvoří jeho prototyp (uživatelského rozhraní).

2.3.11.1 Model požadavků

V případě modelů požadavků má inženýr pro zpracování zadání za úkol najít funkční a nefunkční požadavky, na základě toho stanovit jejich priority a postoupit je architektovi pro přidruží k případům užití. Požadavky dělíme na funkční a nefunkční. Funkčními jsou například požadavky na zákazníky, produkty, objednávky, prodejní kanály či platby. Nefunkčními na výkon, kapacitu, dostupnost, shodu se standardy či zabezpečení. U jednotlivých požadavků rozlišujeme jejich prioritu. Ta se může pohybovat od nezbytnosti až po takové, které se dají odložit do dalších verzí. Rovněž bereme v potaz následující parametry: přímé uživatele systému, ostatní zainteresované osoby, systémy, se kterými bude systém komunikovat (navazovat na ně), hardware, na němž systém poběží, právní a regulační omezení a obchodní cíle. K získávání specifikací používáme konzultace či dotazníky.

[3]Případně úpravy softwaru, aby běžel v uživatelském prostředí.

2.3.11.2 Model případů užití

Modelování případů užití se skládá z několika činností. Je třeba najít hranice systému, aktéry a případy užití v podobě specifických užití a alternativních scénářů. Při hledání hranic systému je třeba určit, co je součástí systému a co není. Aktéři specifikují role představované osobami užívajícími daný systém. Případy užití jsou posloupnosti činností, které systém vykonává prostřednictvím interakce s vnějšími aktéry. Iniciátorem je aktér. V rámci případů užití jsou zachyceny i alternativní scénáře.

2.3.12 Analýza

Další aktivitou jsou práce na objektově orientované analýze. Začínají na konci fáze zahájení a většina z nich se provádí během rozpracování. Analýza se zabývá otázkou, co systém musí dělat, ale nesoustředí se na způsob, jak to má udělat. Výsledkem analýzy jsou analytické třídy tvořící klíčové pojmy v obchodní doméně a realizace případů užití ukazují, jak instance analytických tříd vzájemně komunikují. Architekt vytváří architektonickou analýzu u které inženýr případů užití analyzuje případy užití, a poté inženýr komponent analyzuje třídy a balíčky.

2.3.12.1 Analytické třídy

Analytické třídy reprezentují křehkou abstrakci problémové domény a měly by mapovat pojmy skutečného světa. Je zapotřebí jasnosti a jednoznačnosti. Neexistuje žádný jednoduchý algoritmus ukazující, jak postupovat při hledání tříd. Přesto se během let vyvinulo hodně postupů. Jsou jimi například hledání třídy na základě analýzy podstatných jmen a sloves; pomocí metody CRC karet[4] či stereotypů metodiky RUP.

2.3.12.2 Relace mezi analytickými třídami

Vedle analytických tříd je zapotřebí zachytit i relace mezi nimi. Rozlišujeme několik druhů relací: spojení - sdružování objektů, asociace - sdružování tříd, průchodnost či závislost (např. use, call, derive). Mezi třídami bývají i různé hierarchie v podobě dědičnosti. Jednotlivé třídy se seskupují do balíčků. Balíček seskupuje významově související prvky. Balíčky se mohou vnořovat, nemusejí obsahovat jen třídy, ale i jiné balíčky.

2.3.12.3 Realizace případů užití

Co se týče realizace případů užití, ta slouží k popisu spolupráce instancí analytických tříd. Realizace případů užití se skládají z diagramů analytických tříd, diagramů interakcí, speciálních požadavků a upřesnění případu užití. Důležitým pojmem je čára života představující jednoho účastníka interakce. Pro modelování interakcí používáme sekvenční

[4] Z angl. Class-Responsibility-Collaborator - když hledáme třídu, definujeme i její odpovědnosti a třídy s kterými spolupracuje.

diagramy (časově uspořádaná posloupnost) a komunikační diagramy (zdůraznění způsobu spojení životních čar). Dále vytváříme diagramy aktivit, což jsou objektově orientované vývojové diagramy.

2.3.13 Návrh

Po aktivitě analýzy následuje aktivita návrhu, poslední, kterou se budeme zabývat. Hlavní práce probíhají na konci fáze rozpracování a v první polovině fáze konstrukce. V rámci analýzy se tvoří hlavně logický model budoucího systému. Návrh specifikuje, jak implementovat funkčnost. Výsledkem návrhu je návrhový model skládající se z jednotlivých subsystémů, návrhových tříd, rozhraní, návrhových realizací případů užití a diagramů nasazení.

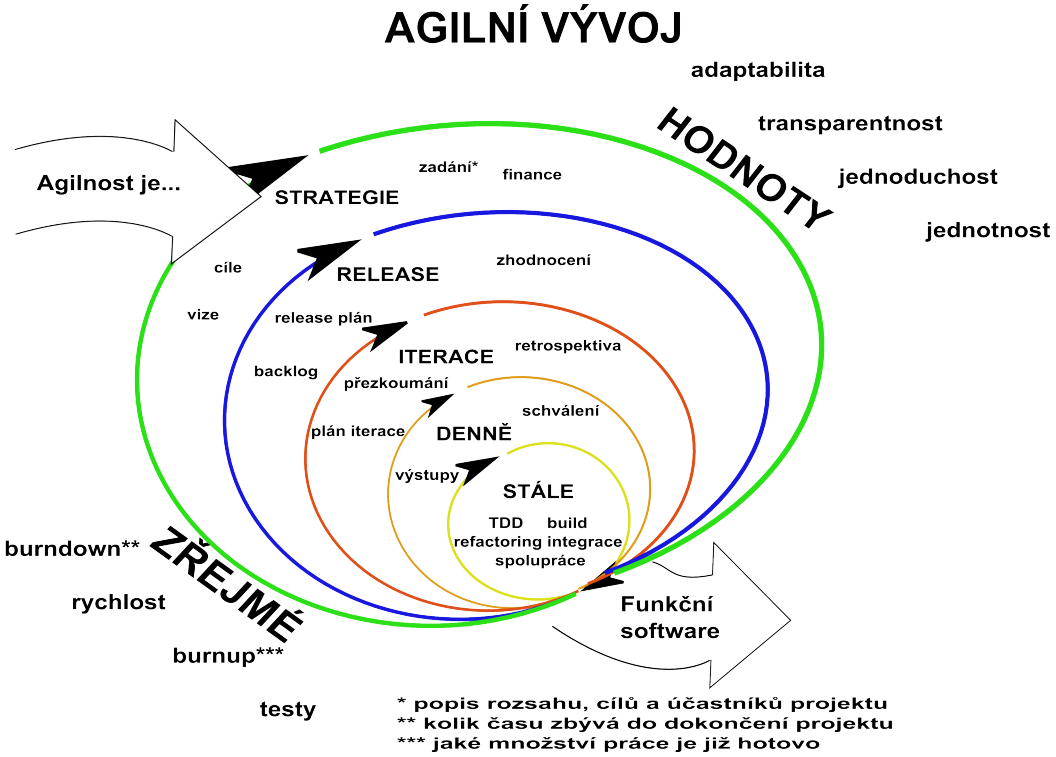
Architekt postupuje tak, že zpracuje architektonický návrh, na základě něhož inženýr vatvoří komponent návrhy tříd. Na základě architektonického návrhu a návrhu tříd vytvoří inženýr případu užití jednotlivé případy užití. Na základě toho inženýr komponent navrhne jednotlivé podsystémy. V době návrhu se přetvářejí analytické třídy a načrtnuté třídy a rozhraní na konkrétní úplné třídy a rozhraní. Dále následuje upřesňování analytických relací do podoby relací návrhových. Setkáváme se tu s pojmy jako agregace a kompozice. V době návrhu zároveň realizujeme případy užití. K tomu slouží návrhové diagramy interakce. Mezi další nástroje využívané pro modelování dynamického chování patří stavové automaty. U diagramů používajících stavové automaty jsou důležité jednotlivé stavy (atributy objektu, relace mezi objekty, aktuální aktivita) a přechody mezi stavy - události, podmínky, akce.

2.4 Agilní metodiky

Ve druhé polovině devadesátých let začala vznikat nová skupina metodik, pro které je hromadně používán pojem Agilní metodiky [AGP/04]. Patří sem například Extrémní programování (používána zkratka (XP) Extreme programming), Scrum či Crystal. Agilní techniky staví na několika základních myšlenkách. Jsou to vícenásobné krátké iterace, v nichž dochází k vytváření funkční verze softwaru s určitým omezeným množstvím funkcí a se schopností tuto funkčnost rozšiřovat při dalších iteracích. Čím kratší iterace, tím lépe. Dále je kladen důraz na komunikaci, a to jak v rámci týmu, tak se zákazníkem. Důležitou vlastností vedoucí ke vzniku agilních metodik byla schopnost přizpůsobit se měnícím se požadavkům. V dnešní době rychle se měnící podoby světa může dojít ke změně zadání ze dne na den. Nebo jen stačí, aby se zákazník s vývojovým týmem špatně pochopili. Ve chvíli zjištění je třeba vše okamžitě napravit. Proto agilní metodiky musí být schopné přijmout požadavek na přeměnu zadání v kterékoliv fázi vývoje. Jestli se může programátor v dnešní době na něco spolehnout, pak je to vědomí, že se určitě něco změní.

Důležitou částí je i neustálé sledování kvality vývoje po celou dobu jeho trvání (u modelu Waterfall k tomu došlo až ve fázi testování). Proto je nedílnou součástí tvorby psaní jednotkových testů, které umožňuje testování těch nejmenších částí a také zajišťuje bezpeč-

nost refactoringu[5] a provádění případných změn. Mnohé týmy praktikují testy řízený vývoj za pomoci metodiky TDD, kdy se nejprve vytvoří test pro určitou metodu, a ta se následně implementuje. Dále se snaží tyto metodiky snižovat přílišnou byrokracii. Z hlediska trojúhelníku vývoje softwaru je považujeme za velice flexibilní. Díky systému postupného přidávání funkcionality v krátkých iteracích můžeme výsledný produkt ovlivňovat v závislosti na zákazníkem požadované ceně a době dodání. Agilní vývoj charakterizuje přehledným způsobem obrázek 2.3 (obrázek inspirován [ASDP]). V následující části krátce charakterizujme některé agilní metodiky.



Obrázek 2.3: Náhled na agilní vývoj.

2.4.1 Extrémní programování

Základní myšlenkou extrémního programování (dále jen XP) je, že:

„Jediným exaktním, jednoznačným, změřitelným, ověřitelným a nezpochybnitelným zdrojem informací je zdrojový kód." [AGP/04]

Uvádí se, že metodika je vhodná pro 2-10 členné týmy. XP je postaveno na čtyřech hodnotách. Komunikace, jednoduchost, zpětná vazba a odvaha. Komunikace je velice důležitá a jakékoliv opomenutí v tomto směru způsobuje problémy nebo zpoždění. Jednoduchost

[5]Jedná se o proces zefektivňování zdrojového kódu (lepší přehlednost, údržba, čistota), aniž by docházelo k ovlivňování jeho chování navenek.

spočívá ve snaze programátorů programovat nejmenší funkcionalitu „právě teď" a nesoustředit se na celkovou funkcionalitu do budoucna. Zpětná vazba je realizována v podobě programátorských testů a uživatelských testů. Odvaha je na první pohled zvláštní hodnotou, ale když se zamyslíme například nad možností odhodit polovinu dosud napsaného zdrojového kódu (i to se může za určitých okolností stát), i tato hodnota má svůj smysl.

Na tyto hodnoty navazuje dvanáct základních postupů. Plánovací hra (vytváření plánu, kterého se účastní všichni členové týmu); malé verze (časté uvolňování verze – mezikroky); metafora (nahrazuje architekturu); jednoduchý návrh (co nejjednodušší v daném okamžiku); testování (průběžné); refaktorizace (zefektivňující restrukturalizace systému bez změn chování); párové programování (dva lidé sdílejí jeden počítač, jeden programuje, druhý s nadhledem kontroluje); společné vlastnictví (kdokoliv můžeme měnit jakoukoliv část zdrojového kódu); nepřetržitá integrace (systém je sestavován nejméně jednou denně); čtyřicetihodinový pracovní týden; zákazník na pracovišti (zákazník se fyzicky účastní vývoje) a standardy pro psaní zdrojového textu (všichni členové týmu musejí programovat stejně). Čtyřmi základními činnostmi vývoje jsou testování (testy modulu jsou programovány před samotným modulem); psaní zdrojového kódu; naslouchání (zákazníkům i kolegům) a design (abychom se nedostali do tzv. „slepé uličky").

Výhodami této metodiky jsou soulad s lidskými instinkty (radost z učení, z komunikace s jinými lidmi, z dokončení úkolu apod.), a také s iterativním inkrementálním vývojem v krátkých cyklech. Nevýhodou je hlavně nasazení této metodiky. Může být problematické zvyknout si na jiný přístup (jednoduchost, přiznání nevědomosti a komunikace).

2.4.2 Scrum

Na rozdíl od metodiky XP stojí metodika Scrum na odpovědnosti konkrétního vývojáře za určitou množinu objektů s jasně definovaným chováním a rozhraním. Podobně jako XP tato metodika staví na iteracích zvaných Sprint. V rámci vývoje je jich 3-8 a každá trvá obvykle jeden měsíc. Asi nejcharakterističtějšíší vlastností této metodiky je stanovení denních schůzek zvaných Scrum Meetings. Na nich dochází ke shrnutí již splněných úkolů a stanovení cílů do další shůzky. Metodika se nesnaží plánovat na delší období než jsou denní schůzky, protože vychází z předpokladu, že není možné odhadnout, jak budou práce probíhat. Každý sprint (měsíční cyklus) je zakončován tzv. demem, což je prezentace dosavadních výsledků zákazníkovi.

Klíčové pro tuto metodiku jsou flexibilní předměty dodání (záleží na dohodě se zákazníkem); flexibilní harmonogram (dodání může proběhnout dříve nebo později, než bylo plánováno, je ale důležité, aby s tím byl srozuměn zákazník); malé týmy (tří až šesti členné, v případě větších projektů na něm pracuje více týmů); časté revize (každodenní schůzky) a spolupráce (v rámci týmu i se zákazníky).

Metodika definuje tři fáze: předehru, hru a dohru. Ve fázi předehry dochází k plánování systémové architektury a vysokoúrovňového návrhu. Fáze hry probíhá v podobě sprintů, jejichž náplní je vývoj, zabalení, revize a přizpůsobení. Ve fázi dohry dojde k uzavření (nevyvíjí se další funkcionalita), provedení integrace a jejímu otestování.

Hlavní výhodou metodiky Scrum je možnost pružně reagovat na změny (na denní bázi). Specifický je rovněž propracovaný způsob odhadu pracnosti. Mezi nevýhody, kromě těch popsaných u XP (nasazení), počítáme i fakt, že se jedná spíše o souhrn vzorů než o popis přesných kroků. Je tedy dobré (nikoli nutné) nasadit Scrum na již běžící metodiku a pouze upravit způsob vedení projektu.

2.4.3 Lean Development

Na rozdíl od ostatních metodika Lean Development (dále jen LD) nemá původ ve snaze zlepšovat vývoj softwaru. Vychází totiž z metodiky Lean Manufactoring vytvořené pro potřeby automobilového průmyslu. LD se snaží vyvíjet software za třetinu obvyklého času, s třetinou obvyklého rozpočtu a se snížením chyb na třetinu obvyklého množství.

Původně vycházela z deseti pravidel, jenž měla být aplikována na vývoj softwaru:

- Odstranění zbytečností.
- Minimalizace zásob.
- Maximalizace toku.
- Vývoj ovlivňuje poptávka.
- Pracovníci mají pravomoc rozhodovat.
- Mezi hlavní cíle patří uspokojování požadavků zákazníků.
- Zavedení zpětné vazby.
- Odstranění lokálních optimalizací.
- Partnerství s dodavateli.
- Kultura pro neustálé zlepšování.

Aby byl minimalizován výdej zásob, nejsou vytvářeny v daný okamžik nepotřebné produkty, jako například dokumentace. Snaha o maximalizaci toku vede k rozdělení prací na menší části a k souběžnému provádění úkonů. Pravidlo vývoje ovlivňovaného poptávkou stojí na přesvědčení, že by se měla rozhodnutí dělat v nejpozdějším možném termínu na základě aktuálních potřeb zákazníka. Skutečnost, že pracovníci mají možnost rozhodovat, dává jednotlivým členům týmu prostor určité věci měnit (vše není řízené shora), ale je třeba určit jasné hranice. Cíl uspokojovat požadavky zákazníků vede k neustálé revizi poptávky (mění se, nebo zákazník neví, co vlastně chce). Je třeba se zákazníkem neustále komunikovat jako s partnerem. K tomu napomáhá zavedení zpětné vazby. Při postupném předvádění práce zákazníkovi dochází k upřesňování požadavků. Odstranění lokálních optimalizací obsahuje požadavek, že bychom se měli vyvarovat plýtvání energie na dílčí optimalizace, protože ty nikam nevedou, pokud nedojde k optimalizaci celku. Partnerství

s dodavateli nám umožňuje vytvářet řetězec subdodavatelů (různých komponent), na které se můžeme spolehnout. Kultura pro neustálé zlepšování může mít podobu například motivace a spadá do oblasti projektového řízení.

Z těchto deseti pravidel vzešlo sedm principů: eliminace plýtvání; rozvoj učení; rozhodování co nejpozději; rychlé a časté dodávky; pravomocní pracovníci; integrita a vize celku. Metodika LD není striktním a komplexním popisem vývoje softwaru. Spíše se pomocí deseti pravidel a sedmi principů snaží naznačit, jak by vývoj měl probíhat.

2.4.4 Feature Driven Development

Další agilní metodikou je Feature Driven Development (dále jen FDD). Jak název napovídá, jedná se o vývoj řízený vlastnostmi[6]. Cílem FDD je dodání kvalitní aplikace (hlavně z hlediska cílových vlastností). Stejně jako u jiných agilních metodik, vývoj probíhá iterativně, přičemž iterace trvá zhruba 1-3 týdny a vždy po skončení iterace se zákazníkovi odevzdává fungující meziprodukt (s novými funkčními přírůstky). Každá vlastnost je definována trojicí: akce, předmět a podrobnosti. Akce je činnost prováděna v rámci dané vlastnosti. Předmět označuje artefakt, nad kterým se tato akce provádí. Podrobnosti upřesňují vlastnosti a další parametry.

V rámci FDD definujeme postup pěti fází: vytvoření celkového modelu, vypracování podrobného seznamu vlastností, plánování, návrh a implementace. V první fázi se vytváří celkový model systému a úvodní objektový model. Součástí bývá i prvotní seznam vlastností. Ty jsou dopodrobna zaznamenávány ve druhé fázi. Pochopitelně není možné vytvořit definitivní neměnný seznam, ale na základě výsledků první fáze je velká část vlastností zřejmá. Ve fázi plánování jsou vytvářeny další plány vývoje. Nejdůležitějším datem, který se v této fázi stanovuje, je datum dokončení, který slouží jako mezní pro ukončení vývoje. Fáze návrhu a implementace jsou iterativně opakovány. Ve fázi návrhu se pracuje s konkrétními vybranými vlastnostmi. Je vybrána vlastnost (soubor vlastností), již bude tým schopný implementovat přibližně za dva týdny. Dále zjišťujeme vlastnictví příslušných objektů (podobně jako u Scrum, kde za daný objekt zodpovídá ten který vývojář) a utváříme dočasné týmy. Během implementace vlastník dotčené třídy implementuje jednotlivé metody, vytváří testovací třídy a provádí potřebné testy. Když je třída implementována, dochází k její integraci do hlavní aplikace.

Hlavním specifikem FDD jsou objektové modelování domén (diagramy tříd); vývoj podle vlastností; vlastnictví tříd (za úspěšnou implementaci jedné třídy zodpovídá jeden člověk) a týmy sestavované podle vlastností. Zde upozorňujeme, že jeden člověk je součástí více týmů. Aby to bylo možné, jsou pro každou vlastnost určeni vedoucí týmu, a programátoři zodpovědní za danou třídu spadající do dané vlastnosti mezi týmy přecházejí.

[6]Feature – rys, znak, vlastnost, důležitá část.

2.4.5 Test Driven Development

Poslední agilní metodikou, se kterou se seznámíme, je Test Driven Development (vývoj řízený testy, dále jen TDD). TDD se spíše soustředí na části testování a implementace. Vychází z předpokladu, že je dobré odstranit chyby v době, kdy vývojář pracuje na dané části kódu. Za charakteristickou vlastnost TDD považujeme požadavek, že testovací kód musí být dokončen před započetím psaní samotného testovaného kódu. Vývoj probíhá v pěti krocích. Nejprve napíšeme test požadované vlastnosti. Poté test spustíme. Protože v tuto chvíli ještě není funkcionalita implementovaná, testy neuspějí. Následuje krok implementace (psaní či úpravy kódu). Když si myslíme, že máme funkcionalitu implementovanou, spustíme znovu testy. Ty by měly proběhnout úspěšně. Pokud ne, vrátíme se k opravě implementace a spustíme testy znovu. V posledním kroku přichází na řadu refaktoring. Důležitou podmínkou zajišťující, že bude zjednodušování kódu bezpečné, je právě provádění testů. Pokud po refactoringu testy projdou, dochází k psaní testu pro další funkcionalitu.

K tomu používáme tzv. jednotkové testy. Použití TDD ale nijak nenahrazuje ostatní druhy testování. Ty je třeba stále provádět, jako například testy uživatelského rozhraní či integrační testy.

3 Projekt Space Traffic

Jak již název kapitoly dává tušit, seznámíme se nyní s projektem ST blíže. Autor stál před otázkou, jak velké množství informací a zkušeností, které při svém působení v projektu získal, přehledně a srozumitelně zachytit. Jím zvoleným řešením je pohled skrze tři podkapitoly na tři akademické roky (dále uváděné zkratkou 1.RST, 2.RST, 3.RST), kdy práce na projektu probíhala. A to přehledně, formou časové osy, kde je možné zhlédnout události týkající se vývoje projektu. V některých případech podrobněji rozepisuje i zajímavé aspekty vývoje. Je zhodnocen význam každého roku pro projekt ST. V poslední čtvrté podkapitole autor zmiňuje řadu poznatků, na které by neměl zapomínat žádný vývojář působící v ST.

Víme, že výstupem ST má být webová hra typu MMORTS (z anglického Massively-Multiplayer Online Real-Time Strategy), tedy strategická hra pro více hráčů hrajících prostřednictvím internetu, která bude zasazena do prostředí solárních systémů a planet ve vesmíru. Mezi nimi bude hráč obchodovat skrze svou společnost a její obchodní flotilu. Hru bude možné hrát v rámci webového prohlížeče a zdarma.

Uveďme nyní zdroje, ze kterých tato kapitola čerpá. Autorovi práce se naskytla jedinečná příležitost přímé i nepřímé účasti na projektu po celou dobu jeho trvání. Díky tomu měl možnost se seznámit s většinou na něm pracujících studentů a čerpat ze svých i jejich zkušeností i nezveřejněných výsledků práce.

3.1 Počátek 2009-2010

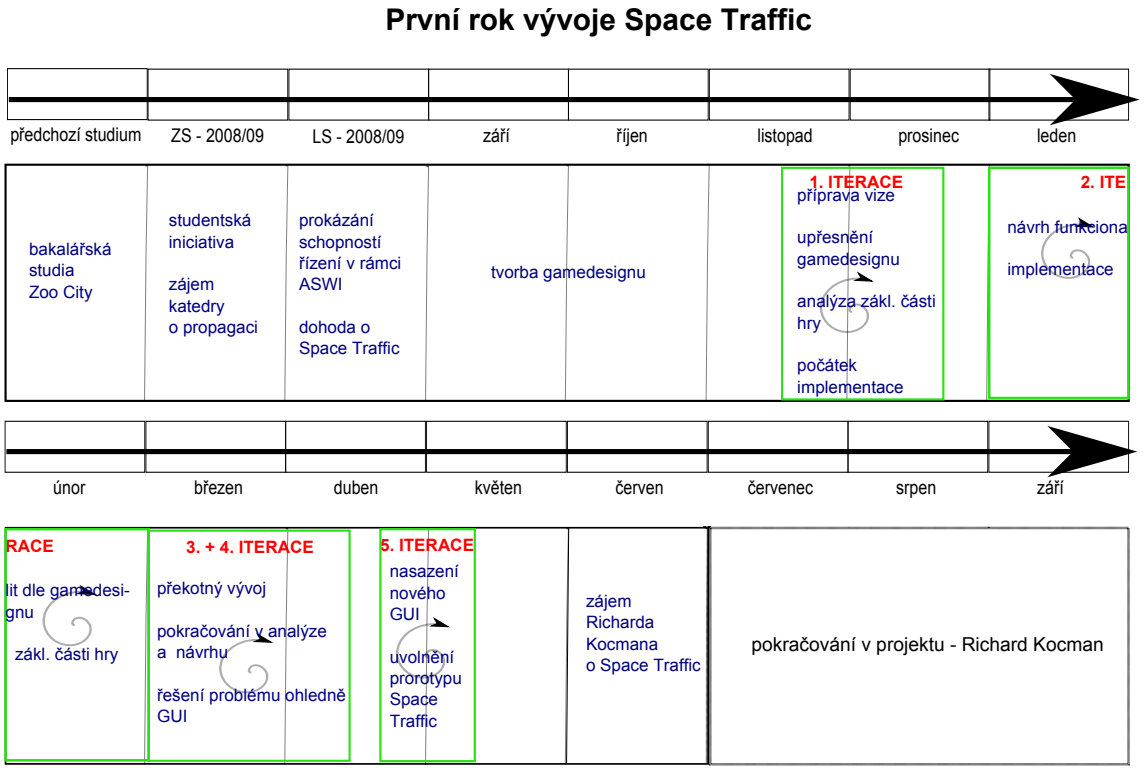
U vzniku ST stál Zbyněk Neudert (dále jen Z.N.). Když Z.N. úspěšně ukončil bakalářské studium na Českém vysokém učení technickém, stal se studentem ZČU v oboru Softwarové inženýrství. Chtěl pokračovat na projektu vycházejícím z jeho bakalářské práce [NWH/08] a zabývajícím se návrhem webové hry Zoo City[1]. Pokračovat ve vývoji této webové hry mu umožněno nebylo, ale dostal možnost zahájit vývoj hry nové, určené k propagaci KIV a zvané Space Traffic. Pod vedením Z.N. byl v 1.RST vytvořen pevný základ pro vývoj dokument gamedesignu[2], prototyp ST a diplomová práce [ANVWH/10]).

