VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

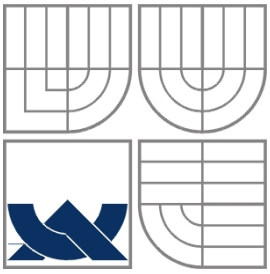
SYSTÉM PRO SPRÁVU IT PROJEKTŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

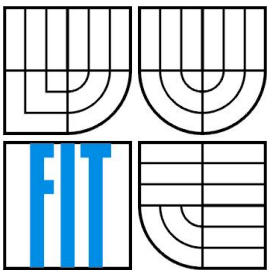
AUTOR PRÁCE
AUTHOR

BC. RICHARD DOBIÁŠ

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO SPRÁVU IT PROJEKTŮ

IT PROJECTS MANAGEMENT SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. RICHARD DOBIÁŠ

VEDOUCÍ PRÁCE
SUPERVISOR

ING. ŠÁRKA KVĚTOŇOVÁ

BRNO 2008

Abstrakt

Diplomová práce se zabývá tématem projektového řízení zaměřeného na vývoj softwarových projektů. Vysvětluje důležitý význam řízení projektů v IT oblasti a zmiňuje důležité standardy. Popisuje používané metodiky a modely životního cyklu vývoje softwarových projektů. Srovnává dostupné nástroje pro podporu projektového řízení. Analyzuje požadavky na systém pro řízení IT projektů a snaží se navržený systém v rozumném rozsahu implementovat s ohledem na budoucí využití v reálné praxi. Vzhledem k formě webové aplikace se fáze implementace zaměřuje na dodržení dnešních požadavků na kvalitní, přístupný a dobře použitelný web. Závěr zhodnocuje dosažené výsledky a navrhuje možná rozšíření realizovaného řešení.

Klíčová slova

Projektové řízení, plánování a řízení projektů, vývoj softwarových projektů, efektivní softwarové projekty, nástroje pro řízení projektů, webová aplikace, použitelnost, přístupnost.

Abstract

This diploma thesis focuses on software development management. It describes the importance of project management in IT fields and mentions its significant standards. Thesis devotes some chapters to software's project life cycle models. Another part of this thesis compares instruments supporting project management and analyzes system requirements of IT project management. A part of thesis implements the designed system concerning its real use in the future. Due to the web application form, the implementation phase focuses on keeping today's standards of accessible quality website, good usability. Conclusion of this thesis evaluates reached outcomes and mentions possible solution enhancement.

Keywords

Project management, project planning and managing, software project development, effective software projects, project management tools, web application, usability, accessibility.

Citace

Dobiáš Richard: Systém pro správu IT projektů. Brno, 2008, diplomová práce, FIT VUT v Brně.

Systém pro správu IT projektů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Šárky Květoňové. Další informace mi poskytl Ing. Stanislav Piskač, jednatel společnosti KomTeSa spol. s r.o., který mi byl v rámci této práce konzultantem.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Richard Dobiáš
31. 7. 2008

Poděkování

Děkuji Ing. Šárce Květoňové za vedení diplomové práce a její ochotu pomoci mi dotáhnout tuto práci do zdárného konce.
Děkuji Ing. Stanislavu Piskačovi za upřesnění zadání projektu, odborné konzultace, kompletaci požadavků na tento typ projektu, poskytnutí programové podpory a dobré rady podložené praxí.
Děkuji Jaroslavu Pešoutovi, DiS., za doporučení a poskytnutí velmi přínosné knihy o efektivních softwarových projektech [15], která rozšířila můj pohled na řešenou problematiku.

Obsah

1 Úvod

Pojmy jako projektový management, řízení či plánování začaly v posledních letech nabývat na svém významu. Oblasti managementu se věnuje čím dál více pozornosti ve sféře publicistiky i vzdělávání. Proč je tedy stále u vysokého procenta středních či velkých projektů pozorováno nesplnění plánovaných termínů, překročení rozpočtu, nedodržení předpokládaných kvalit či dokonce úplný krach projektu? Tyto otázky nutí projektový management ke studii možných příčin a k neustálým inovacím. Vznikají nové standardy týkající se řízení projektů a metodiky doporučující postup při řízení a vývoji aplikací [2,3].

Plánování a řízení se neomezuje jen na oblast informačních technologií (dále jen IT). Lidé se snaží již delší dobu optimalizovat různé lidské činnosti a pokud je to možné, rovněž je standardizovat. Těžko si dnes představíme profesionální týmový sport, ve kterém by neměl tým vlastního profesionálního trenéra. Ten své hráče vede podle zaručených příruček, postupů a svých letitých zkušeností s tréninkem týmu. Pokud má týmová práce dobře fungovat, musí někdo všechny členy týmu vhodným způsobem řídit a motivovat.