[1] Hra Zoo City měla vyplnit mezeru na trhu her pro předškolní a mladší školní věk, pokud nebereme v úvahu jednoduché, flashové a často bezduché mini-aplikace jejichž hlavní nosnou myšlenkou je oblékání panenek, házení míčku a podobně. Pojmem Flash myslíme grafický vektorový program společnosti Adobe sloužící k tvorbě webových reklam, prezentací, nebo jednoduchých her.

[2] Myšlen dokument nebo i činnost návrhu hry, kde se specifikuje téma, příběh a pravidla do nejmenších detailů.

3.1.1 Časová osa - fakta

Časová osa níže, na obrázku 3.1, nás seznamuje s podstatnými fakty, které se v průběhu 1.RST udály.



Obrázek 3.1: Průběh prvního roku vývoje Space Traffic.

3.1.1.1 Předchozí zkušenosti

Z.N. měl zkušenosti s týmovou prací, neboť se během studia na ČVUT podílel na tvorbě webové hry Zoo City (dále ZC) (více bakalářská práce [NWH/08]). V průběhu vývoje byl prováděn průzkum trhu webových her, vytvářen gamedesign, analyzovány požadavky nutné k vývoji her a získány cenné zkušenosti i v problematice řízení projektů. Vývoj webové hry Zoo City nebyl zcela dokončen a realizační tým měl zájem na něm i nadále pracovat, přestože každý z jeho členů pokračoval v magisterském studiu na jiné vysoké škole.

3.1.1.2 Propagace katedry

Pokus navázat na ZC ihned na počátku navazujícího studia na ZČU učinil i Z.N (projekt měl být využít jako podklad pro diplomovou práci). Představitele KIV ale tato myšlenka nezaujala. Měla zájem propagovat katedru mezi žáky středních škol. Společně s Z.N. byl nastíněn jeden ze způsobů, jak tuto potřebu řešit. A to zatraktivněním webu KIV webovou hrou zaměřenou právě na zmíněnou cílovou skupinu.

3.1.1.3 Dohoda o zahájení projektu Space Traffic

Schopnosti Z.N. jako projekt manažera byly prověřeny v předmětech KIV/ASWI, ve kterých vedl několik studentů KIV/ZSWI. Byly získány i podklady pro gamedesign ST a vytvořen velmi jednoduchý prototyp. Na základě těchto výsledků bylo rozhodnuto o realizaci vývoje úplně nové webové hry nazvané Space Traffic. Došlo k vypsání diplomové práce, jejímž vedoucím (i garantem projektu) se stal Ing. MSc. Přemysl Brada Ph.D.(dále jen P.B.).

3.1.1.4 Příprava návrhu hry

V září 1.RST byl vytvořen dokument gamedesignu. Ze zdrojů inspirace uveďme především předchozí zkušenosti s tvorbou hry ZC, webové hry jako Travian a Divoké kmeny, výstupy z předmětu KIV/ASWI, knihu Art of Game Design [AOGD/08] a konzultace s Ing. Petrem Vaněčkem Ph.D. Na základě těchto podkladů byl vytvořen koncept gamedesignu hry (ten je i součástí diplomové práce [ANVWH/10]). Dokument je členěn tímto způsobem:

- Základní rysy a princip hry: Zde formulujeme kategorie a cílovou skupinu hry. Také popisujeme základní herní principy.
- Gameflow: Zde je nastíněn průběh hry z pohledu hráče (tedy od registrace nového uživatele, po dokončení hry).
- Grafika, zvuky a hudba: V této sekci je rozvedena představa ohledně podoby webové hry.
- NPC[3] a herní fyzika: Zde je promyšlen vliv NPC na hráče, fyzika herních objektů.
- Návrh řešitelského týmu: V něm je nastíněna představa ohledně potřebného vývojového týmu k úspěšnému dokončení projektu a odhadnuta doba realizace.

3.1.1.5 Počátky vývoje

Pro řízení projektu byl zvolen iterativní přístup (více viz podkapitola 2.2) a kombinace metodik UP a FDD (viz podkapitola 2.3.2 a 2.4.4). Na základě analýzy požadavků vzešlých z game-designu došlo k vytvoření velkého množství pracovních úkolů setříděných dle jednotlivých funkcionalit. Návrh a implementace měly spočívat na programátorech, ale vzhledem k tomu, že se podařilo získat pouze jednoho, nebylo naplnění metodiky FDD smysluplně možné. Návrhové a implementační úkoly programátora značně zatížily.

3.1.1.6 Grafické práce

Ve hře typu ST je kromě kvalitního obsahu nutné nabídnout i uspokojivou grafiku. Jinak aplikace v hráčích nevzbudí dobrý dojem. I když se přes tento nedostatek přenesou,

[3] Z anglického non-player character, tedy postavy (nebo jiná umělá inteligence) neovládané hráčem.

neustále podvědomě pociťují, že byli o něco ochuzeni. A cílovou skupinou webové hry je v případě ST mládež, která na graficky propracovaných hrách vyrostla. Díky spolupráci s UUD (Ústav umění a designu při ZČU) nakonec grafická podoba ST vznikla (na druhý pokus).

Ačkoli se v průběhu druhé a třetí iterace spolupráce vyvíjela zdárně (byly vytyčeny představy, předány podklady), nebyla nakonec úspěšná (zadaná práce nespěla ke konci, neboť se na ní průběžně nepracovalo). Na tuto skutečnost se přišlo nepřímo, až v druhé polovině 1.RST, kdy stav implementace dovoloval testovací nasazení GUI (grafického uživatelského rozhraní z angl. grafic user interface). Řešením bylo vypsání stipendia díky němuž se podařilo i ve velmi krátkém čase GUI vytvořil a nasadit.

3.1.1.7 Překotný vývoj

Celý vývoj byl silně poznamenán velkou časovou tísní vycházející z vize 1.RST (kde se počítalo s dokončením hry ST do konce akademického roku, z čehož vycházelo i zadání diplomové práce Z.N.). V průběhu čtyř iterací, tedy mezi 4.1.2010 a 11.5.2010 byly všechny síly zapojeny k dosažení výše zmíněného cíle. Ale implementaci a návrh realizoval pouze jediný programátor (v letním semestru už pouze ve svém volném čase). V závěru páté iterace se podařilo výsledný produkt nasadit, bylo provedeno otestování hry, ale pouze z pohledu hráče (po stránce hratelnosti).

3.1.2 Podrobnosti k vybraným aspektům

Na několik skutečností spojených s 1.RST se podíváme blíže.

3.1.2.1 Studenská iniciativa

Studenská iniciativa je na akademické půdě ze stran vyučujících téměř vždy ceněna a přijímána (viz příklad vzniku ST výše). Je ale důležité, aby si student svůj podnět, ať už pro zadání diplomové práce, oborového projektu apod. pečlivě promyslel. Nestačí zvolit jen vhodné, nám na první pohled blízké či jednoduše působící téma. Musíme mít jasnou představu o tom, jaké a jak velké cíle si vytyčíme. Pokud rozsah práce špatně zvážíme, bude nás tato skutečnost (že i při maximálním úsilí práci nestihneme) pronásledovat po celou dobu jejího naplňování, což je velmi skličující.

3.1.2.2 Iterativní vývoj

Před začátkem každé iterace nutně vytváříme plán. Každý z nich musí obsahovat cíle a klíčové úkoly spolu s návrhem, jak cílů dosáhnout s ohledem na čas nutný k jejich splnění. Dále musíme stanovit i kritéria úspěšnosti, dle kterých bude výsledná iterace hodnocena. Konec iterace je dobré spojit s konzultací u garanta, mentora projektu. Jejím obahem by měla být především kontrola odvedené práce na základě zvolených cílů zpravidla spočívající

v kontrole nově implementované funkcionality. V případě zjištění nějakého nedostatku nebo nejasností je možné rychle nastínit návrh řešení či požádat o radu.

3.1.2.3 Průběžná kontrola zadané práce

Průběžnou kontrolu zadané práce považujeme za nedílnou součást každého dobře řízeného projektu. Úspěšně tak předcházíme situacím, kdy někteří členové vývojářského týmu neodvádějí své povinnosti, ale zároveň vzbuzují dojem (mnohdy velmi sofistikovanými způsoby), že na jim přidělených úkolech usilovně pracují. Tak tomu bylo v případě grafických prací viz výše.

3.1.3 Zhodnocení

V 1.RST se podařilo vytvořit betaverzi hry (jak se ukázalo 2.RST, na výsledný produkt bylo možné nahlížet jen jako na propracovaný prototyp), specifikovat dokument gamedesignu (na kterém se staví do současnosti) a analyzovat část požadavků z něj vycházejících. K dalším přínosům počítáme nastínění postupů vhodných či nevhodných pro studenský projekt tohoto typu (využití studentů jako lidského zdroje, plánování v iteracích, průběžná kontrola zadaných úkolů atd.). Ačkoli tedy nebyla splněna původní vize, nasazení webové hry do rutinního provozu, podklad pro budoucí vývoj ano.

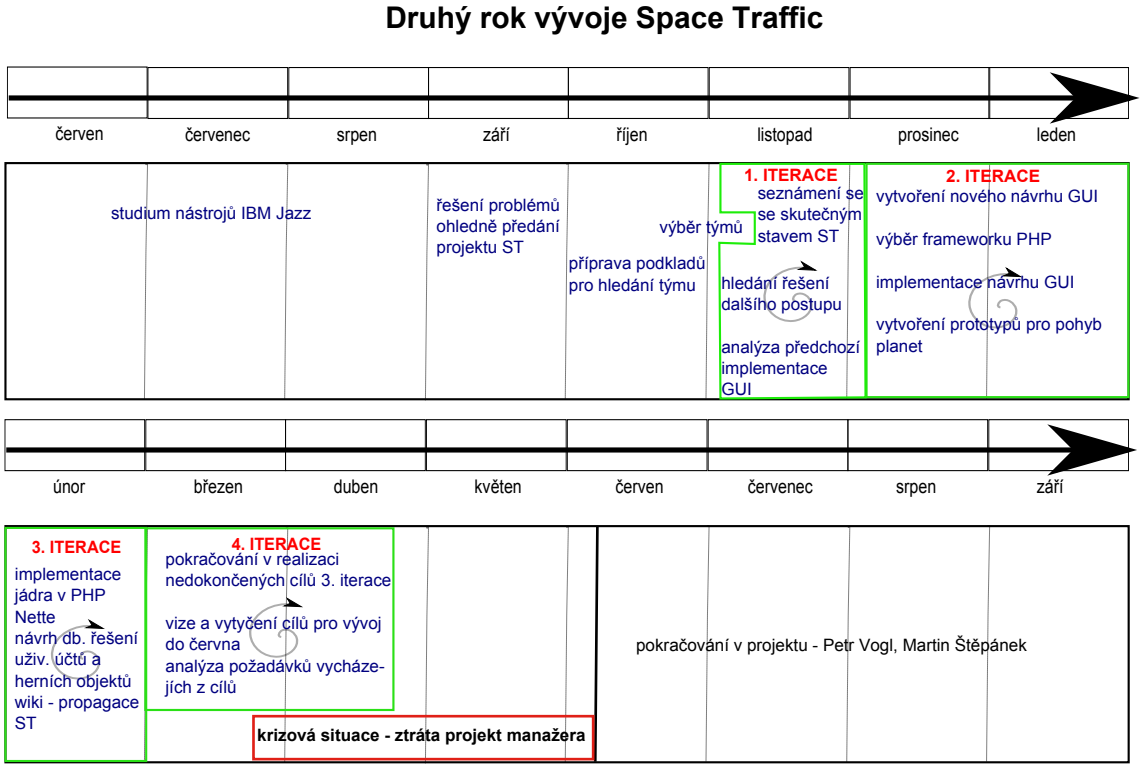
3.2 Pokračování 2010-2011

Pokračovatelem projektu pro akademický rok 2010-2011 se stal Richard Kocman (dále jen R.K.). Přestože se 1.RST přímo neúčastnil, Z.N. blíže znal jako kolegu z vysoké školy a souseda z kolejí, o práci s ním diskutoval a domníval se, že má alespoň malou představu o tom, co projekt ST obnáší. Oba dva měli v oblasti informatiky podobný okruh zájmů a v budoucnu se chtěli zabývat oblastmi plánování a řízení při vývoji softwarových produktů. R.K. tedy bez většího rozmýšlení šance využil, ačkoliv jediným zdrojem znalostí, ze kterého čerpal, bylo absolvování předmětu KIV/ASWI. Chtěl především získat zkušenost ze skutečného projektu v jiné něž programátorské pozici. A protože příležitostí vést skutečný projekt a plánovat jeho vývoj dle postupů softwarového inženýrství není mnoho, byl doslova nadšen. Proto se projektu po odchodu Z.N. ujal.

V 2.RST pak řešil problémy ohledně navázaní na projekt 1.RST. I když se ukázalo, že není možné na implementaci z 1.RST dále stavět, vývoj nezastavil, ale naopak jej opět rozběhnul. Získal také zkušenosti s nasazením metodiky UP, a pod jeho vedením byla mimo jiné provedena hloubková analýza grafického uživatelského rozhraní, navrženo jeho nové řešení a provedena implementace. Ve druhé polovině roku se pokusil vytyčit nové cíle, začal sběrem a analýzou požadavků dalších oblastí webové hry, které plánoval splnit do června 2.RST. To se ale nepodařilo z důvodu mimořádné události (více viz dále).

3.2.1 Časová osa - fakta

Nyní si podrobněji probereme důležité události, které v průběhu druhého roku vývoje ST nastaly. Na obrázku 3.2 můžeme vidět průběh 2.RST i v grafické podobě.



Obrázek 3.2: Průběh druhého roku vývoje Space Traffic.

3.2.1.1 Seznámení s platformou IBM Jazz

Před začátkem akademického roku probíhala příprava především skrze studium nástrojů IBM Jazz. Jako hlavní zdroj znalostí sloužil web IBM Jazz (jazz.net) umožňující i stažení instalačních souborů softwarových nástrojů platformy. Ty byly následně nainstalovány a práce s nimi vyzkoušena. Vybrány byly dva. Pro řízení a vývoj projektu Rational Team Concert (dále jen RTC) a Rational Requirments Composer (dále jen RRC) pro sběr a analýzu požadavků. Více viz. kapitola platforma IBM Jazz a její nástroje. Takto získané znalosti měly význam i při úvodních setkáních s novými vývojáři a jejich zaučování.

3.2.1.2 Předání projektu

V září došlo k řešení problému s předáním projektu ST. Možná nás napadne otázka, proč až po studiu nástrojů IBM Jazz. Příčinou je především fakt, že k žádnému předání na konci 1.RST nedošlo. Díky e-komunikaci s minulým vedením (Z.N.) v průběhu léta se

podařilo získat přehled o postupech řízení a organizaci ST (celá řada těchto informací se dala čerpat z diplomové práce [ANVWH/10]), ale situace ohledně přístupu na server nebyla jednoduchá a ke kontaktu s programátorem odpovědným za návrh a design kódu mělo dojít až v průběhu dalšího akademického roku, nikoli v během letních prázdnin. Přístup na server umožnili správci KIV, ale až v říjnu. Jedna schůzka s programátorem z 1.RST se uskutečnila (byl stále studentem ZČU), ale do projektu se odmítal jakkoliv dále zapojit a ke své implementaci ST se nechtěl více vracet.

3.2.1.3 Výběr týmu

Následujícím měsíci říjnu byl hledán nový vývojový tým. Využilo se poznatků z 1.RST, kdy se ukázalo, že mezi nejefektivnější způsoby oslovení studentů patří náborové plakáty a provádění informačních prezentací na přednáškách a cvičeních předmětů majících obsah osnov podobný aktuálním potřebám ST. Protože platformou projektu bylo PHP a jednalo se o vývoj webové hry, vybrány byly předměty jako KIV/WEB, KIV/PIA a KIV/DB1. Nedošlo ani k podcenění přípravy. Na IBM Jazz serveru byl založen projekt pro 2.RST a pro nové členy vývojového týmu by připraven souhrn informací(jak rozběhnout projekt ve vývojovém prostředí, založení uživatelských účtů), aby jim byl vstup do projektu co možná nejvíce ulehčen. Podařilo se získat dva studenty bakalářského studia a jednoho studenta navazujícího studia (Martina Štěpánka, dále jen M.S., působící i ve 3.RST). Po několika úvodních setkáních byly zavedeny pravidelné schůzky po čtrnácti dnech. Ostatní kontakt probíhal především skrze Skype konference.

3.2.1.4 Seznámení se skutečným stavem projektu

V listopadu byla naplánována první iterace s cílem seznámit se se zdrojovým kódem a navázat na vývoj z 1.RST. Původní vize pro 2.RST počítala s tím, že budou opraveny chyby verze z 1.RST a opravená verze hry představena na dni otevřených dveří KIV. Ve druhé polovině akademické roku bude hra nasazena a testována v rutinním provozu. Při studiu PHP kódu původní implementace se zjistilo, že implementace v 1.RST probíhala hekticky a chaoticky. Na místo betaverze ST tak měl tým 2.RST v rukou propracovaný prototyp, podklad pro analýzu.

Projekt manažer (R.K) se nacházel v obtížné situaci, protože cíle původní vize nebylo možné splnit. Pokusil se situaci řešit naplánováním úkolů hlubší analýzy kódu a smysluplnosti jeho refactoringu. Souběžně s tím probíhala analýza stávajícího grafické uživatelské rozhraní ST. Na počátku prosincové iterace se ukázalo, že na implementaci z 1.RST skutečně není možné stavět. Se stavem projektu byl na úvodní prosincové schůzce seznámen garant projektu (P.B.), který na základě předložených faktů rozhodnutí o nové implementaci posvětil (Více o implementacích viz kapitola Implementace webové hry).

3.2.1.5 Pokračování ve vývoji

Na začátku prosince bylo rozhodnuto o nové implementaci ST a předchozí implementace sloužila jen jako prototyp. Již se počítalo s tím, že prosincová a lednová iterace budou po stránce odvedeného výkonu podprůměrné (vánoce, v lednu zkouškové období). Pro leden a únor byla vytvořena jedna společná iterace. Ze strany vedení týmu vzešla snaha poklesu výkonu zabránit. Ihned na počátku prosince došlo k organizaci několika schůzek, při nichž se kladl velký důraz na jasné rozdělení úkolů mezi členy týmu. Prvním z nich bylo vybrání vhodného frameworku PHP (po dvou týdnech testů zvolen Nette). Druhým vybrání funkcionalit, které budou implementovány. Rozhodlo se o využití kvalitní analýzy grafického uživatelského rozhraní z minulé iterace, tedy i o její implementaci. Dále pak analýze a implementaci registračního procesu webové hry a návrhu řešení pro interaktivní pohyb planet na základě Keplerových zákonů, realizovaný za pomocí technologií jako JScript, SVG, CSS3 nebo HTML5. Postupy z minulého 1.RST nebylo možné použít, protože pohyb každé planety se načítal z textových souborů z dopředu vypočítanými hodnotami.

Plán iterace se podařilo splnit. Implementovalo se GUI dle nového návrhu. Vzniklo i několik velice efektních ukázek realizace pohybů planet ve výše zmíněných technologiích, jenž posloužily i jako podklad pro prezentaci na dni otevřených dveří. Proběhla i analýza registrace webové hry, problematika herních účtů.

3.2.1.6 Únorová iterace - studenské povinnosti

Na únor byla naplánována třetí iterace řešící integrace implementovaného GUI s pohybem planet a implementaci procesu registrace hráčů do hry. Jednalo se o jádro aplikace a základní analytické třídy v rámci Nette MVC architektury. Co se týče analýzy a návrhu, bylo nutné především zvolit a rozpracovat databázové řešení registrace. Probíhal i nábor studentů na letní semestr, započalo vytváření stránky ST v rámci KIV wiki s informacemi pro případné zájemce a rozvěsily náborové plakáty po KIV. V únoru, ale byla odvedena jen malá část naplánované práce, protože studenti při práci na předchozí iteraci nestihli splnit své studijních povinností.

3.2.1.7 Plány na letní semestr

Cíle čtvrté březnové iterace zůstaly z velké části stejné jako cíle únorové iterace, pracovalo se na jejích nedokončených úkolech. Ale bylo rozhodnuto stanovit cíle, kterých je nutno dosáhnout do června. Na základě hlubší úvahy, několika společných setkání a dobré předchozí zkušenosti z výsledků druhé iterace se rozhodlo o těchto:

- Kompletní implementace přihlašování a registrace hráčů.
- Implementace solárních systémů s nástroji na jejich vytváření.
- Implementace map (systému, galaxie) se základní indikací, bez speciálních efektů, včetně testování.

- Implementace pohybů lodí, jejich správa a základní příkazy (posílání lodí po mapě, ruční obchodování).
- Základní implementace zásilek a obchod s NPC (jednoduché NPC základny).
- Komunikace, pošta.
- Alespoň částečná integrace wiki nápovědy.
- Portál s blogem.
- Intro, základní nástroje tutoriálu (musí se rozšiřovat podle nových funkcí).

Při zdárné realizaci výše zmíněných cílů se ještě uvažovalo o vyrobení podkladů pro:

- Základny a jejich budování.
- Rozpracování herní ekonomiky.
- Problematiku pirátů.
- Engine pro programování AI lodí a související editor.
- Hodnocení hry.
- Rozhraní pro minihry a způsob jejich integrace.
- Customizace lodí, vylepšování a levelování.

V průběhu března probíhala dekompozice výše nastíněných cílů do jednotlivých úkolů. Což se ukázalo jako velmi nešťastné řešení, protože vzhledem k jejich velkému množství a komplexnosti nebylo možné tyto úkoly splnit[4]. Už při sběru požadavků, který probíhal formou dvou intenzivních jednodenních setkání se ukázalo, že množství představ, způsobů realizace a konkretních požadavků je obrovské. Jen pokus o analýzu výstupů z těchto schůzek tvořených stovkami papírových poznámek a náčrtů nebo hodinami hlasových nahrávek nebylo v silách vývojářského týmu zpracovat.

3.2.1.8 Krizová situace

V průběhu druhé poloviny března došlo události mající zásadní vliv na projekt ST. Projekt manažer (R.K.) se nemohl věnovat projektu ST, ani svému studiu, protože se v jeho rodině objevily vážné zdravotní potíže, R.K. nebyl po mnoho týdnů v Plzni přítomný. A to se zásadně projevilo jak v jeho studiu[5], tak v ST.

[4]To se potvrdilo i v 3.RST, kdy byla realizována jen malá část z výše zmíněných cílů.

[5]Studium se podařilo stabilizovat v průběhu května a června díky ochotě vyučujících i schválení přerušení studia děkanátem.

3.2.2 Podrobnosti k vybraným aspektům

Na několik výše uvedených skutečností se podíváme blíže.

3.2.2.1 Význam předání softwarového projektu

S nutností předání práce v rámci softwarového projektu se při studiu moc často nesetkáváme, protože víceletých studenských projektů, jako je ST, je relativně málo. U ST ale došlo mezi 1.RST a 2.RST ke kompletní obměně vývojářského týmu. Proto bylo více než žádoucí plynulé předání projektů provést už na konci 1.RST, což se však nestalo. Nouzové řešení, získaní informací o projektu, až v září se ukázalo jako nepostačující. Studenti působící v ST 1.RST už na univerzitě nestudovali, a ti kteří ano, nebyli ochotni v ST dále působit a odmítali spolupráci, ať už z jakýchkoliv důvodů (důvodem mohl být i několika měsíční odstup od účasti na projektu a zapomenutí mnoha souvislostí). Z toho vyplývá, že pokud v daný akademický rok v projektu nepůsobí student, který počítá se svou účastí v ST pro následující akademický rok (a to v odpovědné pozici projekt manažera apod.), musí se při závěru akademického roku vyhradit čas na přípravu podkladů pro předání (tím je myšleno např. web wiki s důležitými informaci, seznámeni s nástroji používanými při vývoji, předání uživatelských účtů nutných k jejich ovládání atd.). V neposlední řadě je zde i nutnost několika osobních setkání, při kterých se projekt svěří do rukou dalšího realizačního týmu tzv. „z očí do očí".

3.2.2.2 Příprava pro hledání týmu

Při hledání týmu a oslovení studentů je dobré přípravu nepodcenit. Protože i ti dají na první dojem. A tak je potřeba při vytváření náborových prezentací i úvodních schůzkách vystupovat profesionálně (srozumitelně představit projekt, seznámit s úkoly na akademický rok, zdůraznit přínosy pro studenta, nastínit možnosti realizace dle různých schopností uchazeče, ochotně zodpovědět případné dotazy tak, aby byl vzbuzen co možná nejpozitivnější dojem). Jednou z nejdůležitější věcí je studentovi jasně předat kontakt na projekt (několikrát jej v rámci prezentace zopakovat), protože zájem o zapojení se zpravidla neprojeví okamžitě. Dále je vhodné konat úvodní schůzky ve zmámen prostředí univerzity a vyměnit si na sebe kontakty, kromě e-mailů i např. skype údaje, aby bylo možné provádět hlasové konference, protože komunikace v prvních fázích spolupráce bývá velmi intenzivní (řeší se účty do nástrojů pro vývoj a jiné nejasnosti atd.). Pokud v úvodní fázi vstupu do ST není uchazeč rychle zapojen do vývoje, jeho zájem o ST se velmi rychle ztratí.

3.2.2.3 Závažná rozhodnutí a obtížné situace

Obtížná a závažná rozhodnutí (jako např. výše zmíněné rozhodnutí o zahájení implementaci ST od začátku) není možné odkládat, i když jejich jednoduché řešení nevidíme a zdá se nám, že nemáme dostatek sil k jejich řešení. Pokud si nevíme rady, je vhodné co nejdříve zkonzultovat problém s kolegy a pokud řešení nenajdeme, tak s garantem projektu. Bez zbytečných prodlev se uspokojivé řešení vždy najde. To platí i v případě, že si

za obtížnou situaci můžeme sami. Je nutné překonat obavu z reakce garanta projektu, s jádrem problému jej seznámit, nečekat jen jak se k němu postaví, ale osvětlit mu i důvodu, proč k němu došlo a jaká vidíme východiska.

3.2.2.4 Studenské povinnosti studentů - úkoly při vývoji Space Traffic

Každý student má velké množství studenských povinností. Aby je mohl naplňovat a odvádět práci i v projektu ST, je nutné dbát na to, aby je mohl plnit pokud možno současně. Tedy aby se práce, které odvádí v rámci ST, staly i povinnostmi-součástmi jeho zápočtových prací. Poté může v průběhu vývoje i ve zkouškovém období provádět úkoly pro projekt ST.

3.2.2.5 Žádné velkolepé plány

Jak se ukázalo, při vytváření vizí není dobré postupovat neuváženě. Činíme-li tak na akademický rok (nebo jen jeho části) nesepisujeme množinu nesouvisejících záměrů a z nich vycházejících cílů, kdy často nemůžeme ani odhadnout, zda je možné námi definované představy v námi zvoleném časovém horizontu realizovat. Představa by měla být vždy jen jedna, měla by určovat rámec, oblasti (např. implementaci grafického uživatelského rozhraní nebo implementaci hvězdných systému), kterých se týká, a tu by měla blíže konkretizovat. Z vize vycházející cíle by ji měly postupně realizovat.

3.2.2.6 Krizová situace - ztráta vedení

Díky krizové situaci (zmíněné výše ve faktech 2.RST), kdy byl náhle pro ST ztracen projekt manažer, se zbytek týmu dostal do problematické situace. Nikdo z programátorů nebyl oficiálně zainteresován do projektu. Projekt manažer byl jediným prostředníkem mezi katedrou a projektem. P.B. vystupoval převážně jen jako vedoucí diplomové práce R.K., ne mentor pro ST, alespoň takto situace působila na členy týmu. Je tedy velmi důležité, aby vedení projektu nestálo pouze na jednom člověku.

3.2.3 Výběr vhodné metodiky pro vývoj

Autor práce zvolil v 2.RST, kdy vedl projekt ST metodikou UP, nástroje RRC a RTC. Metodiku se rozhodl využít na základě faktu, že obsahuje silně definované jednotlivé role, artefakty a pracovní postupy. Vzhledem k tomu, že podobně, jako většina studentů, neměl autor dostatek zkušeností z praxe, nemohl ocenit přínosy, které nabízejí agilní metodiky a bez předchozích zkušeností je nemohl ani dost dobře nasadit.

Počítalo se tedy s nasazením UP, což v počátku vývoje bylo naplňováno za pomocí výše zmíněných nástrojů a zaškolování jednotlivých členů týmu do jejich použití. Skrze tyto nástroje je bylo možné řídit a sledovat aktivitu při plnění úkolů. V tomto směru se ukázal výše zmíněný postup jako úspěšný, i když zpočátku vyžadoval vynaložení velkého úsilí při instruktáži méně zkušených studentů.

Přesto se s nasazením metodiky UP pojilo několik problémů. Bylo velmi náročné nasadit UP na projekt po 1.RST, protože nebylo jasně zřejmé, v jaké fázi vývoje se ST nachází. Před seznámením se s implementací se předpokládalo, že projekt bude v závěru fáze konstrukce nebo zavedení. Autor tedy jako manažer zadal skrze program RTC jednotlivým členům týmu úkoly pro analýzu stávajícího řešení společně s podklady z 1.RST (zdrojovým kódem, dokumentem gamedesignu, který lze částečně chápat jako zdroj požadavků na funkcionalitu a UML model implementovaných funkcionalit), aby předpoklad byl potvrzen.

Ale záhy se došlo k poznání, že softwarový produkt, který by se vzdáleně blížil předpokládanému stavu (byl dostatečně stabilní a nasaditelný u zákazníka), neexistuje. Podklady z 1.RST nešlo použít ani při refactoringu zdrojového kódu produktu, protože ten postrádal návrh, byl chaotický a nepřehledný.

Rozhodlo se tedy, že stav projektu bude nahlížen, jako-by byl ve fázi zahájení. Nutně muselo dojít k naplnění cíle milníku zahájení, tj. vytvoření dokumentu požadavků na funkcionalitu, případů užití, slovníčku projektu a dalším náležitostem.

Při vytváření dokumentů však autor zastával příliš pedantský přístup k vytváření dokumentů. Domníval se totiž, že je to jediný správný postup zaručující snížení rizik neúspěchu projektu a ulehčení dalšího vývoje. Z několika důvodů ve svém předpokladu neuspěl. Nezkušení členové týmu neznali metodiku UP. Dále je třeba říci, že UP je příliš „byrokratická". Dodržováním postupu podle metodiky se vytváří velké množství dokumentů jistě důležitých pro velké projekty a týmy, ale zcela neefektivních, zatěžujících a v podstatě kontraproduktivních při nasazení na projekt rozsahu ST. Tým se skládal převážně z ve fázi zahájení obtížně využitelných programátorů. Na druhou stranu bylo nemyslitelné nechat je čekat do doby (už bylo nabráno značného zpoždění), kdy dojde k vytvoření jednotlivých dokumentů, na jejichž základě by jim mohla být přidělena práce. Takový postup je možný pouze u velké firmy, kde působí větší množství analytiků či architektů. U studentského projektu to možné není, protože vývojářský tým se skládá hlavně z nezkušených programátorů, a analytiků (zkušených) bývá poskrovnu.

Vidíme, že metodiku UP nelze bez dalších úprav pro projekt ST použít. Za ideální považujeme snížení byrokratičnosti celého procesu. Nijak tím není myšleno, že by se mělo vyvíjet stylem „vše dokumentující kód", jak to naznačují některé agilní metodiky (viz. Extrémní programování). To je opačný extrém a vzhledem k víceletému charakteru projektu to není ani možné. Můžeme počítat s udržováním určitého množství dokumentace (například v podobě wiki stránek).

Vzhledem k charakteru projektu je vhodné vytvořit metodiku kombinací metodiky UP (používání nástrojů RTC a RRC pro určitou podmnožinu úkolů) a některých dobrých vlastností agilních metodik (hlavně FDD). V projektu ST ale není možné předpokládat, že každý „programátor bude vlastnit a znát svou komponentu", protože programátoři se často obměňuji. Nad nimi musí dlít senior programátor, mentor, který bude výsledky práce do projektu začleňovat (a průběžně vychovávat nové kandidáty na tuto pozici). Pro realizaci se následně vybere jen malá podmnožina funkcionality, na které dojde k provedení fáze UP. Není tedy možné v rámci metodiky UP pojmout projekt jako celek, ale je nutné tuto metodiku aplikovat na omezené množství funkcí tak, abychom podmnožinu stihli im-

plementovat v rámci akademického roku a jednotlivé funkcionality v období 2-3 měsíčních iterací.

Metodiku UP lze velmi dobře naplňovat skrze nástroje IBM Jazz. Ty ovšem kladou nezanedbatelné nároky na uživatele, což může být pro studenty, kteří s nimi přijdou poprvé do styku, velice obtížné.

V poslední řadě je třeba poukázat na podobu iterací a specifik studentského vývoje. Když stanovíme dobu iterace na jeden měsíc (vyjma zkouškového období a prázdnin), je třeba toto období dále rozdělit na kratší úseky rozdělené pracovními poradami či kontrolami postupu tak, jak to ukazují některé agilní metodiky. Studenti mají tendenci pracovat hlavně před odevzdáním práce. Pravidelná kontrola postupu je velmi vhodná i k případnému objevení nesrovnalostí v návrhu (a včasné reakci na ně).

3.2.4 Zhodnocení

Druhý rok vývoje ST přinesl množství nových poznatků (úspěšný výběr týmu, společná pravidelná setkání, nutnost propojení studenských povinností a úkolů ST, nebo úspěšné navázaní na vývoj, i když nedošlo k hladkému předání projektu). Tým provedl analýzu prototypu z 1.RST, na jehož základě vytvořil propracovaný návrh nového GUI. V PHP Nette implementoval jádro hry, do které bylo GUI implementováno. Dále analyzoval proces registrace uživatelů a vytvořil návrh jeho řešení. Původní cíl 2.RST (nasazení hry do rutinního provozu) se splnit nepodařilo z důvodu rozsahu projektu a neuspokojivého stavu implementace z 1.RST. Ale povedlo se do projektu zainteresovat studenty, kteří v ST úspěšně působí i v 3.RST a kteří na výsledky 2.RST úspěšně navázali.

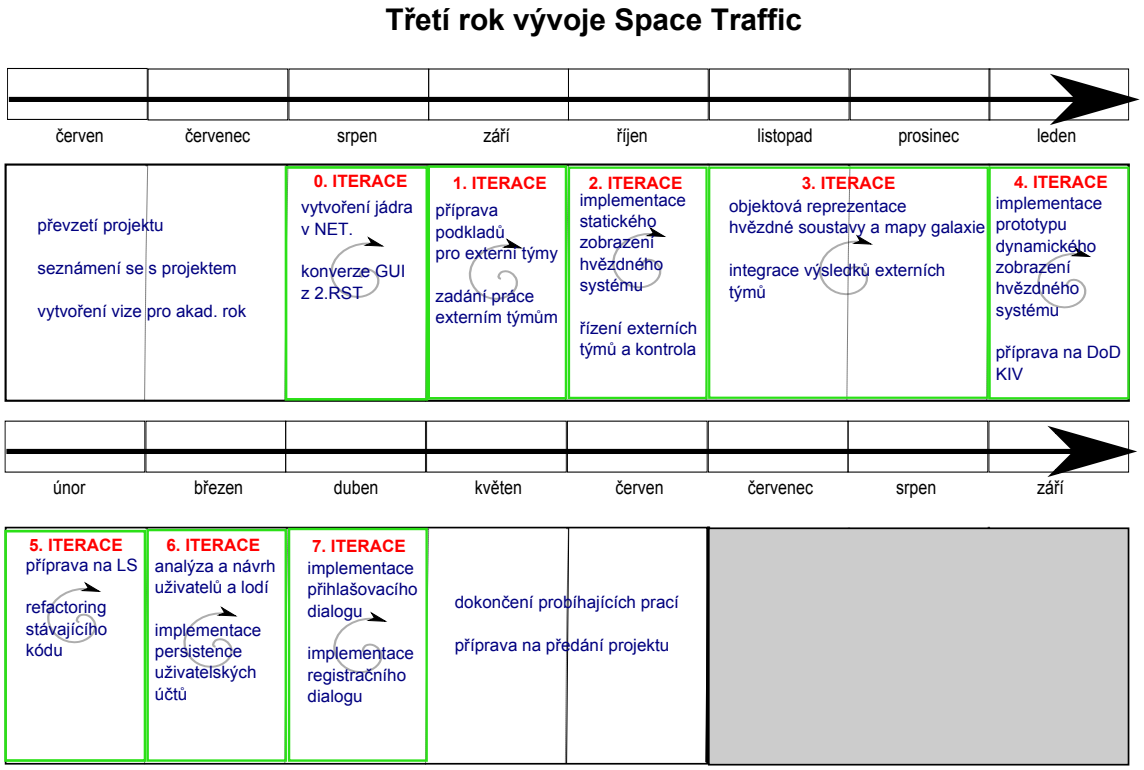
3.3 Další rozvoj 2011-2012

V průběhu 2.RST se projekt ST dostával čím dál více do podvědomí studentů KIV. To vzbudilo zájem i u Petra Vogla (dále jen P.V.), který se rozhodl do projektu zapojit také. P.V. už delší dobu uvažoval o vytvoření vlastní webové hry a po studiu chtěl svou vizi realizovat. Od účasti v projektu si především sliboval získaní „know-how" pro tvorbu her. P.V. působil v projektu z pozice projekt manažera, což pro něj nebyla role nová, protože již své schopnosti prokázal jako vedoucí týmu v KIV/ZSWI, později i v KIV/ASWI. Ve vedení projektu, ale nepůsobil sám. Martin Štěpánek zůstal členem vývojového týmu i v 3.RST. Zastával pozici analytika a návrháře.

V 3.RST se řešily úkoly ohledně změny vývojové platformy, kladl se velký důraz na brzké zapojení nových studentů do ST, jejich průběžnou kontrolu a poklidný a transparentní vývoj (cílem nebylo vytvořit během jednoho akad. roku výsledný produkt, ale přehledný zdrojový kód a dobře organizovaný projekt).

3.3.1 Časová osa - fakta

Obrázek 3.3 nám pomůže vytvořit si představu o 3.RST.



Obrázek 3.3: Průběh třetího roku vývoje Space Traffic.

3.3.1.1 Navázání na předchozí vývoj

Na konci 2.RST byl projekt ST v neutěšeném stavu (krizová situace ohledně projekt manažera). Iniciativu převzal M.S., později po zapojení do ST i P.V. Při setkáních s představiteli KIV před začátkem 3.RST bylo rozhodnuto o dvou zásadních věcech. První věcí byla změna vývojové platformy, protože PHP ani s využitím framworku Nette nebylo pro implementaci projektu vhodné (více viz kapitola Implementace webové hry). Druhou věcí bylo vytyčení cílů pro 3.RST. Změnou bylo to, že se nepočítalo s dokončením projektu v rámci jednoho akademického roku, ale jen s vytvoření architektury pro modulární a parametrizovanou implementaci jádra a to na platformě .NET.

3.3.1.2 Vytvoření kostry aplikace

Už v průběhu léta byla připravována kostra aplikace, aby zapojení nových vývojářů do ST, bylo co nejplynulejší. Kostra jádra sytému byla vytvořena. Dále byla převzata maska GUI z PHP verze, která vznikla v průběhu 2.RST. Byl proveden test skrze volaní

jednoduché služby (pro přihlášení uživatele), kdy událost a data z prohlížeče byly předány klientské aplikaci a přes volání služby ověřeny na serveru. Bylo tím prokázáno, že základ aplikace je životaschopný a hodný dalšího rozšíření.

3.3.1.3 Transparentní vývoj

V zimním semestru v projektu působili čtyři studenti předmětu KIV/PT. Jejich výsledky museli být integrovaný do aplikace pod stálým dohledem vedení ST. Podařilo se vytvořit objektový návrh hvězdných systému a provést jeho implementaci na serverové části aplikace.

Vývoj úspěšně pokračoval i v letním semestru. Podařilo se získat i tým studentů z navazujícího studia (z předmětu KIV/ASWI). Na rozdíl od studentů KIV/PT se jednalo o zkušenější studenty, kteří měli ve svém týmu rozdělené role i vlastního projekt manažera. Jejich přílišná samostatnost však byla i nevýhodou. Protože jakmile dostali zadaní své práce, neměli potřebu s vedením ST dále komunikovat. Mezi úkoly druhé poloviny 3.RST patřil vývoj přihlašovacího a registračního dialogu.

3.3.2 Podrobnosti k vybraným aspektům

Zmiňme ještě několik podrobností patřících k 3.RST.

3.3.2.1 Promyšlený postup při hledání týmu

Postup získávání nových pracovních sil byl dále vylepšen. Hlavní důraz byl kladen na zadávaní práce pro ST skrze zápočtové práce předmětů na KIV. Bylo nutné připravit podklady už na konci léta. Celý postup se skládal s několika kroků. Nejdříve bylo nutné zvolit předměty vhodné pro realizaci vize ST pro daný akademický rok. Pak byl nalezen průsečík mezi obsahem předmětů a obsahem vize. Následně byl osloven cvičící vybraného předmětu. Iniciativa byla přivítána a koncept zadání i myšlenka spolupráce přijata. O vyhašená nestandardní zadaní byl projeven velký zájem a mohl tak být proveden i výběr mezi více zájemci.

3.3.2.2 Profesionální řízení týmu

Aby bylo možné efektivně řídit tým je nutné splnit několik předpokladů. Vedení ST musí v průběhu každé iterace kontrolovat postup podřízených týmů. Ideálně na společných setkáních každých čtrnáct dní. Obsahem schůzky je kromě kontroly odvedené práce i zadaní úkolů nových. Hlavní realizační tým ST v tomto případě zastává i časově velice náročnou roli mentorů. Učí základní programátorské dovednosti a správné vývojové postupy.

3.3.3 Zhodnocení

P.V. a M.S. posunuli projekt o značný kus dál. Dokázali, že vývoj skrze čistě studenský projekt je možný. Stavěli na výsledcích práce svých předchůdců, ale především se poučili z jejich nezdarů a chyb. V mnohém pomohla i kontinuita, kterou představoval M.S., jež se účasnil 2. i 3.RST. Výsledkem 3.RST bylo vytvoření jádra Space Traffic na platformě .NET, implementace hvězdných soustav, mapy galaxie, jejich zobrazení, implementace přihlašovacího i registračního dialogu. Nezapomnělo se ani na důkladnou přípravu na předání projektu. Byl vytvořen wiki portál o projektu na webu „http://spacetraffic.kiv.zcu.cz/code/", kde kromě informací pro zájemce jsou uloženy podklady pro budoucí vývoj nebo návody „HOW TO" jak začít. To vše na jednom místě.

3.4 Důležité poznatky z vývoje Space Traffic

S jednoletým odstupem od účasti autora na projektu ST v pozici projekt manažera můžeme konstatovat, že ne všechna rozhodnutí učiněna v rámci jeho působení byla nejšťastnější. To ale, domníváme se, v žádném případě nesnižuje význam této práce. Autor přiznává, že mohl před dvěma lety při vstupu do projektu postupovat výrazně odlišně. Ale neměl téměř žádné praktické zkušenosti s pozicí projekt manažera a do projektu vstupoval, dalo by se říci, jako velmi naivní student. To co jej dělí od něj samého před dvěma lety je právě zkušenost a poznaní.

3.4.1 Odhadni a koriguj

Odhadování náročnosti úkolů v jakémkoli projektu je velmi složitou záležitostí. Metoda expertního odhadu je sice i v praxi hojně užívána, nicméně ukázalo se, že také velmi nepřesná, což velice komplikuje plánování iterací. (I expertní odhady mívají velkou chybu v odhadu [OSP/06]. V projektu ST je realita mnohdy daleko horší.)

Dle názoru autora je pro projekt ST vhodné odhadovat náročnost úkolů podle postupu popsaném v rámci metodiky Scrum. Stanovení náročnosti probíhá na schůzkách, kde je vždy přednesen konkrétní problém, všichni členové týmu sdělí své odhady a následně alespoň extrémní hodnoty obhajují. Po vyjasnění dotazů probíhá odhadování znovu, dokud nedojde ke shodě.

3.4.2 Cena – čas – rozsah

Přestože jsou v univerzitním prostředí vyučovány a používány nejnovější poznatky z oblasti softwarového inženýrství včetně agilních technik, samotné univerzitní prostředí jde v několika ohledech proti těmto směrům. Víme, že v projektu rozeznáváme tři aspekty ovlivňující produkt, a těmi jsou cena – čas - rozsah. Dále také víme, že zákazník by měl správně definovat pouze dvě z těchto veličin. V univerzitním prostředí je však cena, resp. kapacita jednoho studenta neměnná, čas přesně stanoven a rozsah často nastaven před

započetím práce na projektu. Tato kombinace pak může vést, stejně jako v běžné praxi, ke snížení kvality výsledného softwarového produktu, jeho dokumentace, nebo k jiným obtížím.

3.4.3 Výběr metodiky

Zvolit skutečně univerzální metodiku pro ST není možné. Přikláníme se k názorům Alistaira Cockburna [CRY/04] považovaného za otce metodik z rodiny Crystal, jenž je toho názoru, že každý projekt potřebuje jinou úroveň řízení, dokumentace a plánování, a to s ohledem na rozsah projektu, jeho závažnost a počet zapojených řešitelů. Dále si dovolujeme tvrdit, že pro každý projekt potřebujeme jedinečnou metodu, a to i za cenu mísení několika metodik. To dokazuje zkušenost z 1.RST, 2.RST i 3.RST.

Dle názorů Alistaira Cockburna je možné, nebo spíše nutné, aby se nastavená metodika upravovala pro jednotlivé projekty dle potřeb vývojového týmu. Za neméně důležité doporučení považujeme, aby se metodika pravidelně hodnotila a přizpůsobovala i v rámci jednotlivého projektu. Nenásilně tedy potvrzujeme názory Alistaira Cockburna. Z dosavadní praxe na ST usuzujeme, že podobným způsobem, jakým vznikaly agilní metodiky z jednotlivých best practices, nyní může být vytvořena vlastní metodika na základě best practices z předchozího vývoje ST.

3.4.4 Sběr požadavků a analýza

Pro sběr požadavků a jejich analýzu stále bez problémů vyhovují způsoby uváděné snad nejznámějším zástupcem iterativních metodik, metodikou RUP. Téměř v každém projektu, se kterým se kdy setkáme, budeme narážet na první potíže již ve chvíli sběru a analyzování požadavků zákazníků. Informace získané komunikací se zákazníkem vhodně zaznamenáváme a nashromážděné požadavky jím necháme zpětně revidovat. V případě ST je zákazníkem garant projektu, popřípadě mentor, ale ve velké většině případů si musíme vypomoci sami např. společnou schůzkou týmů. Při analýze lze za stále velmi vhodnou metodu považovat rozbor podstatných jmen a sloves z odborného článku, v našem případě dokumentu gamedesignu a jemu podobných. Na rozhraní mezi analýzou a návrhem stojí metody štítků, v nichž jednotlivé štítky reprezentují problémové domény a mohou se tak stát základem dalšího modelování.

3.4.5 Návrh a architektura softwarového produktu

Oblast softwarově inženýrských prací je pro projekt pravděpodobně nejzásadnější. Důvodem není nic jiného, než neustále se měnící složení týmu analytiků a programátorů a nutnost uchování informací stojících mezi dobrou analýzou a korektní implementací. Doménový model a jeho postupnou aktualizaci vytváříme ve chvíli, kdy návrh nestačí k realizaci nové užitné vlastnosti. Model by měl být pro analytiky pochopitelně sestaven, a programátory striktně dodržován. To je dle našeho názoru jediný způsob jak zajistit,

aby se v každém okamžiku mohl vývojový tým měnit a jeho další práce kotvila v pevném bodě. Nositelem diskutovaných dat je web wiki a přehledný design zdrojového kódu jádra ST.

3.4.6 Implementace

Konkrétní způsob implementace ideálně ponecháváme na týmu programátorů a řešíme v rámci jiných prací, například oborových projektů. Současně nezapomínáme na nutnost odvádět bezvadnou práci v rámci architektury již zohledňující použité technologie. Proto je potřeba kromě pozice projekt manažera obsadit v ST i pozici QA manažera, kterou v 3.RST zastával M.S. (Zastával více rolí najednou, ale bylo by vhodnější tyto pozice z důvodu náročnosti oddělit.) Během implementace považujeme za vice než vhodnou metodu dvou klobouků (z metody Scrum) nabádající programátory, aby ve chvíli, kdy realizují novou funkcionalitu, neupravovali stávající kód, ale pouze přidávali kód nový. Tím se zabrání rozšiřování objemu prací při realizaci konkrétní vlastnosti. Po vytvoření dané funkcionality vyhradíme pro QA manažera určitý čas na úpravu, neboli refaktorizaci zdrojového kódu, čímž dojde k jeho optimalizaci.

3.4.7 Testování a nasazení

Testování definujeme již ve fázi analýzy, kdy je nová užitná vlastnost softwaru považována za splněnou. Tato definice by měla být v každém případě složena z informací, co má nová vlastnost umožňovat a co naopak umožňovat nemá. Taková praxe již byla na začátku projektu zavedena a nedoporučujeme ji nijak měnit. Testování by se měli účastnit zástupci z řad programátorů a vedení projektu i sám zákazník.

3.4.8 Specifika Space Traffic

Rozdíl mezi ST a jinými projekty byl naznačen již v samotném úvodu práce. Autor chce ale nyní myšlenku dále rozvíjet, protože se jedná o rozdíly podstatné. Ačkoli se může zdát, že je ST velmi podobný jiným projektům na KIV, má několik jedinečných vlastností (na základě tvrzení autora shodně s P.V. i M.S.). ST je čistě studenský projekt, ve kterém studenti zastávají významné role projekt manažerů a architektů. Nejen implementace a návrh několika funkcionalit v rámci celého systému, ale celý projekt stojí na bedrech posluchačů ve vedoucích pozicích i mimo ně a vyžaduje značnou odpovědnost jednoho každého z nich za projekt, osobní studium a množství skrze zápočtové a jiné práce úzce spolupracujících kolegů.

Účast na tvorbě ST klade na studenty oproti ostatním povinnostem v některých jiných předmětech předem ne jasně definované, ale velké časové nároky. Pokud se student o ST do budoucna vážně zajímá, nemůže si myslet, že odvede v průběhu semestru několik činností a tím splní svůj díl práce. Projekt vyžaduje osobní iniciativu při zadávání úkolů i při potýkání se s problémy. Klíč k jejich řešení není nikým nabízen a nelze jej nalézt v materiálech studentů z minulých let.

ST nabízí nevídanou možnost růstu a realizace. Posluchač bakalářského studia do něj vstoupí jako junior programátor, ale když se stane absolventem ZČU, může získat zkušenosti i z pozice projekt manažera či analytika. Zápočty z rozličných předmětů tak nebudou shromažďovány jen „do indexu". S mírnou nadsázkou můžeme říci:

„Pokud student chce, může vysokou školu vystudovat skrze Space Traffic".