Ve světě IT toto platí dvojnásob. Vše se točí kolem projektů, jejich termínů, rozpočtu, rozsahu funkčnosti, kvality atd. A právě dnešní doba, typická svým vývojem nových informačních technologií, standardizací, vznikem dalších metodik a zvyšováním konkurenceschopnosti IT firem, potřebuje skutečně kvalitní projektové řízení. Stejně jako hokejový tým potřebuje dobrého trenéra vedoucího svůj tým k vyhranému zápasu, potřebuje tým vývojářů svého IT manažera, který bude řídit projekt až do jeho úspěšného dokončení a splnění požadavků zadavatele.

1.1 Vymezení cíle diplomového projektu

Cílem mé diplomové práce je seznámit se důkladně s problematikou projektového řízení zaměřeného na IT oblast. Na základě potřeb projektového řízení definuji požadavky na programovou podporu, která by ulehčovala správu a řízení IT projektů. Z těchto požadavků vychází návrh systému, který v průběhu práce implementuji. V závěrečné části zhodnotím celý projekt a uvedu možná rozšíření.

Celou práci jsem konzultoval s firmou KomTeSa spol. s r.o. (dále jen KomTeSa), konkrétně s Ing. Stanislavem Piskačem, který mé analýzy a návrhy usměrňoval a doporučoval optimální cesty, kam by se vývoj projektu měl ubírat. Výsledkem konzultací byl záměr pro vytvoření užitečné webové aplikace, která bude nasazena v reálné praxi a bude propojena s firemním informačním systémem. Tato skutečnost již sama o sobě vkládá do projektu jistá omezení, doporučení a nutné cesty vývoje. Vzhledem ke složitosti a rozsahu projektu předpokládám, že navrhnu a zrealizuji základ, na kterém bude moci dále pracovat celý vývojový tým v KomTeSe. Více informací o této problematice uvádím ve třetí kapitole, která se věnuje fázi analýzy projektu.

Ve druhé kapitole této práce se snažím shrnout přehledným způsobem obrovský rozsah problematiky projektového plánování, řízení a vývoje softwarových projektů. Důležitou částí je podkapitola věnovaná významu projektového řízení. I přes spoustu statistických údajů se jedná o přesvědčivé informace, které by měly jasně dokázat nepostradatelnost této problematiky v IT. Celou jednu podkapitolu věnuji zajímavé problematice efektivních softwarových projektů.

Ve třetí kapitole se věnuji analýze požadavků na programovou podporu řízení IT projektů. Fáze analýzy je bezpochyby nejdůležitější etapou při jakémkoliv vývoji. V úvodu kapitoly stručně popisuji běžný postup analytických prací při vývoji softwarových projektů. Následně uvádím důležité body provedené analýzy a výsledky dokumentuji specifikací požadavků a vhodnými diagramy.

Z analýzy vychází čtvrtá kapitola věnovaná fázi návrhu. V této části se věnuji návrhu celého systému. Návrh odpovídá specifikaci požadavků, které byly definovány během analýzy. Konkrétněji se zaměřuji na návrh datového modelu a návrhu uživatelského rozhraní.

Navazuje obsáhlejší etapa zahrnující implementaci navrženého systému. Popisuji všechny použité technologie a jejich vhodnost při řešení tohoto projektu. Zmiňuji zajímavé techniky a způsoby řešení, které jsem použil během realizace systému. V jedné podkapitole vysvětluji, proč jsem kladl velký důraz na dobrou použitelnost a přístupnost webové aplikace. Na konci kapitoly shrnuji neimplementované části, důvody pro odložení realizace některých částí a současně také navrhuji, jakým způsobem by bylo dobré tyto části dále implementovat a rozvíjet.

Šestá kapitola se zabývá fází testování. Stručně popisuji obecnou charakteristiku a význam této etapy vývoje projektu. Uvádím příklady a způsoby, kterými jsem testoval správnou funkčnost a rozsah dostupných funkcí realizované aplikace.

V sedmé kapitole popisuji fázi nasazení realizovaného systému. Uvádím postup instalace a minimální požadavky potřebné pro bezproblémový chod systému. Rovněž uvádím odkaz na web, kde je zprovozněna funkční demoverze aplikace.

Závěrečná kapitola shrnuje celý projekt a zhodnocuje dosažené výsledky. Uvádím problémy, se kterými jsem se setkal během realizace diplomové práce. Vysvětluji přínosy, které jsem do projektu vložil a doporučuji velké množství možných rozšíření projektu do budoucna.

2 Projektové řízení

Oblast projektového řízení je velmi obsáhlá a není snadné do ni proniknout. Dobrý projektový manažer musí umět správně využít své znalosti a praktické zkušenosti s vedením projektů. Pomoci mu v tom mohou i různé podpůrné nástroje, které však samy o sobě nikdy nemohou zaručit úspěch projektu. Pro větší a náročnější projekty je samozřejmě vhodná programová podpora doslova nutností. Klíčovou roli v úspěchu projektů, a samozřejmě i celé organizace, hraje osoba projektového manažera. Kromě teoretických a praktických zkušeností s plánováním a řízením projektu potřebuje také umět správně jednat s lidmi a mít zkušenosti s jejich vedením. Řízení projektů je totiž z největší části o vedení týmu. V této kapitole se pokusím shrnout alespoň stručně ucelený přehled o celé problematice projektového řízení se zaměřením na vývoj softwarových projektů.