3.4.8.1 Shrnutí

S čím je třeba u Space Traffic počítat:

- Space Traffic je plně studenský projekt. Studenti programují, plánují i rozhodují.
- Od člena týmu se vyžaduje vlastní iniciativa a velká odpovědnost.
- Umožní plnit různé studentské povinnosti v rámci jednoho projektu.
- Získáte praktické zkušenosti.

3.4.9 Nezkušený student silnou posilou

Zájmu a iniciativy studentů o ST je třeba si vážit. V případě jakéhokoliv úspěchu nebo projevené snahy nezapomeňme svému mladšímu či zkušenějšímu kolegovi vyjádřit uznání. Motivujme jej. Vždyť bez jeho pomoci projekt ST nikdy nemůže existovat.

Každý student získává a spolu s intenzitou a kvalitou studia zásadně prohlubuje a zdokonaluje úroveň svých znalosti a dovednosti. Více záleží na snaze a píli než na vrozených predispozicích. Berme na tuto skutečnost zřetel především v případě začleňování a vyhledávání nováčků. Teprve s objektovým návrhem a základy programování se seznamující posluchač bakalářského oboru se na první pohled pro ST zdá být zcela neperspektivním. Ale představa přijímání pouze mimořádných osobností je šalebná, protože takové již zpravidla mívají jiné aktivity a často jim nezbývá mnoho času do konce studia.

Ale zpět k nováčkům. Přiznáváme, že jejich využití bývá v počátku velmi náročné a neefektivní. Vyžaduje přímý dohled mentora, ať už projekt manažera nebo zkušeného programátora, ale slibuje jejich vysokou perspektivnost pro ST do budoucna. Mladý student má před sebou čtyři roky studia, dostatek času poznat různé role (analytika, projekt manažera, QA manažera - testera, architekta, mentora, programátora atd.) a vnitřní procesy projektu. V neposlední řadě nachází celou řadu synergií s ostatními předměty. Vybere si vhodnou roli a v navazujícím studiu nebude tápat, ale využije svůj potenciál pro ST s možností završit účast na něm skrze diplomovou práci. (Jako v případě Z.N., R.K., nebo V.P. Dále platí, že většina studentů, kteří jednou do ST vstoupili, v něm působí i nadále.)

3.4.9.1 Shrnutí

Co mít při přijímání nováčku na zřeteli:

– Nový student je investicí do budoucna.

– Jeho počáteční znalosti a dovednosti nejsou tak důležité, jako iniciativa a zápal.

3.4.10 Motivace studentů - studentský syndrom

S otázkou, jak motivovat studenty se patrně potýkají všechny projekty a ani ST není výjimkou. Všeobecně užívanou motivací v univerzitním prostředí je odměna typu když – tak, neboli metoda cukru a biče. Za předem stanovených podmínek dojde buď k odměně (udělení zápočtu, zkoušky), nebo trestu. Různá odvětví psychologie však v současné době nabízejí teorie, že tímto způsobem postavený systém odměn odvětví vyžadujícímu celkovou kreativitu spíše škodí [DRI/09]. Programování a další softwarově inženýrské metody bezpochyby vyžadují velkou míru kreativity, a takto nastavený systém odměn lidské zdroje spíše demotivuje. Jak tuto situaci v rámci univerzitních projektů vyřešit? Mnohé podněty by pravděpodobně kolidovaly se samotným základem vysokého školství a jeho nastavenými pravidly. Otevřením této otázky však chceme zdůraznit, že nejen dostatečný počet zapojených řešitelů je odpovědí na problémy univerzitních projektů včetně ST.

3.4.10.1 Studentský syndrom černého svědomí

Na základě tříletého vývoje ST můžeme přeci jen nabídnout jistý nástroj. Možností je využití tzv. „studentského syndromu černého svědomí". Jedná se o stav vzniklý odkládáním řešení veškerých nebo většiny úkolů až na konec semestru. Díky velkému stresu a obavám z neúspěchu posléze student odvádí maximální výkony. Takové chování je zcela běžné, a snažit se je potlačit mentorováním, varováním či hrozbami mnohdy nepřivodí úspěch. Pokud však kontrolní schůzky plánujeme dostatečně často, u měsíční iterace alespoň jednou týdně, dochází u studentů ke zmíněnému efektu pokaždé před jednotlivou schůzkou. Musí se danému problému věnovat alespoň dva dny před schůzkou a třikrát v měsíční iteraci po dobu čtyř měsíců semestru. I po zohlednění volna a zkouškového období tak napočítáme kolem dvanácti skutečných schůzek. Výtěžnost lidského zdroje značným způsobem vzroste díky průběžnému získávání zkušeností a zpětné vazbě.

3.4.10.2 Shrnutí

Při motivaci studentů pamatujte na:

– Motivace studentů je značně obtížná.

– Řešením je zapojení do projektu v rámci jejich studenských povinností, které je jim umožněno plnit v rámci ST.

– Studenty nutně podrobujeme častým kontrolám, což je spolehlivě přiměje pracovat průběžně.

3.4.11 Role v projektu Space Traffic

Pro řádné fungovaní projektu doporučujeme následující model rolí. Pro představu viz obrázek 3.4. Projekt ST zobrazujeme jako pyramidu a dělíme do dvou částí. V horním patře se nachází stálé studentské vedení. Základnu tvoří externí týmy. Na celý projekt dohlíží mentor z řad vyučujících katedry.

3.4.11.1 Studenské vedení

Studentské vedení se skládá ze tří rolí - projekt manažera, QA manažera a analytika-návrháře. Role detailněji popisuje text níže. Jednotlivci společně rozhodují o směřování projektu, organizují práci externích týmů a vedou je. Studenské vedení ST je složeno z pravidla ze zkušených posluchačů navazujícího studia, ideálně i bývalých členů externích týmů. Tito studenti už často skrze účast na ST realizují své dp. Navzájem jsou si podporou hlavně v případě nejasností nebo potíží, kterých nastává v průběhu vývoje celá řada. V kritických situacích se navzájem zastupují, aby nedošlo k ohrožení existence ST. Když se navzájem dobře znají, je to další velký přínos.

3.4.11.2 Projekt manažer

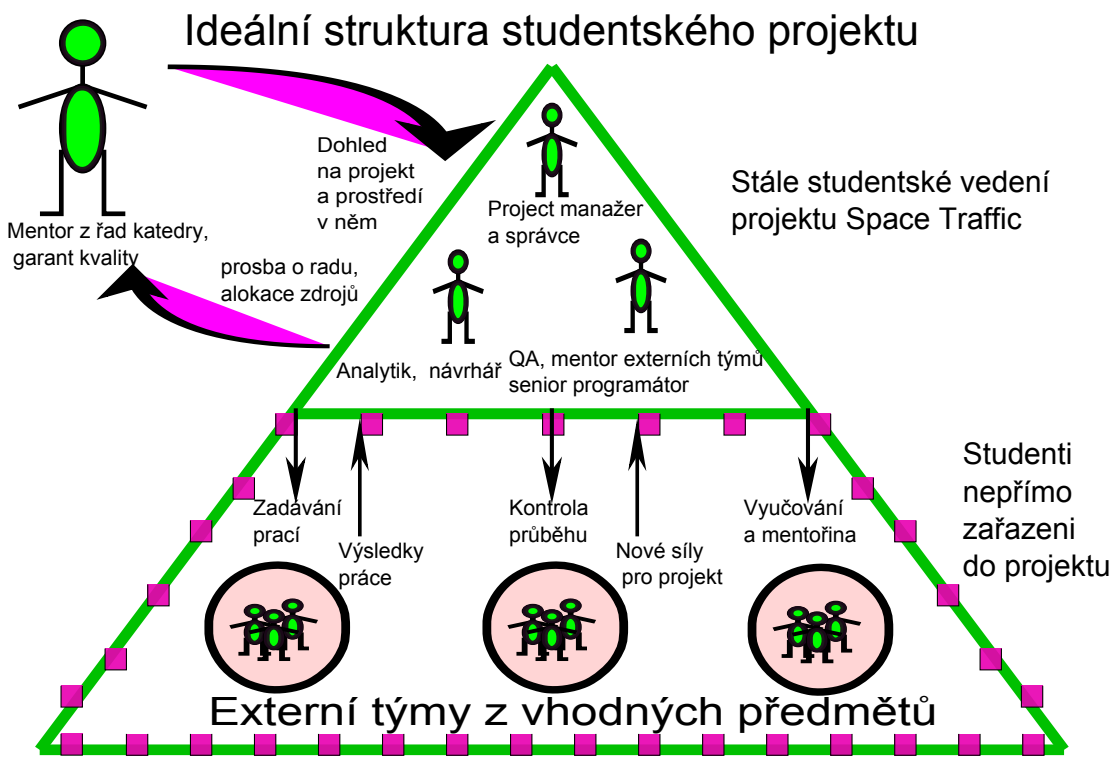
Náplní role projekt manažera je organizace spolupráce, plánování úkolu. Má poslední slovo při rozhodování o klíčových věcech. V tomto projektu bývá zmíněná pozice spojena i se zajištěným vhodných podmínek pro vývoj. Sem patří: správa SVN a uživatelských účtů; komunikace s katedrou; shánění prostředků (zajištění serveru pro vývoj, tisk plakátů apod.); odpovědnost za tvorbu dokumentace; obsah wiki; získávání lidských zdrojů. Projekt manažer spolupracuje s mentorem z řad vyučujících katedry a kontaktní osobou pro vnější svět. Jeho činnost považujeme za naprosto klíčovou a pokud selže, neovlivní to pouze jeho, dotkne se to realizace projektu jako celku, ale i studentů kteří jsou do projektu zařazení skrze externí předměty a pracují na kvalifikačních pracích.

3.4.11.3 Analytik - návrhář

Kromě povinností vyplývajících z názvu jeho pozice, tedy tvorby specifikací projektu a návrhu architektury, je pravou rukou projekt manažera. Má přehled o technologiích, zajímá se o problematiku webových her, spravuje sekcí wiki se specifikacemi a dalšími dokumenty a připravuje zadání pro externí týmy. Zastává velmi důležitou roli při stanovení cílů pro jednotlivé iterace a pro celý akademický rok.

3.4.11.4 QA manažer

QA manažer dbá všeobecně na veškerou práci od návrhu přes vývoj po výstupní kontrolu. Jeho role může být částečně kryta pozicí analytik - návrhář, což není na škodu. Student zastávající ji by ale měl mít i programátorské schopnosti a v případě většiny



Obrázek 3.4: Ideální struktura studenského projektu.

problému sloužit i jako mentor pro externí týmy. Stěžejní činností je průběžná kontrola těchto týmů a zpracovávání jejich výsledků i po stránce hodnocení, kdy připraví podklady pro cvičící, aby studenty mohli ohodnotit.

3.4.11.5 Externí týmy

Externí týmy jsou ideálním realizátorem většiny prací. Bývají sestaveny z iniciativních studentů bakalářského studia s chutí zapojit se do života na KIV. Účastí na projektu realizují své zápočtové práce a oborové projekty. Je potřeba zdůraznit, že výsledky jejich snah nejsou vždy ideální a oni sami často při nasazení potřebují pomoc QA manažera. Aby se výsledků vůbec dosáhlo vynakládá studentské vedení nemalé úsilí v podobě průběžných kontrol mezivýsledků a stálé mentorské činností. Přínosem je výchova zkušených pracovních sil a kandidátů do vedení ST.

3.4.11.6 Mentor projektu

Pokud studentskému projektu chybí mentor, riziko neúspěchu znatelně stoupá. Mentor by měl především dohlížet na všechny procesy v projektu ST, aby špatná rozhodnutí odhalil včas a aby nedocházelo k zásadním chybám daleko větších rozměrů. Ideálním kandidátem je člen katedry, nebo doktorand se zájmem o webové hry či softwarové inženýrství ochotný průběžně sledovat aktuální stav ST a v případě potřeby do něj aktivně zasáhnout. Když se

nepodaří pro akademický rok získat studenty na pozici ve vedení ST, projekt krizově řídí a zajistí tak jeho kontinuitu.

3.4.11.7 Shrnutí

Strukturu projektu můžeme specifikovat takto:

- Projekt manažer: Vedoucí studenského projektu.
- Analytik - návrhář: Správa požadavků a návrh funkcionalit. Příprava podkladů pro externí týmy.
- QA manažer: Kontrol kvality výsledných prací, mentor a hodnotitel externích týmů.
- Externí týmu: Realizátoři většiny standardních úkolů. Zdroj nových talentů.
- Mentor: Odborník z řad katedry, dohlížející na projekt. Garant projektu.

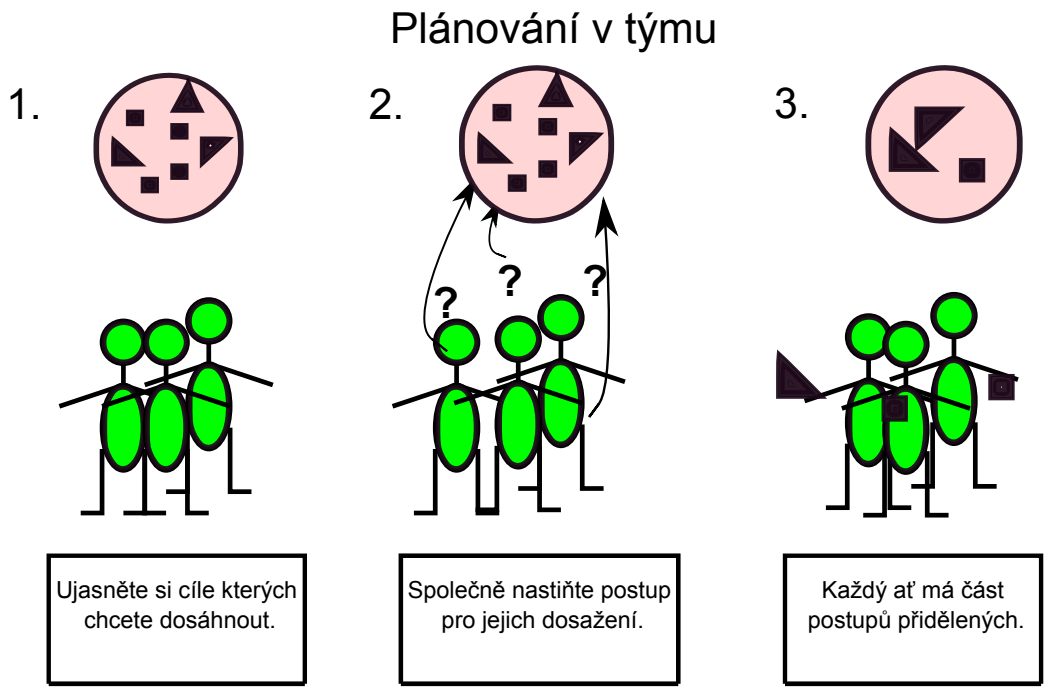
3.4.12 Plánování v týmu

Plánování úkolů nepovažujeme za triviální záležitost. Každý případ je specifický, zde si nyní nabídněme několik obecných doporučení: Před začátkem každého plánování by se měl sejít celý realizační tým. Na setkání nesmí chybět především jeho zkušenější část. Projekt manažer nebo jiná autorita představí cíle, podnítí debatu na toto téma a nadále ji moderuje. Obsah rozhovorů sleduje především nalezení řešení nutných k realizaci cílů a postupy, jak je naplnit.

Rozdělujeme povinnosti mezi členy týmu. Dobrým vodítkem nápomocným při rozhodnutí může být vystupování účastníků k jednotlivým bodům, kdy se ukáže jejich fundovanost. Pokud se rozhovor k plánu iterace nedaří rozproudit, musí moderátor diskuze klást otázky cíleně jednotlivým účastníkům. Po vytvoření letmé představy o řešení jednotlivých cílů plánu a identifikaci jejich řešitelů je setkání přerušeno, aby se každý mohl nad svým úkolem, přiděleným návrhem řešení, hlouběji zamyslet. Pro představu viz obrázek 3.5.

Představa, že úkoly ihned zadá projekt manažer studentům je špatná, stejně jako myšlenka, že je možné přímo vytvořit dokonalý detailní plán. I v případě tvorby plánu postupujeme postupným zpřesňováním a snažíme se v každém opakování o větší konkretizaci. Na počátku má projekt manažer jen cíle a mlhavou představu o způsobu řešení, které využije pří řízení diskuze. (Může mít návrh na vhodného kandidáta na konkrétní úkol atd.)

Po přerušení setkání byl měl každý ze zainteresovaných svůj úkol rozepsat a postup jeho řešení konkretizovat ještě tentýž den. Utříbí si tak svou představu, zjistí dodatečné informace. Může odhalit nedostatky v letmém návrhu řešení. V nejbližších dnech by měla schůzka pokračovat. Na ní se proberou nalezené překážky a mylné odhady. Když se objeví závažné problémy, „krokujeme zpět". Přehodnotíme rozdělení úkolů, úkoly samotné,



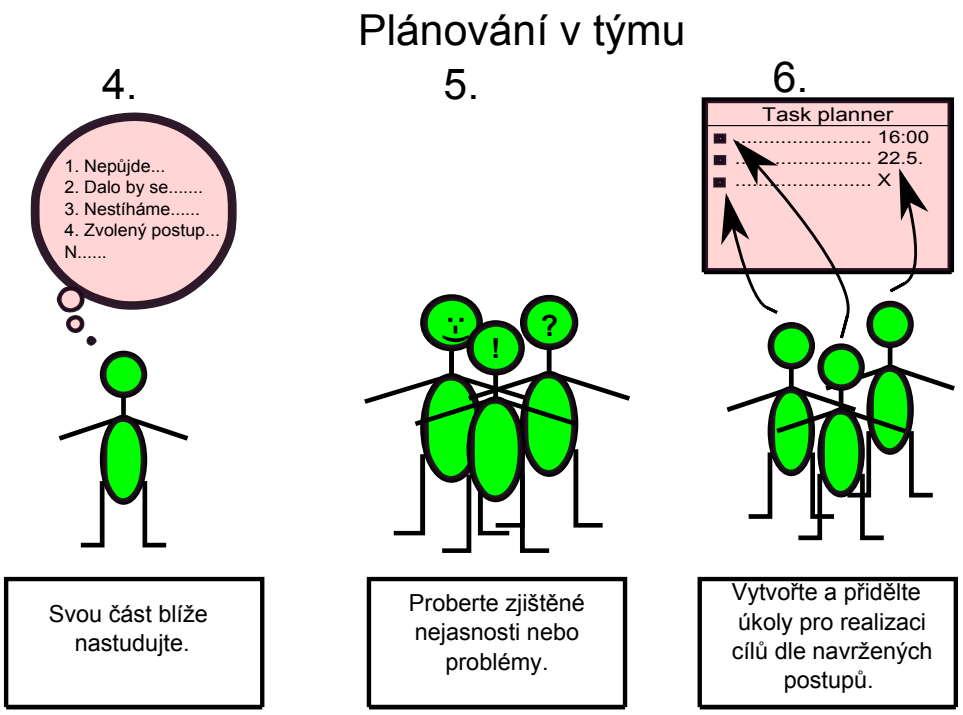
Obrázek 3.5: Plánování pracovních úkolů iterace - úvod.

v nejzazším případě pozměníme i cíle. Na závěr úkoly uložíme do nástroje pro řízení projektu, přidělíme řešitelům a na základě bližších představ provedeme poměrně přesný časový odhad jejich náročnosti. Pro představu viz obrázek 3.6.

3.4.12.1 Shrnutí

Při plánování úkolů není naškodu se inspirovat následujícím postupem:

1. Ujasněte si cíle které chcete dosáhnout.
2. Společně nastiňte postup pro jejich dosažení.
3. Každý ať má část postupů přidělenu.
4. Svou část blíže nastudujte.
5. Proberte zjištěné nejasnosti nebo problémy.
6. Vytvořte a přidělte úkoly pro realizaci cílů dle navržených postupů.



Obrázek 3.6: Plánování pracovních úkolů iterace - pokračování.

3.4.13 Různé role - různé typy úkolů

Jak jsme již řekli, v ST se uplatní celá řada studentů zastávající rozličné role. Jejich úkoly se liší nejenom svým obsahem, ale i typem. Mohli bychom je rozdělit na manažerské (také administrativní a správní) a programátorské (vytváření zdrojových kódů, analýza a návrh). Význam manažerských povinností je některými programátory bagatelizován. Někdy se dokonce na neimplementační úlohy nahlíží jako na podřadnější, jednodušší činnosti. Tento pohled je samozřejmě nesprávný. Pokud by v projektu rozsahu ST byli jen samotní programátoři, nikdy by nebylo možné jej dovést do cíle.

Postoj programátorů ale lze za určitých okolností chápat, pokud jsou nuceni do pro ně neoblíbených činností: příliš pečlivé evidence a organizování úkolů, vytváření dokumentů atd. Přistupují k nim s jistou mírou despektu. Mohou se objevit otázky typu: Řeší snad manažer složitý algoritmus? Propojil různé komponenty systému skrze složité rozhraní, nebo jen naplánoval další práci pro nás programátory? Programátoři někdy obtížně chápou, že někteří jejich kolegové, manažeři, zastávají k těmto (evidence a organizování úkolů, vytváření dokumentů…) druhům úkolu postoj zcela opačný.

Ano, za programátorem bývá vidět práce dříve a jasněji. Protože se na jeho práci nahlíží s nadhledem, jako na celek. Není hodnocena drobná práce při provázání datové vrstv s aplikační, při kosmetických úpravách grafického uživatelského rozhraní, nebo psaní komentářů ve zdrojovém kódu. Jeho hodnocen za celou jím implementovanou funkcionalitu. A manažer za řízení projektu, dohled na plánování, plynulý postup a fungující komunikaci v rámci týmu. Výsledkem manažera je dobře vedený projekt. Na druhou stranu jím není

celý výsledný produkt, manažer jej pomáhal jen vytvořit, a bez ostatních členů týmu by neobstál.

3.4.13.1 Shrnutí

Každá role v projektu má svůj význam:

- Výsledky práce programátora bývají na první pohled zřejmé.
- Projekt manažer má velké množství drobných povinností, ale jen díky nim funguje spolupráce na projektu.

3.4.14 Plánování vývoje Space Traffic

Iterace se osvědčilo plánovat na období jednoho měsíce. Vzhledem k tomu, že v měsíci prosinci a lednu studenti odvedou méně práce, doporučuje se prosincová iterace spojit s lednovou. Situaci ohledně prosince a ledna výstižně charakterizují následující slova:

Na prosinec a leden jsme navrhli menší plán. Skutečnost nás ale přesto zaskočila. Nebylo uděláno méně, ale téměř nic! Petr Vogel - Projekt manažer Space Traffic

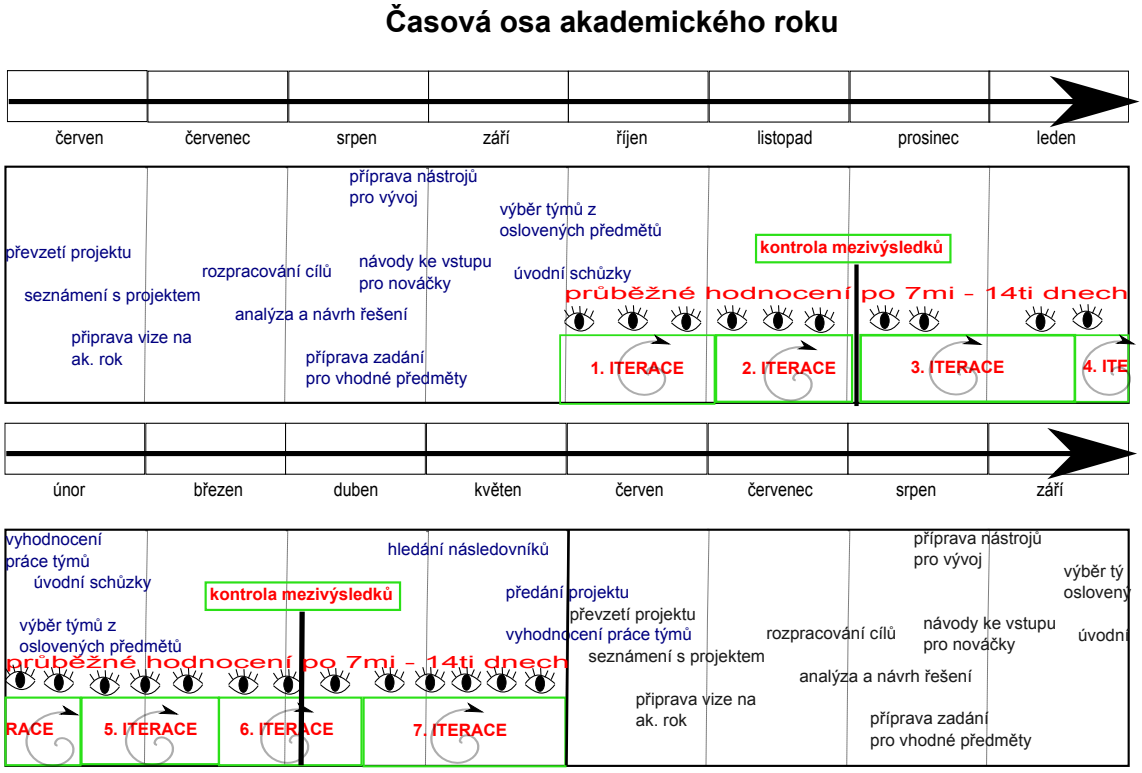
Osvědčilo se, že prvním krokem při plánování ST je zvolit si cíle. Protože se bavíme o iterativním vývoji, vytyčíme si cíle pro každou iteraci a definuje oblast funkcionalit, které je nutné implementovat. Ale jak zjistit, že je možné množinu úkolu vzešlých z těchto podkladů splnit? Bohužel na tuto otázku není jednoduchá odpověď. Neexistuje všeobecný, z knih nestudovatelný postup. Nikdo nezná splnitelnost množiny úkolů, dokud je nesplní. Jediným pomocným nástrojem pomáhajícím v odhadu, je zkušenost. Tu ale většina studentů zpočátku nemá.

Jak vidíme, plánování jednotlivých iterací bude v prostředí univerzity vždy nelehkým úkolem. Z dosavadních zkušeností i zkušeností z praxe však můžeme říci, že užitečným postupem v otázce plánování je plánovací hra taková, jakou ji popisuje metodika Scrum. Ze soupisu jednotlivých funkcionalit sestavíme něco jako je backlog, tedy seznam všech požadavků kladených na náš projekt. Pro každou iteraci pak řešitelský tým ohodnotí realizovatelné vlastnosti časovou náročností a prioritou. Zákazník nebo vedení týmu ohodnotí jednotlivé vlastnosti hodnotou „důležitost" a ze všech těchto atributů následně určíme, kterým vlastnostem se při realizaci budeme věnovat v následující iteraci. To nám také dávám prostor pro změnu priorit, i informaci o důležitostí samotných funkcionalit během projektu a náš postup se tak stává více či méně agilní.

3.4.14.1 Začněte brzy

Prvním krokem na začátku projektu, po seznámením se s ním, má být vytvoření vize ST pro akademický rok. A to již v průběhu června a července, v každém případě před zahájením vývoje. Na základě vize a cílů z ní vzešlých, navržení i řešení těchto úkolů. Dále

je nutné vytipovat vhodné předměty pro nábor studentů a připravit zadání. Studenty bude nutné rychle zapojit, a tak se nesmí podcenit ani příprava podkladů pro nováčky, včetně nástrojů nutných k vývoji projektu. Pro lepší představu obrázek 3.7.



Obrázek 3.7: Doporučený průběh vývoje Space Traffic v rámci akademického roku.

Na konci září proběhne nábor a série úvodních schůzek. Následuje výběr první sady úkolů pro implementaci zvolené funkcionality, a pokud se zjistí, že odhad nebyl reálný, v listopadu se přizpůsobíme, v prosinci taktéž... Metoda pokus omyl. Je nutné provádět průběžné schůzky, alespoň třikrát v měsíční iteraci. Při ní proběhne kontrola práce a zajišťujeme tak průběžné plnění studenských povinností. Do půlky semestru je vhodné pro externí týmy nastavit jakýsi milník, kdy musí být schopni odevzdat mezivýsledek.

Je potřeba zdůraznit, že pokud nezačneme implementovat dostatečně brzy, nebude možné získat potřebné zkušenosti, a to se promítne do celkového stavu projektu. Pokud přípravu podceníme, pozdější náprava nebude možná (na nápravu nebude čas). Např. pokud začneme plánovat a plnit iterace až v listopadu (jako v případě 1.RST a 2.RST), tým se „nesehraje" ani nezíská dostatečnou jistotu. Pokud uvažujeme, že efekt sehranosti a dobrého odhadu přichází po třech měsících (závisí na předchozích zkušenostech), dostavil by se až v průběhu února, protože všeobecně platí, že prosinec s vánoci a leden se zápočtovým a zkouškovým obdobím jsou měsíci, ve kterých bývají studenti zaměstnáni mnoha dalšími povinnostmi, které narušují plynulý běh projektu.

Na začátku letního semestru se musí provést hodnocení externích týmů a provést nábor pro předměty letního semestru. A vše se opakuje. V závěru akademického roku, který bývá

zpravidla pro vedoucí projektu zkrácen, protože ukončují své studium, je nutné nezapomenut na hledání následovníku a předání projektu v utěšeném stavu.

3.4.14.2 Rozdělte práci v týmu

Nejde jen o plán pro realizaci cílů celé iterace. Zkušenost často chybí i v řešení, jak jednotlivé podoblasti vybraných funkcionalit rozdělit do úkolu z pohledu jejich počtu a rozsahu. I když subjektivně čelíme jednoduchému a drobnému úkolu, který jsme však nikdy před tím nerealizovali, musíme se soustředit a být pozorní. I instalace programu trvající odhadem pět hodin času se po provedení často ukáže jako zcela chybná, a může trvat pětadvacet hodin. Pro studenta majícího plnit časový plán se jedná o výrazný časový úsek. Úkoly pokud možno rozdělujeme dle typu. Pro manažera, analytika a správce bude programátorský úkol několikanásobně náročnější než pro zkušené kodéra, což se promítne i v trvání realizace. Je tedy dobré, aby jednotlivé úkoly specifikoval a zanášel do plánovače „odborník na daný problém". A tím samozřejmě většinou není projekt manažer.

V týmu potřebujeme jak manažery, tak hlavně programátory a soft. architekty, odborníky, pro realizaci náročných úkolů. Proto úkoly nemůže specifikovat jen manažer, a o to je méně logické, aby je zadával do systému jen on. Manažer ale musí mít stále na paměti, co se má do konce iterace realizovat[6]. Úkoly tak nakonec zadávají všichni většinou sami pro sebe. Samozřejmě to ale není absolutní pravidlo, manažer dbá na to, aby se na žádnou oblast nutnou ke splnění iterace nezapomnělo, aby si někdo neplánoval moc málo, nebo naopak více, než lze objektivně stihnout.

3.4.14.3 Řešení obtíží

V průběhu tvorby přicházíme na podněty, nápady a chyby, které je dobré zachytit a později zpracovat. K tomu slouží tzv. backlog, do kterého tyto typy úkolů zanášíme, ale nemusíme hlouběji specifikovat, kdo je má provést a v jaké iteraci[7]. Téměř vždy na konci iterace zbude část úkolů, které se nepodařilo realizovat. Není potřeba zoufat. To je věc zcela normální. Ale tento efekt nemá mít rostoucí trend, který by svědčil o chybách při plánování. Faktory jako náhlá nemoc části týmu a jiné neočekávané události není možné predikovat.

3.4.14.4 Získejte maximum - zkušenosti

Bylo by chybou za výstup iterace považovat jen kvantum odvedené práce. Pokud iteraci hodnotíme dle kritérií úspěšnosti vycházejících z jejích cílů, musíme si dát tu práci a analyzovat i to, co bylo důvodem nesplnění či zrušení některých úkolů, nebo jejich přesunutí do další iterace. Musíme zhodnotit, zda se jedná o faktory, jež jsme mohli odhadnout, se kterými jsme se setkali už dříve, a uvážíme, proč se nám nepodařilo zjednat nápravu. Pak

[6]Samotný výběr co se má realizovat, ale byl určen už před začátkem iterace, třeba na společném brainstormingu, kde měli slovo všichni, ale veto měl projekt manažer.

[7]Ale měli bychom jej kontrolovat vždy při plánování iterace nové, jestli se v něm nenachází úkol související s námi stanoveným cílem nové iterace.

i neúspěch přinese přidanou hodnotu, a ze získaných zkušeností mají přínos všichni. S úkolem následně naložíme dle aktuální potřeby vložením do backlogu, naplánováním na další iteraci atd.

3.4.14.5 Shrnutí

Vzpomeňte si při plánování projektu ST:

1. Ujasněte si vizi[8] projektu na celý akademický rok.
2. Vyberte podmnožinu cílů, které si myslíte, že stihnete v průběhu jedné iterace.
3. Pokuste se rozdělte si mezi sebou úkoly nutné k realizaci této podmnožiny.
4. Pokud se Vaše odhady ukázaly mylné, analyzujte důvody.
5. Při plánování další iterace vyjděte z předchozích zkušeností.

3.4.15 Zadání diplomových prací - uvědomte si

Diplomové práce jsou nejúčinnějším motivačním nástrojem pro studenta zapojeného do ST. A to především z toho důvodu, že si je vědom její důležitosti. Dp. patří mezi základní předpoklady pro úspěšné ukončení studia. Jednoduše tak můžeme dojít k závěru, že zadáme studentovi dp., kde uvedeme konkrétní cíle, které je nutné v daném roce splnit, a tím zabezpečíme úspěšné pokračování ST. Tento předpoklad je, ale mylný a skrývá se v něm nemalé nebezpečí.

Nejprve si je třeba uvědomit proč vize a cíle projektu pro daný rok, nejsou to samé jako cíle v obsahu zadání dp. konkrétního studenta, ať se jedná o projektového manažera, analytika-architekta, programátora apod. Projekt je ovlivněn řadou nepředvídatelných faktorů. Povede se pro daný rok sehnat dostatečné množství studentů? Jak budou studenti aktivní, iniciativní a jak rychle jim půjde práce od ruky? Nedojde v průběhu roku v ST k zasadím změnám v jeho směřování? Pokud tyto faktory nastanou mohou změnit i celou výslednou podobu projektu. Ale cíle dp. studenta změnit nejdou či jen velmi obtížně. Student se pak nachází v nezáviděníhodné situace, kdy musí naplňovat své úkoly v rámci ST a dále odvádět činnost pro svou dp. V nejhorším případě jdou tyto úkoly proti sobě. Student je ve velkém časové tísni a přednost má samozřejmě absolvování studia než projekt ST. Pokud se nachází např. v pozici projekt manažera ovlivňuje jeho postoj celý projekt daleko závažněji a může jej i citelně poškodit. K této situaci u ST už došlo nejednou.

3.4.15.1 Případ Zbyňka Neuderta

V případě Z.N. nebylo možné sehnat dostatečně velký realizační tým. Dále se až v průběhu roku ukázalo, že projekt je velmi obsáhlý a jeho realizace bude trvat několik let[9]. V zadání

[8] Představu s vytyčenými cíli.

[9] To je i poslední odhad P.V. a M.S., úspěšných řešitelů z 3.RST.

jeho diplomové práce se, ale předpokládalo, že hra se podaří z velké části v 1.RST implementovat. Když se ukázalo, že se ve skutečnosti jedná o utopickou představu, Z.N. měl oprávněnou obavu o následnou obhajobu své práce. Tlačil tedy na svého jediného programátora, aby vytvořil verzi, kterou bude možné prezentovat. Programátor se tedy soustředil pouze na výsledek, ale nebyl čas pro promyšlení návrhu nebo dokumentaci, a to později způsobilo nemalé obtíže. Zde je potřeba zdůraznit, že se nejednalo o chybu nebo nekvalitní výkon, ale jen důsledek faktu, že nebylo v silách dvou jednotlivců takové množství práce stihnout. Kdyby nebylo při zadání práce Z.N. počítáno s dokončením implementace, jehož předpoklad ani nemohl být objektivně odhadnut, protože nic podobného nebylo dosud realizováno, mohl být výsledek zcela jiný. V průběhu roku by se zjistilo, že představa byla mylná a v rámci iterativního vývoje by se vytvořilo jen jádro webové hry. Programátor by měl čas vytvořil pevný a propracovaný základ, postavený na dobrém návrhu. Velikost týmu by ovlivnila jen množství prací, ale ne kvalitu výsledku a jeho využitelnost do budoucna.

3.4.15.2 Případ Richarda Kocmana

U R.K. se v mnoha aspektech problém opakoval. Na začátku 2.RST se nevědělo v jakém stavu je zdrojový kód webové hry. Trvalo přes měsíc než se problém plně analyzoval a další měsíc než se našla odvaha problém řešit. Opět zde zásadní roli hrál obsah dp. R.K., kde se počítalo s tím, že projekt je plně rozběhnout a pro implementaci několika funkcí bude možné webovou hru nasadit do rutinního provozu.

3.4.15.3 Případ Petra Vogla a Martina Štěpánka

M.S. situaci v 2.RST zažil, viděl nesnáze které to přineslo R.K. a byl si tedy dobře vědom obtíží, které může špatná forma zadaní dp. projektu ST, jako i studentu způsobit. Podělil se tedy se svými zkušenostmi s P.V., a když padlo konečné rozhodnutí, že na projektu budou pokračovat, obsah jejich zadání to reflektoval.

3.4.15.4 Shrnutí

Před specifikací zadání diplomové práce je důležité si uvědomit následující:

- Důkladně se seznamte se stavem projektu před sepsáním jednotlivých bodů zadání.
- Stanovte si jen ty cíle, jejichž realizaci můžete přímo ovlivnit.
- Buďte velmi opatrní při vytváření bodů, které se podmiňují[10].
- Snažte se, aby jednotlivé body určovaly rámec Vaší práce, ale ne obsah (rámec sděluje, čím se musíte zabývat, ale umožňuje Vám naplnit jej až na základě Vámi odvedené práce).

[10]Např. nemůžete nikdy popisovat nasazení webové hry, pokud se v daném roce nepodaří hru do vhodného stavu dovést.

– Vyhýbejte se bodům s obsahem pro projekt ne přímo přínosným. V ST můžete uplatnit celou řadu znalostí a dovedností, ale pokud si vyberete dovednosti, které nebudou pro ST přínosné, může se stát, že Vaše práce nebude ostatními „ceněna"; a ve špatné atmosféře se pracuje o poznání hůře.

4 Implementace webové hry

V následující kapitole se pokusíme zhodnotit projekt za období tří minulých let především z hlediska návrhu, implementace a použitých technologií. V jejím závěru pak zodpovíme dvě otázky vycházející ze zadání této práce, a to dovedení hry do rutinního provozu a zhodnocení webové hry s ohledem na původní záměr.

4.1 První rok vývoje 2009-2010

První období vývoje webové hry ST vedl Z.N. Vyznačovalo se velmi malým týmem, což se projevilo i na rozsahu výsledného projektu. Pro implementaci bylo zvoleno PHP, a vzhledem k obtížím při pokusu o navázání na tuto implementaci lze na výsledný produkt pohlížet pouze jako na prototyp. Zabývejme se nyní jednotlivými nedostatky řešení a zamysleme se, jak je možné se z nich poučit do dalších období. Jako ilustrace nám může posloužit i sada obrázků viz příloha 6.1.

4.1.1 Pokus o třívrstvou architekturu

Jak již bylo zmíněno, pro 1.RST byl vybrán jazyk PHP a řešení bez jakéhokoliv frameworku, což ztížilo vývoj. Z pohledu výsledného kódu lze těžko říci, jak mnoho je na vině původní návrh, nebo nedostatek času. Vidíme ale, že se kódem postupně šíří rozdíly ve zvoleném řešení. Z hlediska programování byl zvolen objektový model s pokusem o architekturu model view-controler (více dále). Zpracování požadavků probíhalo značně chaotickým způsobem. Každá jednotlivá stránka v závislosti na parametrech buď sama zpracovávala požadavky, nebo je přenášela na metody třídy stejného jména, a případně v rámci těchto metod ještě dále (viz příloha 6.1(5)). Někdy docházelo dokonce k dělení funkcionality částečně přímo v souboru PHP a částečně v rámci jednotlivých tříd. Z tohoto důvodu by v případě potřeby jakékoliv změny muselo dojít k pátrání, kde by se měla taková změna provést.

Autoři se pokusili zavést třívrstvou architekturu model-wiev-controler. Bohužel funkcionalita jednotlivých vrstev v některých místech prorostla mezi sebou a původně zamýšlené oddělení tak ztratilo jakýkoliv význam. V případě nedodržení třívrstvé architektury je kamenem úrazu view. Ačkoli by měl mít podobu HTML šablon s některými doplňenými vygenerovanými hodnotami, v případě řešení projektu se jedná o změť echo příkazů generujících výsledný HTML kód. Viz příloha 6.1 (3). Ty jsou navíc doplněny celou řadou dalších, které generování kódu znepřehledňují a patří do jiných vrstev architektury. Minimálně v jednom případě došlo místo k vypisování pouze pomocí příkazu echo ke směšování klasického HTML a PHP kódu. Celým projektem se prolíná problematika nekonzistence v názvech. Viz příloha 6.1 (2). U některých tříd jsou navíc verze pro přihlášeného a ne-

přihlášeného uživatele.

4.1.2 Použití kaskádových stylů a JScript

Dalším problémem se stalo používání CSS. Přesto, že je součástí projektu CSS soubor, v rámci kódu používáme celou řadu různých řešení. Můžeme najít volání funkce GenerateStyleString stejně tak jako výpis jednotlivých vlastností vepsaných přímo do dotyčného tagu HTML. V případě nutnosti provést jakoukoliv změnu bychom museli složitě pátrat, jakou metodou je zrovna tento prvek formátovaný. Někdy nastávají problémy s postupným děděním CSS vlastností. Mixování řešení by jenom přidělávalo další. Za nejsprávnější postup můžeme považovat nadefinování jednotlivých tříd pro dané prvky a jejich užívání v rámci šablon. Tyto styly by se měly striktně definovat v oddělených CSS souborech, aby případné změny, např. barev, mohl provádět i někdo jiný, než programátor pátrající v rámci PHP kódu.

Další významnou částí funkcionality je JavaScript. Popis hlavní logiky byl vepsán do jednoho rozsáhlejšího souboru. Tento soubor se snaží pospojovat jednotlivé funkce bez specifické struktury, nijak nectí architekturu programu. Postupné volání funkcí a registrace zpětných volání může čtenáře uvést ve zmatek. Bylo použito minimum komentářů, spíše žádné. Případnému pokračovateli projektu tedy seznámení se s kódem a případné provedení změn pravděpodobně zabere zbytečně mnoho času. I pro samotné autory muselo být po určité době obtížné přidávání další funkcionality.

4.1.3 Zhodnocení

Nyní jsme v podstatě kriticky zhodnotili proces implementace. Přesto byla tato fáze pro projekt přínosná. Pokud hledíme na výsledný produkt shovívavěji - jako na prototyp, můžeme pominout fakt, že hra obsahovala celou řadu chyb. Například ztrácení lodí, načítaní pohybu každé planety z textového souboru s přepočítanými souřadnicemi atd. Viz příloha 6.1(4). Další vývojáři měli možnost prohlédnout si grafické rozhraní (viz příloha 6.1(1)) a základní myšlenku logiky hry. Vývoj v této fázi v podstatě ukázal směr, kam se ubírat, ale ne způsob, jak se tam ubírat.

4.2 Druhý rok vývoje 2010-2011

Již v předchozí části bylo řečeno, že na začátku školního roku 2010-2011 stál nový vývojový tým R.K. před problémem, jak pokračovat. Po prozkoumání a bližším analyzování implementace bylo jasné, že kód je třeba přepsat. Předchozí vývoj posloužil jako velmi propracovaný prototyp, který odhaloval některé nedostatky prvotního návrhu, ale nesplňoval dvě základní hlediska. Značně nepřehledný programový kód nedržel žádnou strukturu. A za druhé grafický návrh aplikace byl nedostatečný a ukrytý v po kódu roztroušených příkazech echo. Nový tým tedy nejprve řešil otázku, v čem bude hra do budoucna implementována. Po prvním roce vývoje došlo k poznání a shodě, že čisté PHP je pro vývoj

nepoužitelné, a má - li zůstat zachováno, je nutné vybrat vhodný framework. O jiné platformě se již neuvažovalo, protože řešitelský tým byl vybrán pro platformu PHP. Jako framework nakonec posloužilo týmem již dříve odzkoušené Nette.

4.2.1 Důraz na grafické uživatelské rozhraní

Dále se budeme podrobněji zabývat oblastí designu grafického rozhraní. Tým musel především projít stávající grafické rozhraní, odhalit jeho nedostatky a navrhnout ho tak, aby bylo konzistentní a splňovalo všechny potřebné vlastnosti. Uživatelské rozhraní považujeme za klíčový prvek pro fungování hry; část, s níž se uživatel setkává v první řadě a která ho provází po celou dobu hry. I kdyby byl herní princip sebedokonalejší, bez vhodného řešení grafického rozhraní by uživatel hru přestal hrát. Nebavíme se zde o barevném designu, i když i ten je důležitý, ale ovládání hry jako takovém. Výsledný produkt musí mít vlastnosti jako intuitivní a snadná ovladatelnost, jednoznačnost, konzistentnost a pochopitelnost. Pro nezkušené hráče by měla být k dispozici okamžitá nápověda.

4.2.2 Prototyp - zdroj inspirace

Prototyp obsahoval celou řadu nejednoznačností. Například management lodí byl dostupný ze dvou různých míst (ikona, menu), ale jiné položky tyto vlastnosti neměly. V pravé části GUI se v případě nezvolení akce zobrazovaly možnosti podobné stavovému panelu. Ambasádě se dala zvýšit úroveň jen v tomto místě. Menu pozbývalo logické struktury; při zobrazování finančních obnosů chyběly oddělovače tisíců; nebylo možné zjistit, kde se nacházejí hráčovy lodě nebo červí díra, aniž bychom na ni klikli. Dále přehledu lodí a základen chybělo jednotné ovládání; u lodi na cestě označení místa, kam míří a u názvů planet spojení se jmény hvězd, aby se daly rozlišit. Při zadávání rozkazu lodi se neobjevil její stav a pokud jsme ji vybrali, vizuálně se posunula; lodě se objevovaly v hvězdě; pokud jsme je chtěli koupit nebo rozkliknout jejich management, panely se otevíraly ve vrstvách a bylo třeba je postupně zavírat, zároveň ale nebylo možné zobrazit již otevřený panel a mezi záložkami nešlo přepínat; ikona planety s prstenci byla deformována a stávalo se, že v GUI zůstávaly informace o planetě, v jejímž hvězdném systému jsme se ovšem v dané chvíli již nenacházeli.

4.2.3 Nový velmi detailní návrh

Nyní se budeme věnovat jednotlivým částem uživatelského rozhraní a v něm navrženým změnám. Každý uživatel ST se nejprve setká s přihlášením a registrací. Bylo rozhodnuto, že k původnímu přihlašování přibyde volba automatického přihlášení. Dále došlo ke zrušení záložkového designu textů. Registrace hráčů měla probíhat tak, aby zpočátku stačilo zadat jen nejnutnější údaje v podobě e-mailu, hesla či přezdívky. Další, žádoucí pro tvorbu statistik, by uživatelé vyplňovali až v průběhu hry za určitou odměnu. Tým zamýšlel plynulý přechod registrace v tutoriál a poté samotnou hru. Proto rozhodl, že povinnost

aktivovat účet pomocí e-mailu může být splněna do jednoho týdne. K zablokování hry by došlo až po nedodržení termínu.

Původní grafika hry se skládala z pěti panelů (viz příloha 6.1(1)). Kontextový panel není vidět:

- Stavového panelu.
- Menu panelu.
- Hlavního panelu.
- Kontextového panelu.
- Panelu událostí.

Stavový panel leží v horní části GUI a zobrazuje vybrané informace o stavu hry. Například o počtu lodí, základen, zpráv, peněz, aliancí a kontaktů s piráty (koncept pirátů a aliancí byl v 3.RST zcela opuštěn). Problémem původního panelu byla značná nepřehlednost. Některé údaje (např. počet lodí) téměř postrádaly význam a jiné chyběly (pozice lodí ve slunečních soustavách). Jednotlivé položky měly posloužit i rychlé navigaci (klikací odkazy), s níž se počítalo již v původním návrhu. Od kontextové nápovědy se očekávala adresnost. Některé původní údaje byly z panelu vypuštěny.

Hlavní menu v původním návrhu obsahovalo položky:

- Koupit loď.
- Koupit licenci.
- Management lodí.
- Pořadí hráčů.
- Aliance.
- Mapa galaxie.

Tato podoba menu v žádném případě neplnila svůj účel. Jednotlivé volby byly nesystematicky uspořádány a implementovány různým způsobem. Rovněž příliš dlouhé názvy položek zbytečně zabíraly místo. Navíc dva různé názvy začínající stejným slovem komplikovaly orientaci. Proto bylo nutné menu od základu přepracovat a zvolit systém, ve kterém žádná položka neprovádí akci přímo, ale otevírá okno hlavního panelu.

Původní návrh čítal osm voleb, ale všechny nakonec při implementaci návrhu použity nebyly.

- Události (viz příloha 6.2(2)).

– Lodě.

– Základny.

– Zprávy.

– Diplomacie.

– Mapa.

– Hodnocení.

– Nápověda.

Mezi panely byl navržen ještě kontextový panel (viz příloha 6.2(4)). Ten by měl poskytovat informace o vybraných objektech a volby pro manipulaci s nimi. V původním návrhu zobrazoval informace o hráčově společnosti (duplicitní informace), úrovni ambasády s možností tuto úroveň zvýšit a doručovaných zásilkách.

Dále došlo k implementaci ve hře úplně chybějících funkcionalit a opravě implementací funkcionalit již stávajících. Jsou to tyto:

– Přehled událostí.

– Detaily lodi.

– Detaily flotily.

– Přehled základen.

– Detaily základny.

– Detaily planety.

Přehled událostí by měl zobrazovat informace týkající se hráče a mít podobu krátkých zpráv rozlišených ikonami a barvou. Bylo třeba roztřídit události (nové, staré, podle dnů) a do přehledu lodí zařadit záložky lety, seznam lodí, flotil, nákup a správa licencí. Seznam letů by měl zobrazovat lodě a letky na cestě a k tomu odpovídající volby, seznam lodí zobrazovat podobnou tabulku jako seznam letů a zobrazení mít formu tabulky. Za podstatné považujeme především údaje jako stav, náklad, cíl a doba letu a příletu. Vedle již zmiňovaných vlastností by tabulka zobrazovala informace jako typ, model a cena lodi. Mělo by být možné jednotlivé lodě seskupovat do flotil podle různých kritérií. Detaily lodi jsou součástí jiného zobrazení, o kterém se rozhodlo, že bude možné pro každou loď zobrazit. Nákup zobrazuje seznam lodí k nákupu a jejich ceny (viz příloha 6.2(3)). Dle představ vývojářů by byl seznam řazen podle typu lodi nebo ceny; detail lodi poskytoval veškeré informace o lodi a její historii a čítal kompletní seznam vlastností (jméno, druh lodi, typ lodi, datum nákupu, cena, příslušnost k letce, poškození, náklad, rozkazy, poslední událost lodi) a akcí dostupných pro danou loď a detail flotily poskytoval především výčet jednotlivých lodí ve flotile a rozkazy flotily.

4.2.4 Implementace návrhu a zhodnocení

Pro implementaci tým využil presentery Nette, jim odpovídající šablony phtml (viz příloha 6.2(6)) a statické CSS soubory. Pro některé ještě neexistující grafické prvky použil placeholdery. Implementované presentery odpovídaly položkám menu. Jednotlivá zobrazení byla reprezentována akcemi presenterů. Šablonové soubory Nette obsahovaly HTML s doplňujícím značením. Každé akci odpovídala jedna šablona psána jako XHTML. CSS soubory sloužily pro všechny šablony a názvy tříd a elementů musely být voleny tak, aby nedocházelo ke konfliktům. Jednotlivé styly byly ukládány do samostatných souborů, jejichž název odpovídal názvu ovládacího prvku nebo prezenteru. Navíc se používaly tři specifické CSS soubory:

- Development.css : pro vývoj aplikace.
- Cssmap.css : v době vývoje mapuje všechny aktivní css soubory.
- Common.css: styly společné pro celou aplikaci.
- Styly s příponou dim obsahující rozměry a pozicování u fungující aplikace nastavované přes javascript.

Byl vytvořen layout, který představoval základní HTML stránku. Ten definoval kostru bloků vyplňujících šablony konkrétních akcí prezenterů. Hlavní layout dělíme do tří oblastí:

- Header : výběr jazyka a prvky související se hrou.
- Content: zobrazení hry.
- Footer : informace o copyrightu a další informace.

Content se dále skládal ze čtyř oblastí: topPanel, menuPanel, mainPanel a contextPanel. Tyto oblasti odpovídají oblastem herního uživatelského rozhraní. Výstupem období 2.RST byla implementace GUI, které zobrazovalo základní funkční navigaci a ukázková data a vytvoření základní kostry, na níž by v budoucnu probíhala implementace. V dalším období 3.RST došlo k přechodu k jinému programovacímu jazyku implementace, a tak bylo tohoto cíle dosaženo skrze něj. Především díky kvalitnímu návrhu došlo k velmi rychlému přepsání zdrojových kódů (viz příloha 6.2(5)), HTML šablon a datových objektů do .NET. 2.RST období mělo na vývoj výsledné aplikace nemalý podíl.

4.3 Třetí rok vývoje 2011-2012

Zatím posledním obdobím vývoje hry ST je období implementace založené na platformě ASP .NET. Na jeho počátku se zásadně přehodnocovaly základy vývoje a činila závažná rozhodnutí (měna architektury a nová platforma). Za důležitý milník považujeme právě rozhodnutí přejít z PHP Nette používaného v rámci druhého období na ASP.NET MVC.

4.3.1 Změna platformy

Také díky Ing. Petru Vaněčkovi Ph.D. se implementačním jazykem stal Csharp, který se v současné době rozšiřuje v rámci studia na FAV, a tak je vedle Javy studenty hodně užíván. Tento fakt umožnil nábor vývojářů z různých předmětů, protože dnes již během studia příliš nepoužívané PHP je v podstatě až zatracováno. Dá se říci, že pouhé PHP je pro projekt tohoto rozsahu a požadované funkcionality naprosto nedostatečné. A ani PHP rozšířené o některý framework nemůžeme považovat za vhodnou volbu, protože nedostatek studentů vývojářů ovládajících tento jazyk by značně ztěžoval až znemožňoval vývoj webové hry. Díky nasazení ASP .NET MVC bylo možné ušetřit velké množství implementačního kódu. Dalším problémem PHP oproti novému řešení byl i problém implementace aktivního serveru, o němž hovoříme dále (GameServer).

4.3.2 Architektura a struktura výsledné webové hry

ASP .NET MVC byl uvolněn firmou Microsoft v roce 2009. Jedná se o rozšíření klasického ASP .NET frameworku, který implementuje návrhový vzor model-wiev-controler. Tento framework umožňuje vytváření webových aplikací složených ze tří částí. Model zachycuje stav modelovaného světa. Tato vrstva často spolupracuje s databází. View představují komponety zobrazující uživatelské rozhraní. Controller má za úkol zpracovávat povely od uživatele, modifikovat model a vybírat pohledy (view), které mají být zobrazeny (viz příloha 6.3(3)). Pohledy zobrazují pouze informace.

Aby tým dostál celé funkcionalitě webové hry, jak byla popsána v dokumentu ST gamedesign, bylo nutné celkový projekt rozdělit na několik částí (struktura projektu viz příloha 6.3(1)) a vytvořit webovou aplikaci (klienta), s níž budou interagovat jednotliví uživatelé. Ta byla vytvořena právě s pomocí MVC a nazývá se GameUI. Protože se herní svět mění v reálném čase, bylo třeba vytvořit serverovou aplikaci rovněž existující v reálném čase a postupně modifikující databázi podle vývoje herního světa. V tomto případě se jedná o konzolovou aplikaci s názvem GameServer. Třetí částí se stala Core, což je knihovní podprojekt implementující datové a servisové kontrakty a společné rozhraní jako je parsování xml a podobně (viz příloha 6.3(5)).

GameServer, neustále běžící konzolová aplikace, komunikuje s uživatelským rozhraním bezstavově. Jejím hlavním úkolem je udržovat aktivní model světa a zajišťovat perzistence dat, tj. komunikuje s databází. V případě, že GameUI potřebuje nějaká data, tak se na ně dotazuje právě GameServeru. Díky tomu, že je oddělen GameServer od GameUI, je možné obě aplikace nasadit každou na jiný fyzický server (možností je samozřejmě nasadit obě na stejný). Teoreticky lze vytvořit jen jednu aplikaci a GameUI s GameServerem spojit, ale vzhledem k potřebě neustálé aktualizace herního světa by muselo na pozadí běžet vlákno, které by toto neustále provádělo. A zůstává otázkou, zda by takové řešení nepřivedlo do projektu další problémy. Díky oddělení je možné mít více fyzických serverů. Kromě již zmíněných dvou ještě třetí databázový server.

Z hlediska třívrstvé aplikace MVC představuje GameServer model s perzistencí. Naproti tomu webová aplikace GameUI controller a view. Model je zde zastoupen jen velmi

málo. Má podobu formulářů a výměn dat mezi webovou aplikací a herním serverem. V této komunikaci není posíláno vše, ale pouze relevantní data. Příkladem výměny dat může být posílání spojení červích děr. View je představováno z předchozího roku převzatými (překonvertovanými) HTML (PHP nette) a také dalšími dodělánými šablonami (viz příloha6.3(2)). Jedná se o statické stránky, do nichž controller doplňuje hodnoty. Dále jsou zde controllery pro herní akce (např. posílání zpráv, zobrazení mapy) a pro jednotlivé položky menu.

Významnou částí GameUI je JavaScript (viz příloha 6.3(4)). Jedná se o renderovací jádro. V rámci cyklu dochází k aktualizaci pozic vykreslujících se objektů (př. vykreslování planet obíhající hvězdu). V rámci implementace jsou připraveny podklady pro interaktivitu. Zatím je funkční pouze kliknutí na červí díru a řešené i další interakce, ale výhledově se počítá s dokončováním v dalších letech. Součásti JavaScriptu jsou i JavaScriptové knihovny a v rámci JavaScriptu implementovány i takové věci jako překreslování za běhu (na základě změny stavu na serveru), nebo zvětšování-zmenšování. V tomto případě je však velice obtížná práce s kolečkem myši, a tak ovládání JavaScript zatím nebylo implementováno nebylo. Core definuje mimo jiné herní objekty. Zatím byly naimplementovány jen již použité (např. planeta, hvězda). Představují jednotlivé objekty, které jsou vykreslovány. O to se stará vykreslovací engine (Javascript).

4.3.3 Zhodnocení

Na testování nezbylo příliš mnoho času. V dalších letech by bylo vhodné, aby se osoba v roli QA manažera o tuto problematiku postarala. V tomto období zaznamenáváme snahu o zavedení junit testů (viz příloha 6.3(6)), ale pokrytí je velmi malé. Neexistovala osoba, která by je vytvořila, neboť ve studijním plánu je na junit testy kladen malý důraz. Mnohdy ani není reálné věnovat jim prostor vzhledem k rozsahu jednotlivých implementací. Proto funkcionalita a testování v podobě junit testů byla opomíjena. K testování docházelo, ale na jiných úrovních, například pomocí grafické aplikace. Docházelo však k chybám typu proměnlivé vstupy (např. aktuální čas počítače).

Co se týmu podařilo dokončit? Jednoznačně byla vytyčena kostra celé aplikace, na které budou následující roky studenti stavět (pokud se ovšem zařídí dle jim odkázaných instrukcí). Odhadem projekt potřebuje další tři roky intenzivní práce. Vývoj může pokračovat pomaleji, než bychom třeba čekali, ale cílem projektu není pouze vytvořit funkční webovou hru, ale také prostor, ve kterém se studenti učí v rámci pracovního postupu správně používat metodiky vývoje softwaru (i za cenu nižší rychlosti). Byla dokončena správa uživatelů a lokální pohyb (lze procházet pomocí červích děr). Zatím se nepodařilo vyrobit mapu vesmíru jako celku. Do konce semestru by měl být dodělán pohyb lodí a jejich ovládání pomocí textových seznamů, kam má loď letět. Jsou připraveny hlavičky metod obsahující komentář, co by měly dělat, ale čekají na implementaci. Současný tým se zatím nezabýval nasazením aplikace do reálného provozu. Bude třeba vyřešit, jakým způsobem aplikace poběží, zda se nasadí na virtuální stroj, jestli bude databáze oddělena, jakým způsobem dojde k řešení aktualizace systému, postup při nasazení nové verze, odstavení aplikace a systém zálohování.

4.4 Dovedení realizace hry do rutinního provozu

Jedním z bodů zadání diplomové práce, kterým se budeme nyní zabývat, je dovedení realizace hry do stavu vhodného pro rutinní provoz. Snahou autora této práce v zimním semestru 2.RST připravit vše proto, aby mohl být tento bod plně naplněn. To bylo částečně i podmínkou toho, aby došlo k naplnění i dalšího bodu zadání.

Proto první aktivitou, kterou se vývojový tým zabýval hned po jeho sestavení, bylo seznámení se stavem implementace z 1.RST. Při tom však došlo k zásadnímu zjištění, že kvalita implementace a hlavně návrhu nebyla ani zdaleka ve stavu, na který bylo možné přímo navázat. Nebylo jiného řešení, než implementaci webové hry provést od začátku. Toto řešení ale autor jako vedoucí týmu viděl jako nejzazší možné a v první iteraci se snažil tým vést k naplnění zadání cestou postupné refaktorizace stávajícího kódu, který by se dal rozšířit o nové funkcionality, jak bylo původním záměrem. Ale jen se ukázalo, že cesta úprav stávajícího kódu je zcela neschůdná. Po konzultaci s garantem ST (zároveň vedoucím diplomové práce), na které se seznámil se skutečným stavem implementace převzatého projektu ST, bylo rozhodnuto, že implementaci skutečně bude nutné provést znovu.

Pokud bychom se dívali na tento úkol zadání jako na nasazení webové hry (výsledek softwarového projektu ST) do rutinního provozu, pak by tento bod zadání splněn nebyl. Ale jak se ukázalo v 3.RST, byl tento úkol v rámci akademického roku nebo v horizontu 1-2 let nerealizovatelný. Pokud bychom ale tento bod zadání chápali jako dovedení projektu vývoje webové hry do rutinní provozu, je tento bod bezesporu splněn (vývoj úspěšně pokračuje i v 3.RST). Vývoj byl prováděn ve větším týmu než v 1.RST, dva z vývojářů, kteří vstoupili do projektu 2.RST v projektu pokračovali i v 3.RST, kdy jeden z nich (M.S.) na významné pozici analytika-návrháře. V 2.RST, kdy projekt vedl autor, byl vytvořen návrh grafického uživatelského rozhraní na základě prototypu z prvního roku vývoje, implementovány grafické šablony a ty (i další podklady a zkušenosti) použity při vývoji v dalším roce realizace ST.

4.5 Zhodnocení vlastností webové hry

Druhým úkolem vycházejícím z dalšího bodu zadání je zhodnocení vlastností webové hry s ohledem na původní záměr na základě zkušeností s dostatečně dlouhým provozem. Když se na tento bod podíváme z hlediska vývoje projektu, je zřejmé, že se původní návrh webové hry v průběhu třech let vývoje příliš nezměnil a je stále v rámci vývoje naplňován. V současné době dochází k implementaci pouze jedné malé části původního návrhu (jádro, tedy hvězdné systémy a pohyb mezi nimi) a tato část je rozpracovávána do hloubky. Jsou řešeny i oblasti jako registrace a přihlášení do hry. To tedy neznamená, že by docházelo k zasadím změnám navržených funkcionalit, nebo dokonce revizi původního záměru (vytvoření webové hry pro propagaci KIV). Právě naopak. Je pozorován trend zpřesňování a rozšiřování původního návrhu, který ale ctí a staví na tom původním. V budoucnu tak bude vývoj jednotlivých funkcionalit postupně zpřesňovat původní návrh a naplňovat původní záměr výsledky z vývoje.

Je tedy evidentní, že původní gamedesign byl navržen dobře, ale bude postupně dále rozšiřován a prohlubován. Evidentní také je, že nasazení webové hry do provozu, nebude mít za následek ukončení projektu. Na základě zpětné vazby od uživatelů, bude odhalena celá řada chyb a projekt může úspěšně pokračovat dále na pracích týkajících se hry Space Traffic.

Na projekt je zapotřebí se dívat a hodnotit jej jako víceletý. Studenti si postupně předávají mezi jednotlivými lety vývoje celou řadu zkušenosti, a to jak díky diplomovým pracím (kde je možné se dočíst o předchozích postupech a dále se poučit pro další roky), tak i spoluprací při vývoji na skutečném softwarovém produktu.

5 Platforma IBM Jazz a její nástroje

V této kapitole si probereme platformu IBM Jazz [JAZZ/10] a její nástroje, které byly také použity v projektu ST. Dozvíme se, že platforma nabízí velmi silné nástroje, které pomáhají při vývoji udržovat pevný řád, ale na druhou stranu složitost těchto nástrojů znesnadňuje jejich nasazení do projektů, kde působí i méně zkušení studenti.

5.1 Co je IBM Jazz

V roce 2008 začala firma IBM vyvíjet projekt s názvem Jazz. Snahou iniciativy Jazz je usnadnit vývoj softwaru pomocí větší spolupráce, produktivity a transparentnosti. Toho dociluje pomocí integrace informací a úloh v rámci celého vývojového procesu. Iniciativa Jazz se skládá ze tří částí: platformy, nástrojů a komunity.

5.2 Platforma

Jazz představuje inovativní přístup integrace. Na rozdíl od monolitických platforem je tato otevřená, umožňuje zlepšit vývojový cyklus a překonat bariéry mezi nástroji, a poskytuje technické základy pro integraci několika typů nástrojů pro vývoj softwaru. Platforma se skládá z architektury, sady aplikačních frameworků a nástrojů. Jazz je postaven na několika základních principech.

5.3 Architektura IBM Jazz

Jedná se o architekturu klient-server. Implementace nástrojů je oddělena od definic a přístupu k datům. Data jsou nezávislá na jednotlivých produktech a uložena v databázích, ke kterým se přistupuje pomocí HTTP protokolu přímo v úložišti. Není zapotřebí je importovat-exportovat, aby bylo možné je využít v jiném nástroji, než v jakém byla vytvořena. Jazz je navržen pro použití otevřeného flexibilního datového modelu. Jednotlivé nástroje lze vyvíjet na základě práce s HTTP protokolem. Jazz nijak neurčuje frameworku, jak mají být jednotlivé nástroje implementovány. Navíc podporuje mnoho technologií. Umožňuje použít například klienty na bázi Eclipsu. Serverová část (Jazz Team Server) je webová aplikace implementována v jazyce Java EE 1.4.

5.3.1 Aspekty architektury IBM Jazz

Existují dva základní aspekty architektury Jazz. Open Services for Lifecycle Collaboration (dále jen OSLC) a Integrační služby. OSLC pracuje na principu poskytovatel-spotřebitel. Poskytovatel je nástroj vlastnící data, která poskytuje ostatním nástrojům na základě OSLC specifikace. Zákazník je nástroj přistupující k datům jiného nástroje prostřednictvím specifikovaného rozhraní. Tento způsob komunikace umožňuje sdílení dat mezi nástroji. Místo transformace dat z jednoho nástroje k druhému a jejich zdvojení se prostřednictvím OSLC rozhraní pracuje jen s jedněmi daty.

Integrační služby dále posilují spolupráci nástrojů. Některé poskytují schopnosti, které mohou být užívány všemi nástroji (např. správa uživatelů). Jiné služby jsou implementovány v některých nástrojích a pro jejich spolupráci (např. tabule).

5.4 Pracovní postupy v IBM Jazz

Jednotlivé artefakty jsou uloženy v úložišti, ke kterému přistupují autorizovaní uživatelé. Úložiště obsahuje oblasti projektu odkazující na jednotlivé artefakty. Každá oblast projektu má přidružený proces, který upravuje chování platformy. Je možné vybrat, jaký proces bude užíván. Platforma Jazz nabízí několik předdefinovaných: Agile, Eclipse Way, Scrum, OpenUp a Simple, nebo můžeme použít vlastní. Každý proces je nakonfigurován a definuje chování během iterací.

Oblasti projektu jsou dále děleny mezi jednotlivé týmy. Uživatel může být členem více týmů. Aktivita probíhá v rámci vývojové linie. Jednotlivé práce jsou plánovány pomocí pracovních položek. Ty jsou řazeny podle kategorií. Práce probíhají v rámci iterací. Pomocí vytvoření plánu iterace lze naplánovat práci pro danou iteraci. Veškeré práce se soubory daného uživatele probíhají v pracovním prostoru úložiště. Server sleduje veškeré změny pomocí sad změn. Slouží tedy jako systém pro správu a verzování zdrojových kódů. Tyto soubory ale ještě nejsou součástí společného projektu. K tomu slouží proud projektu, který používá celý tým. Je tedy možné provádět změny postupně a zveřejnit je až najednou. Proud taktéž sleduje veškeré změny. Dále může mít každý tým sestavení popsané pomocí definice sestavení. Sestavení lze provádět na různých strojích. Pro komunikaci mezi uživateli slouží tzv. kanály. Pokud je na nějaký artefakt napojen kanál, v případě jeho změny dochází k automatickému rozesílání upozornění.

5.5 Komunita

Díky tomu, že je platforma Jazz neustále ve vývoji, vytvořila se okolo ní komunita vývojářů. Je možné přímo komunikovat s vývojovým týmem nebo s ostatními uživateli. Novinky jsou zveřejňovány v rámci blogu, na twitteru, nebo třeba na konferencích. Lze stahnout trial nebo beta verze nových produktů a případně zanechat připomínky.

5.6 Nástroje IBM Jazz

IBM Rational a partneři vyvinuli řadu produktů využívajících platformu Jazz. Ne každý z nich však využívá všech vlastností architektury. Vzhledem k tomu, že platforma se stále vyvíjí, je dokonce nepravděpodobné, že by nějaký produkt využíval vše. Ale právě tuto vlastnost považujeme za silnou stránku architektury Jazz. Můžeme totiž využívat jen její podmnožinu, kterou potřebujeme, na rozdíl od tradičních komplexních API (rozhraní pro programování aplikací z angl. Application Programming Interface). Následující produkty byly vyvinuty na platformě Jazz:

- Rational Team Concert: Pracovní prostředí pro spolupráci vývojářů, architektů a manažerů projektu s unifikovanými pracovními objekty, kontrolou zdrojových kódů, řízením projektu a plánování iterací.
- Rational Quality Manager: Na webu založené přizpůsobitelné prostředí pro řízení testování. Slouží pro plánování testů, kontrolu postupů, zaznamenávání a vyhodnocování.
- Rational Requirements Composer: Řešení pro efektivní definování požadavků. Umožňuje jejich využití v průběhu vývojového postupu.
- Rational Insight: Nástroj pro automatické měření postupu programu a projektu.
- Rational Build Forge: Přizpůsobitelný framework, který automatizuje, řídí a sleduje procesy a jejich předávání v rámci vývojového postupu.
- Rational Asset Manager: Knihovní systém, s jehož pomocí je možné katalogizovat, organizovat, používat, znovu využívat, spravovat a reportovat jakýkoliv typ aktiv od obchodních, přes technologické až po softwarové.
- Rational Rhapsody and Rational Software Architect Design Management: Návrhářská řídící struktura integrována do celé aplikace a systému vývojového postupu. Umožňuje spolupráci týmů na návrzích napříč organizacemi, obory a doménami.

5.6.1 Rational Team Concert

Rational Team Concert [RTCJ] (dále jen RTC) je nástroj sloužící k týmové spolupráci v rámci vývojového cyklu softwaru. Z historického hlediska byl RTC prvním produktem postaveným na platformě Jazz. RTC umožňuje přímou výměnu informací. Pokud dojde k nějaké změně, např. zadání, ostatní členové týmu (ale i další zainteresované osoby) jsou automaticky informováni. Na data existují různé pohledy. Je možné sledovat aktivity jednotlivých členů týmu a informace prezentovat. Tento nástroj integruje mnohé aspekty vývojového cyklu jako jsou plánování iterací, definice procesu, správa zdrojů, testování chyb, správa sestavení, nebo vytváření sestav. Všechny jednotlivé aspekty jsou integrovány v jednom prostředí. Projekty v rámci RTC se řídí jedním vybraným procesem. Výběr tohoto procesu tedy upravuje chování produktu.

Proces definuje role, postupy, pravidla, oprávnění a pokyny používané v rámci projektu. Proces můžeme nastavit předdefinovaný, nebo upravit na míru podle potřeb organizace. Jednotlivé komponenty proces sdílejí. Pravidla zadáváme ve tvaru podmínka akce. RTC umožňuje práci prostřednictvím webového rozhraní nebo pomocí rozhraní klienta na bázi Eclipse. Webové rozhraní je spíše vhodné pro administraci, prohlížení dat v úložišti či aktualizaci úloh. Klient na bázi Eclipse je vhodný pro sestavování a dodávání jednotlivých artefaktů. V následující části se postupně seznámíme s jednotlivými vlastnostmi RTC.

5.6.1.1 Procesy a nastavení

RTC umožňuje nastavení prostředí ve smyslu procesních postupů organizace, ve které je nasazen. Místo toho, abychom neustále v dokumentaci hledali, jaký krok bude následovat, RTC sám kontroluje a automaticky detekuje jakékoliv vybočení z postupu. Je možné nastavit jednotlivé role a jejich hierarchii v rámci projektu. Každý projekt má svůj vlastní proces, což je důležité, pokud organizace vyvíjí více projektů najednou. Navíc každý proces může být v rámci projektu dále upraven. Lze jej upravovat pomocí nastavení práv, podmínek a akcí pro různé operace, jako jsou např. pracovní položky, modifikace oblastí týmů či nastavení sestavení. Jednotlivé role jsou definovány procesem a je i možné je vytvářet v rámci projektu. Práce definujeme pomocí pracovních položek. Každá oblast projektu má definováno hierarchii iterací, které určují běh projektu. Práce mohou probíhat buď v jednom časovém proudu, nebo je možné jich provádět více souběžně (např. vývoj a údržba).

5.6.1.2 Týmové povědomí

RTC umožňuje práci na úrovni týmů, jejich organizaci a vzájemnou informovanost o artefaktech, na nichž jednotliví členové pracují. Uživatelé si mohou nastavit prostředí tak, aby viděli jen týmové artefakty, které je opravdu zajímají. Jednotlivé týmy lze utvářet z již existujících, nebo nových uživatelů. Vybraní obdrží pozvání do týmu, které musejí přijmout. Automaticky jsou vytvořeny tři informační kanály zasílající informace o změnách: vytvoření události v oblasti týmu; vytvoření události pro více týmů (mezi které tým patří) a událost, která se dotýká určité pracovní položky. RTC dále podporuje týmovou synchronizaci. Například chat nebo zasílání souborů.

5.6.1.3 Sledování pracovních položek

Pracovní položky jsou základním mechanizmem RTC pro řízení úloh a pracovních postupů. Mimo to slouží k mnoha dalším účelům, např. podpoře integrace s jinými produkty. Pracovní položky lze měnit, aby lépe odpovídaly používanému procesu. K úpravám lze využít webové rozhraní a klienty na bázi Eclipse. Vlastnosti položek jsou dobře formátovatelné a definovatelné na základě jiných aktuálně vybraných vlastností; a může být použita validace. Pracovní položky tagujeme a dle uvážení rozšiřujeme o obrázky, nastavujeme schvalovací proces a tiskneme. Jsou podporovány i šablony umožňující přednastavení výchozích hodnot. V případě potřeby utvoříme předdefinované dotazy, které upraví jed-

notlivé pohledy na položky. RTC vyhledává i případné nechtěné duplicity a sleduje změny jednotlivých položek.

5.6.1.4 Správa zdrojových kódů

Systém pro správu zdrojových kódů, komponentově založená verze systému kontroly, je navržen pro podporu týmů nenacházejících se na stejném místě a paralelní a agilní vývoj softwaru. Tato komponenta podporuje různé programovací techniky, jako vývoj založený na komponentách, nebo vývoj řízený vlastnostmi(FDD). Ukládací model tvoří sady změn. Sada změn může být sdílena, pozastavena, zrušena nebo vrácena. Jednotlivé změny jsou vedeny v rámci proudů.

5.6.1.5 Plánování

Plánování je v RTC navrženo tak, aby bylo možné využívat jak tradiční, tak agilní metodiky. U agilních metodik vytváříme backlogy, plány pro jednotlivé vývojáře a sledujeme postup v rámci iterace. U klasických modelů RTC zabezpečuje jednotlivé závislosti v rámci rozvrhu prací a poskytuje editor pro tvorbu plánu projektu. Nezávisle na typu procesu mají k plánu přístup všichni členové týmu a lze jej měnit na základě aktuálních potřeb vývoje.

5.6.1.6 Průběžné sestavování

Team Build komponenta integruje systém poskytující kontrolu, informovanost a sledovatelnost celému týmu. Členové mohou sledovat postup jednotlivých sestavení a být informováni o výsledcích, problémech a požadavcích. Podpora sestavení je navržena tak, aby spolupracovala s existujícími systémy (např. Ant.) a nesnažila se je nijak nahradit. Sestavení lze plánovat. Můžeme rovněž vytvořit soukromé sestavení na základě kódu, který vývojář ještě nedal k dispozici ostatním členům týmu. Je možné sestavení porovnávat, budovat o nich reporty a zasílat o nich informace prostřednictvím elektronické pošty.

5.6.1.7 Reporty o projektu

Součástí RTC jsou komponenty poskytující přehled o zdraví projektu. Jedná se o sadu různých, v rámci procesu konfigurovaných reportů nabízejících informace o historických trendech v sestavení, proudech, pracovních položkách a dalších artefaktech. K více jak padesáti předdefinovaným reportům je možné nakonfigurovat vlastní. Lze je zobrazovat ve formátu HTML, ale i exportovat do dalších formátů (např. PDF, Excel, Word atd.). Data jsou ukládána do speciální databáze a je možné v nich analyzovat různé trendy a na jejich základě provádět změny v postupu.

5.6.1.8 Administrace

Nástroj Team Server je webové rozhraní pro nastavení a administraci. Nastavení mohou provádět autorizovaní administrátoři a lze jej aplikovat bez nutnosti restartování serveru. Administrátorům se dále zobrazují klíčové statistiky a aktuální požadavky. K dispozici mají sadu wizardů (typ uživatelského rozhraní představujícího sled dialogových oken vedoucích uživatele pomocí řady přesně definovaných kroků) pro běžná nastavení, jako je např. konfigurace databáze, emailové upozornění, správa uživatelů nebo LDAP. Jedná se o protokol k ukládání dat (např. o uživatelích) uchovávaných ve stromové struktuře. Team Server může používat externí LDAP. Díky tomu, že společnost již tento systém využívá pro správu uživatelů, nebude zapotřebí udržovat dvě databáze uživatelů.

5.6.1.9 Zabudované integrace

RTC je integrován spolu s dalšími produkty platformy Jazz, jako jsou např. Rational Quality Manager nebo Rational Requirements Composer. Spolupráce konkrétně s těmito dvěma produkty staví na základě Collaborative Lifecycle Managementu (CLM). Rational ClearQuest využívá pro synchronizaci ClearQuest Synchronizer. Dalším nástrojem synchronizovatelným s RTC je Rational ClearCase používající ClearCase Synchronizer.

5.6.1.10 Rational Team Concert – praktické zkušenosti

Subjektivní názor na práci s nástrojem Rational Team Concert utvářelo několik hledisek. Instalace RTC klienta spolu s vývojovým prostředím Eclipse je poměrně snadnou záležitostí, nicméně doinstalování klienta do již fungujícího prostředí s sebou přináší různá úskalí v konfiguraci obou nástrojů a výběru správných verzí instalovaných pluginů. K obtížnému pochopení všech souvislostí navíc přispívá nepřehledné webové prostředí serveru jazz.net nabízející ke stažení různé verze tohoto nástroje.

Produkt RTC v podstatě patří mezi velmi silné nástroje, a i když snahou společnosti IBM je prosadit ho v malých vývojových týmech, domníváme se, že malé týmy režie strávená konfigurací ve srovnání s jinými konkurenčními produkty jako je Flyspray nebo Redmine značně zatěžuje. Tyto bezplatné produkty poskytují malým týmům a projektům dostatečnou podporu správy požadavků a týmové komunikace. Po vlastních zkušenostech tedy nástroje platformy IBM Jazz můžeme doporučit především pro projekty časově a počtem řešitelů rozsáhlé.

Další, z našeho pohledu nešťastně řešenou záležitostí je sdílení a verzování souborů, které nejsou přímou součástí výsledného softwarového produktu. Jinými slovy, nejedná se o zdrojové kódy, ale například podpůrné dokumenty, jenž se v průběhu iterací mohou drobně lišit. Takové soubory je nutno zpracovávat lokálně a poté je do sdíleného úložiště pracně nahrávat.

Určitou výhradu máme i k doprovodné úloze webového prostředí. Chápeme, že produkt RTC je předurčen ke spolupráci přímo s vývojovým prostředím Eclipse, nicméně webové prostředí plní spíše náhledovou funkci a nemá všechny funkce pluginu v Eclipse. To může

nové uživatele značně mást.

Složitost vytýkána produktu RTC je relativní. Z praxe můžeme potvrdit, že nároky kladené na uživatele a především administrátory nástrojů platformy IBM Jazz bývají srovnatelné s konkurenčními produkty (např. Microsoft Project Server a Team Foundation Server). Nicméně v praxi si můžeme dovolit nástroje poměrně dlouhou dobu zavádět, a zaškolovat zainteresované osoby, což v univerzitním prostředí nelze. Hlavní překážku všeobecného užití těchto nástrojů spatřujeme v časté obměně studentů účastných běžících projektů a z toho plynoucí nemožnosti je řádně zaškolit do práce s požadovanými nástroji. K řešení situace by bylo nutné věnovat na univerzitě nástrojům platformy IBM Jazz větší pozornost a jistým způsobem nutit posluchače pracovat s nimi během drtivé většiny projektů napříč několika předměty. Celou záležitost ovšem komplikuje fakt, že do magisterského studia bývají přijati studenti jiných univerzit, kteří nástroje neznají.

Uživatelé RTC dodržují určitý řád, což je jistě ku prospěchu věci. Za jeden z nejvýznamnějších přínosů považujeme skutečnost, že změny prováděné ve zdrojovém kódu nelze nahrát do úložiště, aniž by bylo zaznamenáno, k jakému požadavku a jaké pracovní položce se daná změna vztahuje. V nejzazším případě operativně vytvoříme novou pracovní položku a ukládanou změnu kódu k ní přiřadíme. RTC, velmi mocný nástroj, pomáhá udržet práci programátorů v předem stanoveném rozsahu.

5.6.2 Rational Requirements Composer

Rational Requirements Composer [RRCJ] (dále jen RRC) slouží jako nástroj pro definování a řízení požadavků ve vývojovém cyklu softwaru. Umožňuje spolupráci široké škály zúčastněných stran od zákazníků přes analytiky a vývojáře až po testery. Tento nástroj definuje požadavky pomocí textových i vizuálních definic. Mmohou být řízeny, organizovány, analyzovány a použity v rámci vývojového procesu pro informování a podporu důležitých obchodních rozhodnutí. Nástroj RRC slouží ke komplexnímu zachycení požadavků kladených na softwarové dílo v jakékoli fázi jeho životního cyklu. Díky webové platformě navíc umožňuje zapojení celého řešitelského týmu, zástupců zákazníka, a všech ostatních zainteresovaných osob (Stakeholders).

Velikou výhodou tohoto nástroje je i možnost specifikování požadavků v textové a grafické podobě včetně možnosti vzájemného provázání forem. Obecně tedy lze říci, že v nástroji RRC požadavky komponujeme od prvotní myšlenky až do finální podoby, členíme do podprojektů a s ohledem na jejich formu, zachycení a účel zakládáme do vytvořených podskupin. K dispozici máme rovněž slovník pojmů (Glossary). Jedině tak může být zaručena jednoznačnost konkrétní definice požadavku mezi členy vývojového týmu a především mezi týmem a zákazníkem.

5.6.2.1 Typy artefaktů

U jakékoli nově založené informace do RRC bychom měli zadat její typ. Nutně uvádímei druh zadaného artefaktu. Takový postup v konečném důsledku veškeré informace

zpřehlední a umožní jejich snadnější a konkrétní vyhledávání. RRC nabízí různé typy artefaktů. Nejdůležitějšími z nich jsou však dle autorova názoru tyto:

- Actor, hráč, aktér: Tento typ artefaktu definuje aktéry v systému. K prostému pojmenování můžeme připojit i detailní popis. Artefakt pochopitelně používáme při tvorbě diagramů případů užití.
- Bussines Goal použijeme pro popis obchodních a strategických cílů vyvíjeného software. Volitelnou položkou atributu je hodnota Satisfied by, do které můžeme umístit odkaz na jiný typ artefaktu pokrývající daný obchodní cíl.
- Bussiness Process Diagram: Tímto artefaktem graficky znázorníme firemní procesy, a můžeme zde samozřejmě zachytit, které kroky tohoto procesu budou obsluhovány naším softwarovým produktem a které budou řešeny mimo hranice systému.
- Feature: Jedná o textový popis požadované užitné vlastnosti softwarového produktu.
- Non Functional: Tímto typem atributu zaznamenáváme nefunkční požadavky na náš softwarový systém. Můžeme sem řadit jakékoli vnější vlivy nebo omezení, která nesouvisí přímo s poskytovanými službami, ale týkají se například prostředí, ve kterém bude software nasazen.
- Screen Flow: Tímto diagramem můžeme znázornit konkrétní představu o sledu akcí v uživatelském interface. Například pro projekt elektronického obchodu zde můžeme znázornit konkrétní sled prvků v GUI při objednávání určitého produktu.
- Term: Termín je velice důležitým typem artefaktu obzvláště při komunikaci se zákazníkem a uvnitř týmu. Každý termín je dílčí jednotkou slovníku, díky kterému se vyhneme případům, ve kterých si každý člen vývojového týmu nebo skupiny stakeholders představí pod pojmem „uživatel" jinou osobu obsluhující systém.
- Use Case Diagram: Jedná se o klasický diagram případů užití, jak jej definuje UML. Definuje, kteří uživatelé či aktéři pracují se systémem, a jak. Pomáhá definovat i hranice systému a může zachycovat i operace vykonávané mimo systém. Jsou srozumitelné jak pro vývojáře tak pro zákazníky, a tím usnadňují vzájemnou komunikaci.
- Use Case Specification: Tímto artefaktem zaznamenáváme konkrétní podobu práce uživatele se systémem. Velmi často je specifikace zachycena formou scénáře popisujícího, co se se systémem při daném typu uživatelské operace děje. V závislosti na okolních podmínkách můžeme definovat i alternativní scénáře pro jeden případ užití. Příkladem z prostředí elektronického obchodu může být například operace potvrzení objednávky mající rozdílný scénář pro registrovaného a neregistrovaného uživatele.

5.6.2.2 Atributy

Atributy v RRC podávají dodatečné informace o jednotlivých artefaktech. Hlavními atributy jsou:

- Business Priority: Definuje váhu, kterou danému artefaktu přiřazuje zákazník.
- Status: Udává stav vytvořeného artefaktu. Artefakt může být například ve stavu prvního konceptu (draftu), může být právě revidován nebo již schválen.
- Difficulty: Udává odhadovanou složitost; nízká, střední, vysoká (low, medium, high).
- Stability: Tento atribut slouží k podání informací, jak veliké změny ve specifikaci daného artefaktu předpokládáme; jinými slovy, jak je artefakt stabilní (low, medium, high).
- Origin: Atribut podává informaci kým byl daný artefakt vytvořen.

5.6.2.3 Skladba požadavků

Základní formou resp. formátem k zachycení požadavku je obyčejný text. V případě nástroje RRC se však jedná o bohatý text (Rich text), jehož součástí mohou být hypertextové odkazy na další způsoby zachyceného požadavku jako jsou např. diagramy případů užití, uživatelské příběhy (Uses cases diagrams, userstories-storyboards), obrázky nebo emaily. Vzájemná provázanost všech forem pomocí odkazů je nesmírně důležitá k tomu, aby všechny zainteresované osoby plně porozuměly specifikaci daného požadavku a předešlo se tak zbytečným komplikacím během realizace nebo i po realizaci (implementaci) daného požadavku.

5.6.2.4 Typy vazeb

Požadavky lze mezi sebou provazovat. Pro vetší přehlednost a další funkcionality:

- Satisfied by: Udává, kterými artefakty je nahlížený artefakt uspokojen, neboli splněn. Například jediný bussines goal lze splnit několika užitnými vlastnostmi (Features) a některé features mohou být řešeny ve více případech užití, nebo jim mohou patřit různé toky uživatelského rozhraní (Sceen flow). Tato vazba nám umožní najít požadavky vedoucí ke splnění vytyčených cílů a současně dává do rukou nástroj zabraňující vytváření požadavků, které nikdo nechtěl. V tomto ohledu často chybují začínající analytici.
- Embedded in: Pomocí této vazby je možné jednotlivé artefakty do sebe vnořovat.
- Link from: Odkazuje na artefakt, který se sám na právě nahlížený artefakt odkazuje.

– References Terms: Za velmi užitečnou vazbu považujeme vazbu references terms umožňující odkazovat na pojmy projektového slovníku přímo z dalších artefaktů. Je tím opět zaručena jednoznačnost a přehlednost, která ve fázi sběru požadavků hraje zásadní roli.

5.6.2.5 Možnosti kategorizace v RRC

Kromě typů artefaktu dalšímu usnadnění třídění a pozdějšímu vyhledávání informací a konkrétních požadavků slouží možnost štítkování, neboli tagování. Tento způsob třídění informací se v prostředí internetu v dnešní době těší stále větší oblibě. Štítkování navíc skýtá možnost označit požadavky kladené na funkcionalitu jádra, nebo požadavky, které by měly být řešeny v konkrétní iteraci.

5.6.2.6 Rational Reguirements Composer – praktické zkušenosti

O práci s nástrojem Rational Requirements Composer jsme se již zmiňovali. Nyní se však na tento nástroj podíváme z jiného úhlu pohledu. Jak jej nainstalujeme? Za nezbytně nutné považujeme instalaci RRC i RTC a dalších nástrojů na jeden IBM Jazz server. V opačném případě není možné jednoduchým způsobem propojit například požadavky z RRC s pracovními položkami RTC. Nicméně takového stavu v univerzitním prostředí docílíme obtížně, neboť spuštěná platforma Jazz s nainstalovaným RRC plně vytíží stroj s operační paměti 4GB a na užívaném serveru znemožní provozovat jiné aplikace. K tomu také došlo při instalaci RRC na virtuální počítač poskytnutý KIV pro projekt ST. Z toho důvodu se obáváme, že plnohodnotné nasazení všech nástrojů platformy IBM Jazz nebude v univerzitním prostředí nijak jednoduché a v každém případě považujeme za nutné nasazení i následnou konfiguraci do detailů promyslet a navrhnout. Proces samotné korektní instalace RRC na IBM Jazz server je další překážkou k masovému rozšíření tohoto nástroje. Situaci příliš nenapomáhá ani interaktivní instalační příručka, která po zadání vstupních parametrů generuje instalační návod. Celou sadu nástrojů by bylo nutné instalovat a administrovat specialistou na jednom zvláště vyčleněném serveru KIV.

Nasazení všech IBM Jazz nástrojů na jeden server nebylo v druhém roce vývoje ST možné z důvodu řešení otázky pokračování projektu a nutnosti oboustranně velmi časově náročné spolupráce mezi administrátory KIV a projekt manažerem ST, pro kterou nebyl ani z jedné strany dostatečný prostor. Nedošlo ani ke zvážení možností vzájemného napojení nástrojů. Ve třetím roce vývoje projekt manažeři zvolili pro studenty známější a jednodušší vývojové nástroje (SVN a správa požadavků v Redmine).

Ke kladům RRC patří (podobně, jako v případě RTC) skutečnost, že nutí uživatele dodržet tolik potřebný řád v projektech většího rozsahu. Modelovat v tomto skladateli požadavky pro menší projekty je značně nekomfortní. I pro namodelování jednoho požadavku bychom nutně tvořili mnoho typů potřebných artefaktů a nástroj se museli učit ovládat. Vzhledem k tomu, že projekt ST je rozsáhlý, ale v jednotlivých semestrech rozdělený na relativně malé části, je použití RRC komplikované, a stejně jako v případě RTC by se muselo v rámci katedry začít masivně využívat.

6 Závěr

V rámci diplomové práce jsme se seznámili s vývojem webové hry Space Traffic, použitými metodikami a nástroji pro vývoj a implementaci hry. Závěrem shrneme naplnění zadání práce a další možné přínosy pro vývoj Space Traffic.

Prvním úkolem autora bylo seznámení se se stavem vývoje webové hry ST a s použitými technologiemi. (Jak uvádíme mimo jiné v 3.kapitole, hra byla v prvním roce (2009-2010) vyvíjena v PHP bez využití jakýchkoliv frameworků. Po nasazení na server působila relativně dobrým dojmem. Problémy se ukázaly až ve chvíli, kdy byl sestaven tým v druhém roce vývoje (2010-2011) pod vedením R.K., který měl za úkol navázat na předchozí implementaci a pokračovat v rozběhnuté práci. Zjistilo se, že implementace nebyla postavena na funkční architektuře a že bez jejího přepracování nebylo možné pokračovat. Na aplikaci z prvního roku se dalo pohlížet jen jako na prototyp. Mimo prototypu jsme měli k dispozici jen propracovaný dokument „Space Trafic – Gamedesign" a analýzu několika z něj vycházejících požadavků.

Dalším úkolem bylo vybrat vhodnou metodiku vývoje a aplikovat ji. Autor stavěl na znalostech především metodiky UP/RUP mající propracovaný postup, dobře definované workflow (průběh) a specifikované jednotlivé milníky. Od počátku usiloval o její naplnění, a to ještě dříve, než se začal zabývat samotným kódem webové hry (předchozí úkol).

Podpůrná sada nástrojů měla být složena ze zástupců platformy IBM Jazz. Z té byl vybrán Rational Team Concert, nástroj pro vedení a řízení projektu sloužící především ke správě zdrojových kódů, vytváření plánů iterací s jednotlivými úkoly a jejich přidělování jednotlivým vývojářům. Dále Rational Requirements Composer sloužící ke specifikaci požadavků a jejich analýze. Díky těmto nástrojům bylo možné pokračovat v iterativním vývoji a implementaci ST. Na druhou stranu, ačkoliv se jedná o nástroje velmi propracované, s tím související složitost negativně působila především na méně zkušené členy týmu ST, což je od nástrojů odrazovalo a značně snižovalo jejich přínos pro projekt ST.

Tvrzení dokládáme na příkladu využití RTC, které vyžaduje být instalováno jako speciální plugin do Eclipse. Instalace pluginu v době realizace (2010-2011) nebyla zautomatizována, vyžadovala splnění celé řady závislostí na jiných pluginech a dodatečnou, ne zcela triviální konfiguraci. Ani práce s RTC nebyla intuitivní. Odevzdávání zdrojových kódů probíhalo na rozdíl od případu použití všeobecně známého nástroje SVN dvojkrokově (soubor si nejprve nahrajete do soukromého úložiště a pak teprve může být předán celému týmu), a to jen z prostředí Eclipse. Odevzdání jiných typů souborů (soubory jiného charakteru např. textové dokumenty, specifikace), které standartě nejsou součástí projektu Eclipse, bylo nutné nejprve do projektu importovat.

Sestavování týmu proběhlo v podstatě úspěšně na základě odhadů, jaké způsoby oslovení studentů (jako nových potenciálních členů týmu) jsou nejefektivnější. Patřily mezi ně především náborové prezentace v rámci vhodných předmětů KIV a informační plakáty. Podařilo se vytvořit tříčlenný tým (kromě autora jako projekt manažera) tvořený dvěma

studenty bakalářského studia a zkušeným vývojářem z navazujícího programu. Litujeme, že nábor započal z technických důvodů až v říjnu (bylo nutné nastudovat a připravit technické zázemí pro projekt, přístup na serveru s ST, návody na konfiguraci nástrojů IBM Jazz atd.). Za předpokladu časnějšího sestavování týmu by bylo možné, jak ukazuje vývoj v následujícím roce (2011-2012), zapojit více studentů oslovených skrze zápočtové práce.

Před začátkem druhého roku ST vznikla domněnka, že projekt se nachází na konci fáze rozpracování. Za cíl druhého roku považujeme převzetí a uvedení ST do rutinního provozu skrze fázi nasazení. ST se ovšem ve skutečnosti nacházel pouze ve fázi zahájení, což nejen, že původní rozsah harmonogramu projektu odhalilo jako nadsazený, ale jak se později bohužel ukázalo i jako v rámci jednoho akademického roku nerealizovatelný. V dlouhodobějším horizontu (v následném roce) se ukázalo, že nebylo možné hru dokončit ani v případě, že by byl projekt v původně myšleném stavu. Rozdíl v odhadu potřebné doby pro vývoj mimo jiné způsobila i absence zkušeností s vývojem větších aplikací v rámci čistě studentské komunity, kdy kromě vývoje probíhá i proces získávání znalostí a zkušeností, a to nejen na úrovni programátorů, ale i vedoucích týmů. Plánování studentského projektu je velmi časově náročné a vyžaduje od projekt manažera značný výdej energie při rozdělování dílčích úkolů jednotlivým členům týmu (často studentům prvních ročníků bez praktických zkušeností s vývojem) a neustálou pozornost, aby ST směřoval do cíle daného vizí pro akademický rok.

Jedním z výsledků je i zjištění, že z časového hlediska aplikace takového rozsahu v prostředí studentského projektu nelze vytvořit v krátkém čase 1-2 let. Když se na vývojový proces podíváme z pohledu UP, ukazuje se, že první rok představoval fázi zahájení, druhý přechod k rozpracování (bylo třeba opravit některé nedostatky a navrhnout celkovou architekturu systému, grafické uživatelské rozhraní) a třetí období konstrukce. (Došlo ke změně implementačního jazyka, vytvoření výsledné podoby architektury, převzetí návrhu a šablony grafického uživatelského rozhraní z druhého roku vývoje.)

Když se zamyslíme nad vlastnostmi webové hry s ohledem na původní záměr, musíme konstatovat, že v základních věcech tým postupuje dle původního dokumentu gamedesignu (návrhu hru) z prvního roku vývoje. V současné době některé návrhy neprošly (aliance, útoky pirátů na obchodní flotily), ale myšlenka obchodování ve vesmíru a vytváření scriptů pro automatické obchodování zůstává hlavním motivem hry a cílem ke kterému se spěje. I softwarový vývoj studentského projektu může být plynulý tak, že každá fáze staví na fázi předchozí. Dobré výsledky a zkušenosti přetrvávají a z věcí, které se ukázaly jako chybné, se v další fázi vývojový tým nových studentů skutečně poučí a osvědčené postupy vylepší.

Příkladem může být způsob hledání týmu. Víme, že v prvním roce se podařilo najít pouze jednoho programátora. Ve druhém způsob náboru doznal vylepšení na základě předchozích zkušeností, a tak zájemci přibyli, přičemž jeden z nich se podílel na vedení projektu i ve třetím roce, v němž projekt manažer řídil již několik týmů z řad studentů.

Jako druhý příklad můžeme uvést způsob plánování iterací. V prvním roce během prosince a ledna probíhala realizace projektu značně neefektivně (studenti se věnovali především odevzdávání semestrálních prací a dalším úkolům plynoucím ze zkouškového období), ve druhém se podařilo tento efekt více potlačit, ale přesto došlo ke stagnaci během únorové iterace (studenti museli řešit jiné odložené povinnosti) a ve třetím se iterace plánovaly

již od září, protože se počítalo s tím, že v období prosince a ledna nebude odvedena téměř žádná práce.

Na studentský projekt ST je třeba nahlížet jako na druh praktické výuky, zdroj nových zajímavých zkušenosti. Nikoli pouze jako na pracovní proces, jehož jediným výsledkem je hotová aplikace. Z tohoto důvodu nezbytně potřebujeme dbát o pečlivost a přehlednost vývoje a jeho vedení tak, aby ST mohl pokračovat i v budoucnu. Tomu může posloužit i tato práce, neboť chce plnit roli určitého průvodce pro studenty rozhodnuté se do projektu zapojit.

Zkratky

1.RST	– první rok vývoje ST, pod vedením Z.N. 2009-2010
2.RST	– druhý rok vývoje ST, pod vedením R.K. 2010-2011
3.RST	– třetí rok vývoje ST, pod vedením P.V. 2011-2012
dp.	– diplomová práce
ČVUT	– České vysoké učení technické.
FDD	– agilní metodika z angl. Feature Driven Development
KIV/ASWI	– předmět KIV Pokročilé softwarové inženýrství
KIV/DB1	– předmět KIV Databázové systémy 1
KIV/PIA	– předmět KIV Programování Internetových aplikací
KIV/PT	– předmět KIV Programovací techniky (základy obj. prog.)
KIV/WEB	– předmět KIV Webové aplikace (seznámení s CSS a PHP)
KIV/ZSWI	– předmět KIV Základy softwarového inženýrství
IBM	– přední světová společnost v informačních technologiích z angl. International Business Machines Corporation
KIV	– katedra informatiky a výpočetní techniky ZČU
LD	– agilní metodika vývoje softwaru z angl. Lean development
M.S.	– Bc. Martin Štěpánek, programátor, později architekt modulární a parametrizovatelné implementace jádra 2010-2012
P.B.	– Ing. MSc. Přemysl BRADA Ph.D. - garant projektu 2009-2012
P.V.	– Bc. Petr Vogl, vedoucí projektu 2011-2012
R.K.	– Bc. Richard Kocman, vedoucí projektu 2010-2011
RRC	– Nástroj platformy IBM Jazz Rational Requirements Composer
RTC	– Nástroj platformy IBM Jazz Rational Team Concert
RUP	– komerční implementace UP společnosti IBM z angl. Rational Uniefed Process
ST	– projekt webové hry Space Traffic
TDD	– agilní metodika vývoje softwaru z angl. Test Driven Development
XP	– agilní metodika vývoje softwaru- extrémní programování z angl. Extreme programming
UP	– metodika vývoje softwaru z angl. Uniefed Process a je i otevřeným standardem
Z.N.	– Ing. Zbyněk Neudert, vedoucí projektu 2009-2010
ZČU	– Západočeská univerzita v Plzni

Literatura

[AEJ] *jazz.net* **Articles about Jazz**
https://jazz.net/library/#type=article&q=
online březen 2012

[AGP/04] *Václav Kadlec* **Agilní programování - Metodiky efektivního vývoje softwaru**
Computer Press,
Brno 2004

[ANVWH/10] *Zbyněk Neudert* **Analýza, návrh a vedení týmu v projektu webové hry**
ZČU,
Plzeň 2010

[AOGD/08] *Jesse Schell* **The Art of Game Design**
Morgan Kaufmann Publishers,
USA 2008

[ASDP] *ebookbrowse.com* **Agile Software Development Poster**
http://ebookbrowse.com/agile-software-development-poster-en-pdf-d115015982
online březen 2012

[CRY/04] *Alistair Cockburn* **Crystal Clear: A Human-Powered Methodology for Small Teams**
Addison-Wesley Professional,
2004 ISBN 0201699478

[DRI/09] *Daniel H. Pink* **Drive: The Surprising Truth About What Motivates Us**
Riverhead Hardcover,
2009 ISBN 1594488843

[FDD] *www.featuredrivendevelopment.com* **Feature Driven Development**
http://www.featuredrivendevelopment.com
online březen 2012

[JAZZ/10] *Adrian Cho* **The Jazz Process: Collaboration, Innovation, and Agility**
Addison-Wesley Professional,
2010 ISBN 0321636457

[JPL] *jazz.net* **Jazz Platform**
https://jazz.net/about/about-jazz-platform.jsp
online březen 2012

[JTSM/09] *Alan Page, Keh Johnston, Bj Rollison* **Jak testuje software Microsoft**
Computer Press,
Brno 2009

[LSD] *www.poppendieck.com* **Lean Software Development**
http://www.poppendieck.com/index.htm
online březen 2012

[NWH/08] *Zbyněk Neudert* **Návrh webové hry**
ČVUT,
Praha 2008

[OSP/06] *Steve McConnell* **Odhadování softwarových projektů**
Computer Press,
Brno 2006

[RPIT/11] *Kathy Schwalbe* **Řízení projektů v IT**
Computer Press,
Brno 2011

[RRCI] *www.ibm.com* **Rational Requirements Composer**
http://www-01.ibm.com/software/awdtools/rrc
online březen 2012

[RRCJ] *jazz.net* **Rational Requirements Composer**
https://jazz.net/projects/rational-requirements-composer
online březen 2012

[RTCJ] *jazz.net* **Rational Team Concert**
https://jazz.net/projects/rational-team-concert
online březen 2012

[RTCI] *www.ibm.com* **Rational Team Concert**
http://www-01.ibm.com/software/rational/products/rtc
online březen 2012

[RTCB] *boulder.ibm.com* **Rational Team Concert**
http://pic.dhe.ibm.com/infocenter/clmhelp/v3r0/index.jsp
online březen 2012

[SCRUM] *www.scrumalliance.com* **Scrum Alliance**
http://www.scrumalliance.com/pages/what_is_scrum
online březen 2012

[UML2/07] *Jim Arlow, Ila Neustadt* **UML2 a unifikovaný proces vývoje aplikací**
Computer Press,
Brno 2007

[USDP/99] *Ivar Jacobson, Grady Booch, James Rumbaugh* **The Unified Software Development Process**
Addison-Wesley Professional,
1999 ISBN 0201571692

[XP] *xprogramming.com* **XProgramming.com**
http://xprogramming.com/what-is-extreme-programming
online březen 2012

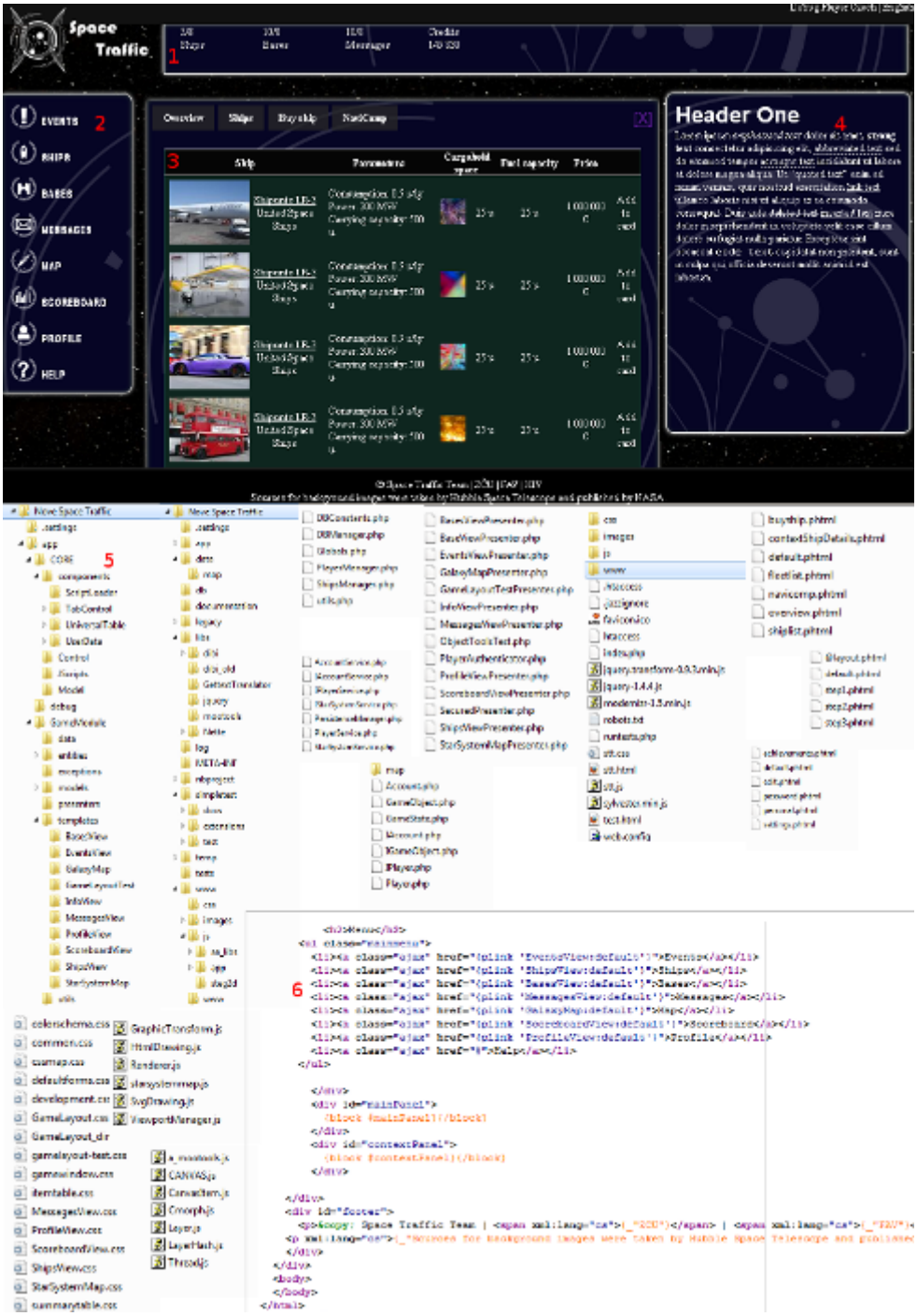
[ZKV/07] *Mike Gunderloy* **Z kodéra vývojářem**
Computer Press,
Brno 2007

Poznámka: Pokud u některých citací není údaj, nepodařilo se jej zjistit.

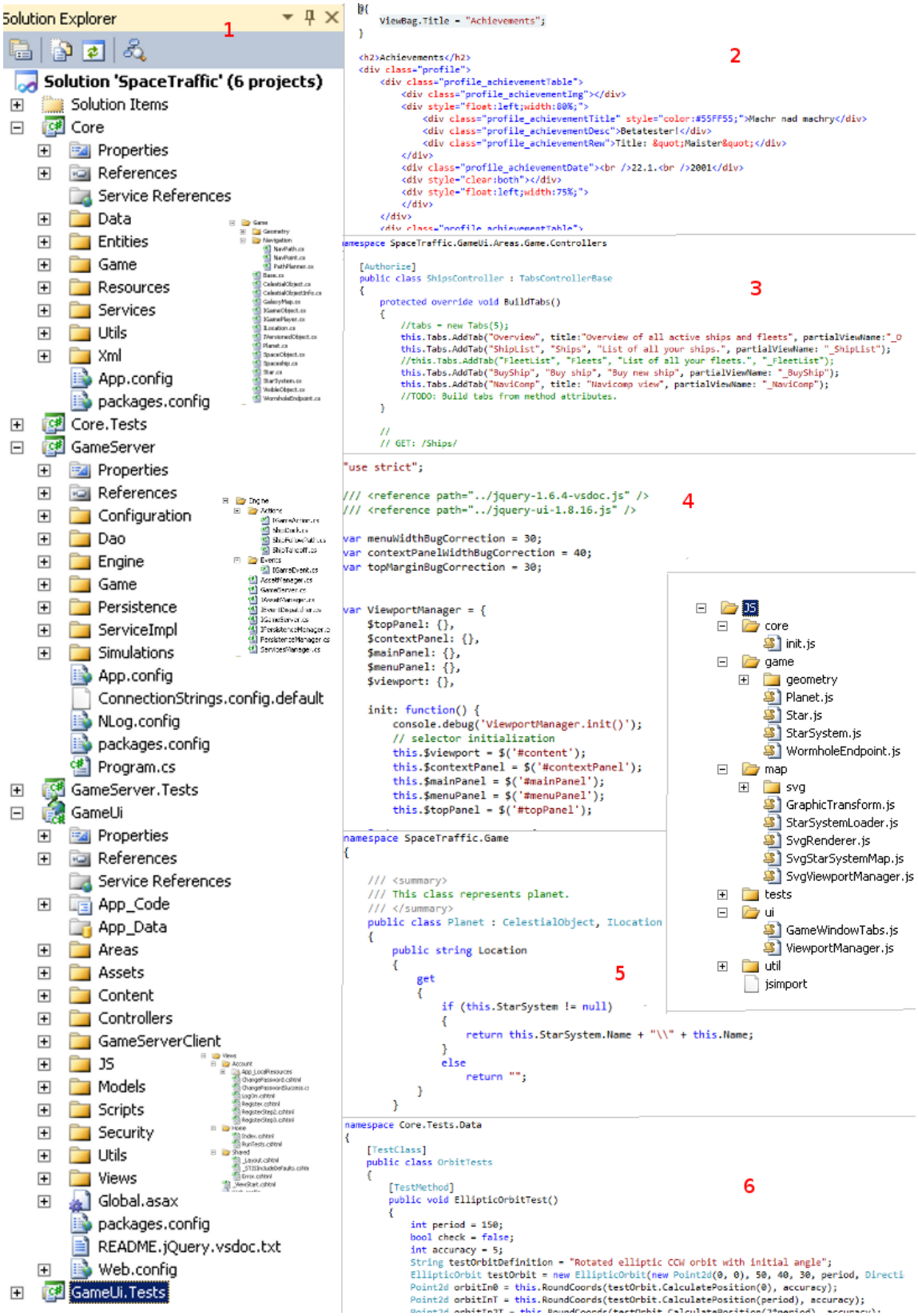
Přílohy

Na následujících stranách jsou vloženy celostránkové přílohy.

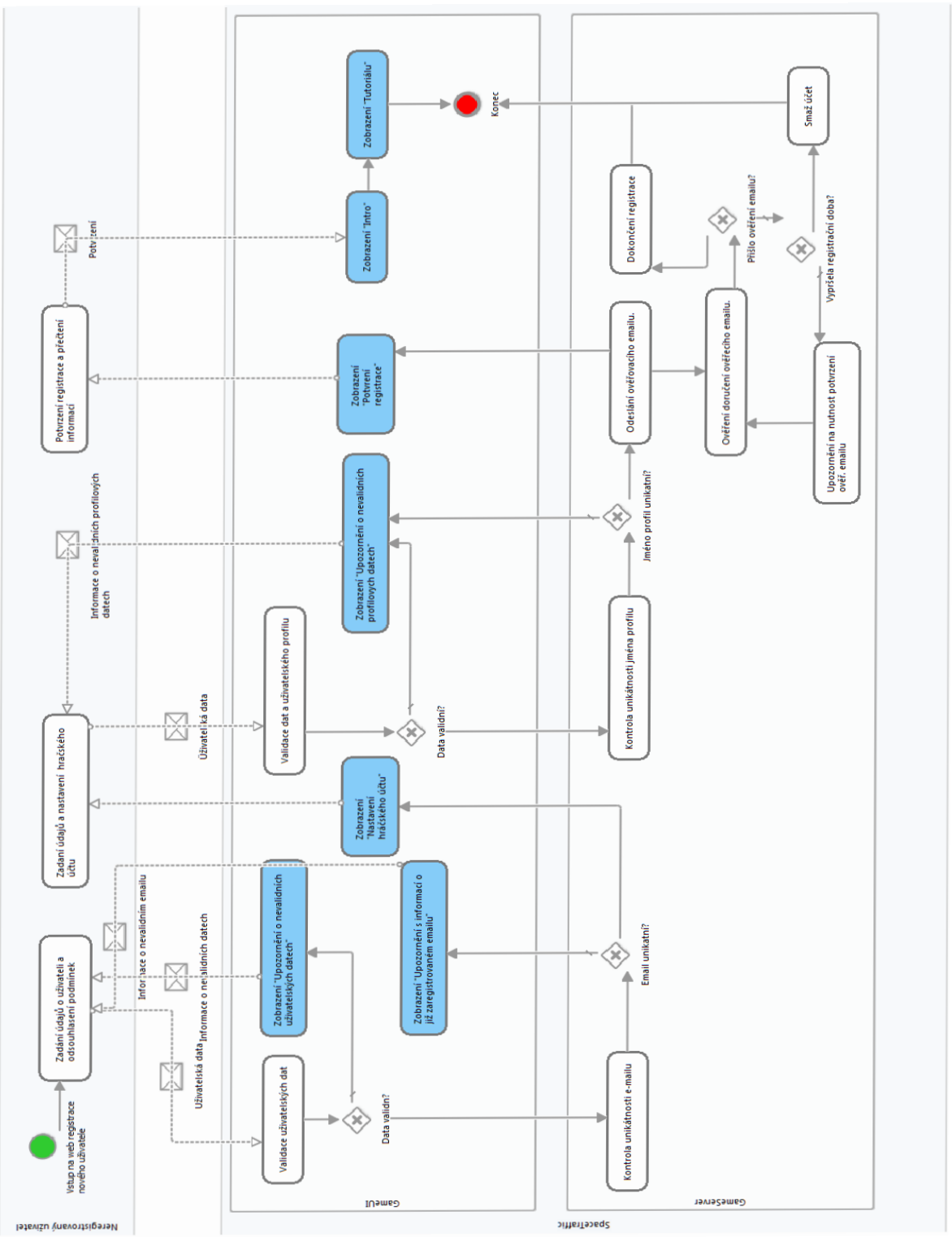
Obrázek 6.1: Náhled na první rok vývoje Space Traffic.



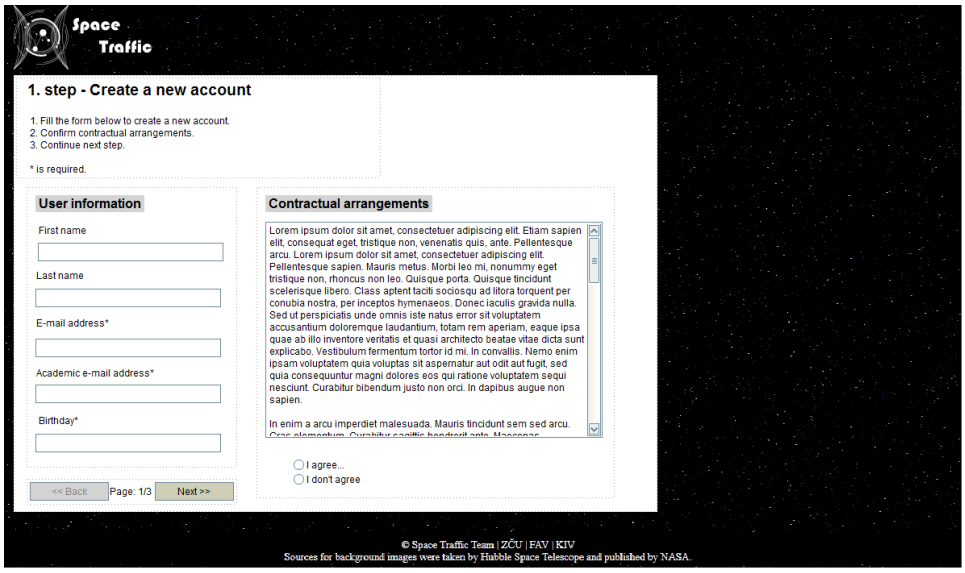
Obrázek 6.2: Náhled na druhý rok vývoje Space Traffic.



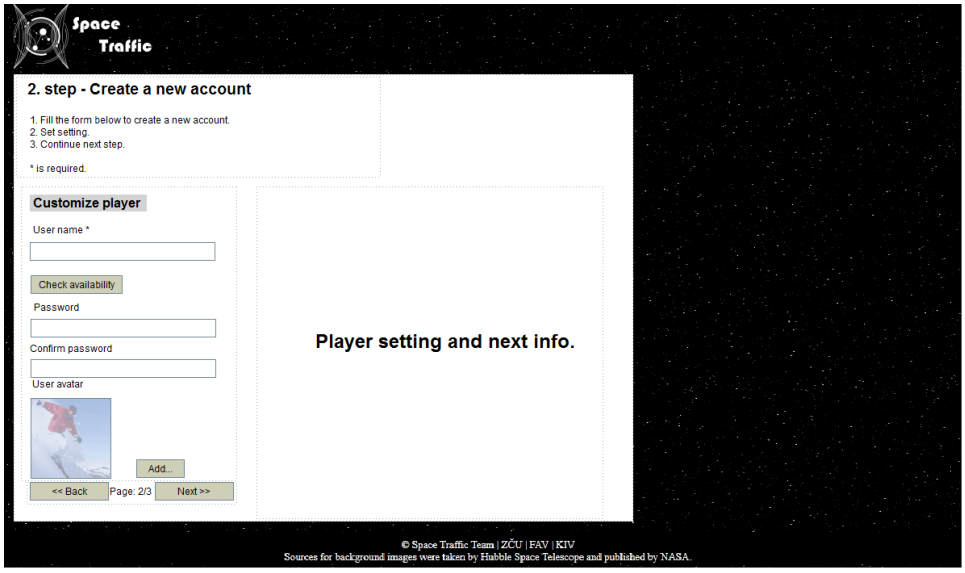
Obrázek 6.3: Náhled na třetí rok vývoje Space Traffic.



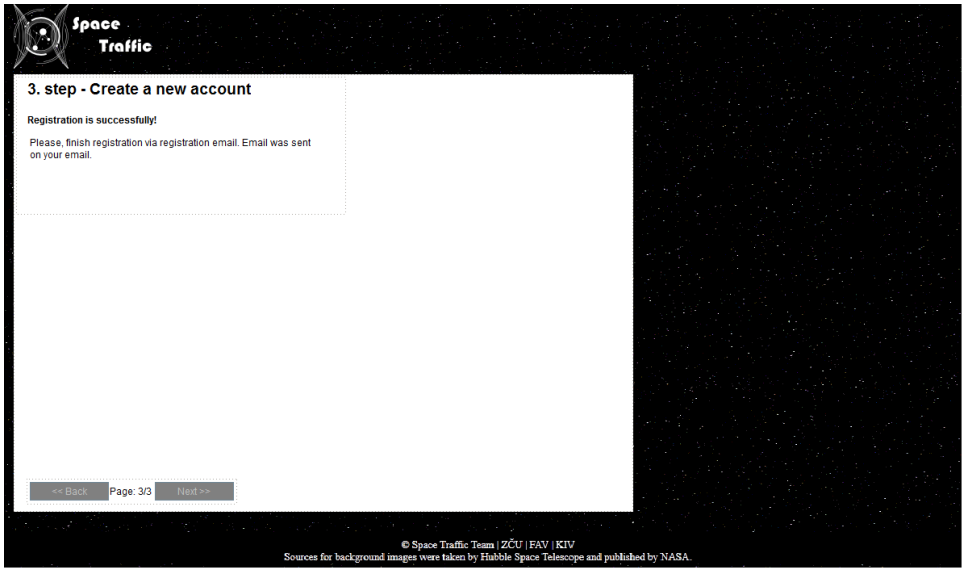
Obrázek 6.4: Ukázka analýzy registrace v nástroji RRC (Business Process Diagram).



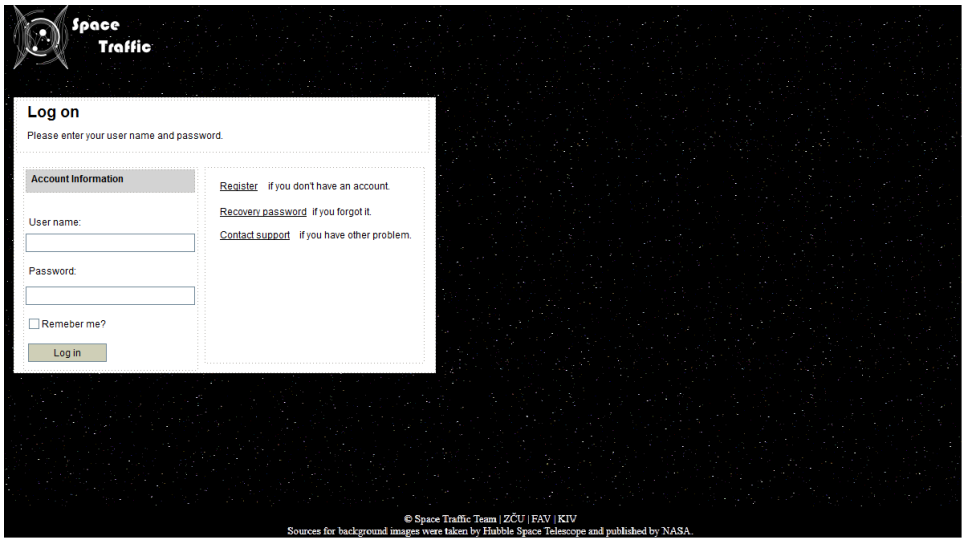
Obrázek 6.5: Ukázka analýzy registrace v nástroji RRC (UI sketch)- Krok první.



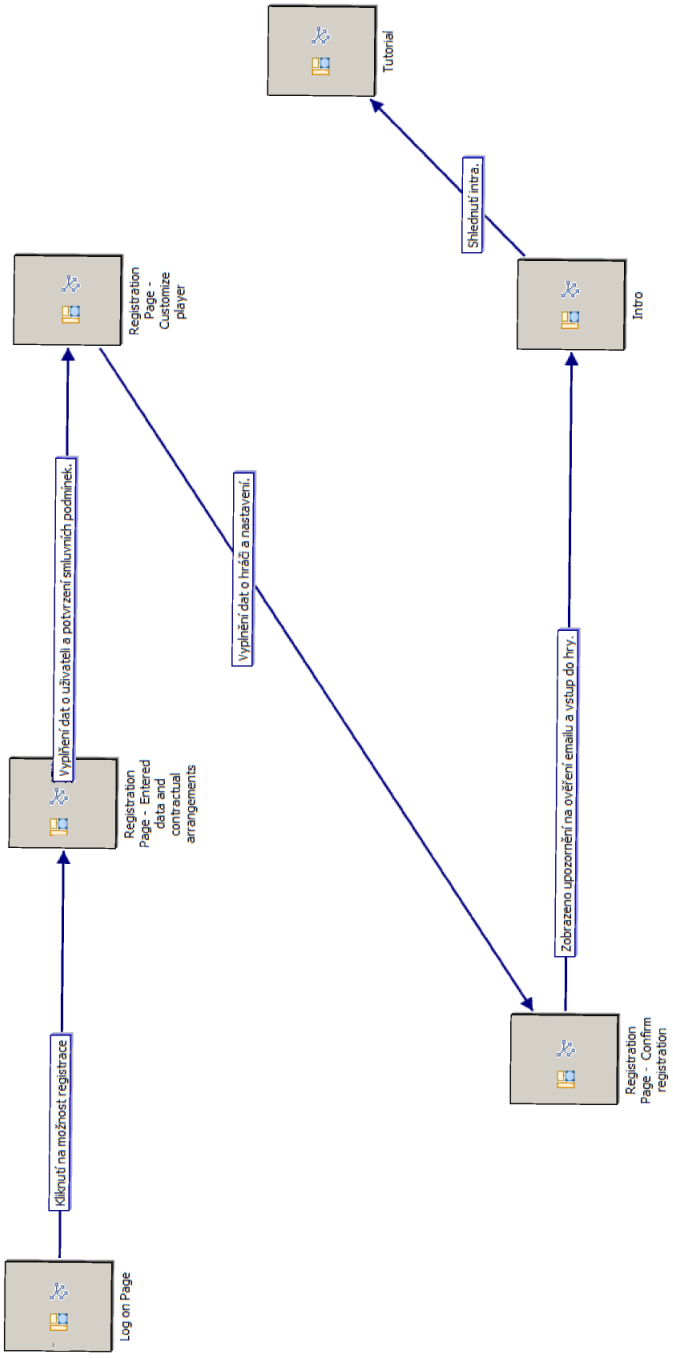
Obrázek 6.6: Ukázka analýzy registrace v nástroji RRC (UI sketch)- Krok druhý.



Obrázek 6.7: Ukázka analýzy registrace v nástroji RRC (UI sketch) - Krok třetí.



Obrázek 6.8: Ukázka analýzy registrace v nástroji RRC (UI sketch) - Přihlášení.



Obrázek 6.9: Ukázka analýzy registrace v nástroji RRC (Screen Flow).

Seznam obrázků