2.1 Seznámení s projektovým řízením

Literatura týkající se řízení projektů bývá plná nových pojmů a jejich definic, které si projektový manažer musí osvojit. V následujícím textu uvedu společně s úvodem do celé problematiky alespoň nejčastěji používané pojmy, které jsou důležité z hlediska pochopení základních myšlenek této diplomové práce.

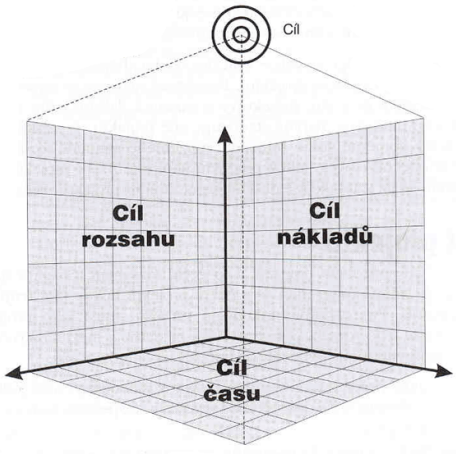
Pro pojem *projekt* lze v literatuře nalézt několik definic [1]. Projekt je jedinečný proces sestávající z řady koordinovaných, řízených činností, s datem zahájení a ukončení, prováděný pro dosažení určitého cíle, který vyhovuje specifickým požadavkům, včetně omezení daných časem, náklady a zdroji. Pro projekt je důležité, že se jedná o unikátní dílo. Vývoj produktu se skládá z několika fází, které se musí koordinovat. Některé úkoly bývají závislé na dokončení jiných úkolů. U projektu se požaduje měření kvality pomocí různých metrik a samozřejmě existuje rozpočet, který zpravidla bývá těsný. Projekty se od běžných provozních činností liší skutečností, že projekt je dočasný, tzn. po splnění stanovených cílů skončí. Zjednodušeně bychom projekt mohli označit jako „časově omezenou pracovní činnost, jejímž cílem je vytvoření jedinečného produktu, služby, nebo dosažení jiného výsledku“ [6].

Pro doplnění je vhodné ještě upřesnit význam zmíněného pojmu *proces*. Podle [1] je proces po částech uspořádaná množina činností, které tvoří opakovatelným způsobem požadované výstupy na základě jednoho či více vstupů. Proces můžeme tedy chápat jako určitý tok práce postupující strukturou organizace od jednoho účastníka k druhému. Hlavním cílem je uspokojit potřeby zákazníka procesu. Stručně řečeno, proces se skládá z uspořádaných činností, má jednoznačně vymezený začátek a konec, transformuje definované vstupy na výstupy, využívá zdroje, je opakovatelný a může být popsán na různých úrovních abstrakce. Za zmínku ještě stojí pojem *podnikový proces*. Ten představuje množinu jedné či více činností, které dohromady realizují

podnikový cíl. Tento cíl má nějakou hodnotu pro zákazníka. Činnosti obvykle definují funkční role a vztahy v rámci organizační struktury.

Projekty mohou být velké i malé. Mohou se týkat tisíců osob, malého týmu nebo někdy i jedné osoby. Spoustu úkolů či celé zakázky, se kterými se dnes setkáváme, lze řešit formou projektu. Ve světě IT bývá nejčastějším úkolem vytvoření produktu či služby. Jako příklady IT projektů lze uvést např. rozšíření technické infrastruktury fakulty o bezdrátový přístup k Internetu, vybavení celé organizace novým hardwarem, vývoj nového systému pro zvýšení produktivity práce zaměstnanců v prodeji, vývoj webových stránek umožňující jednodušší prodej výrobků přes Internet atd.

Podle [5] existují čtyři typické charakteristiky projektů, které odlišují projektové řízení od dalších manažerských činností. *Projekty jsou jedinečné*. I když mohou být dva projekty velmi podobné, vždy budou existovat jisté odlišnosti, které z každého projektu činí unikátní úkol pro dodavatele cílového produktu. *Projekty využívají zdroje*. Ty jsou materiální a také lidské. Milton D. Rosenau, autor velmi úspěšné knihy o řízení projektů [5], výstižně uvádí: "Řídit projekty znamená řídit lidi". Právě vedení lidí bývá tou nejtěžší částí projektového řízení a zvládnout jej představuje velkou výzvu pro každého projektového manažera. *Projekty se realizují v rámci organizace*. A co jiného tvoří organizaci než lidé. Projekt pak ovlivňuje míra zvládání mezilidských vztahů, které jsou neodmyslitelnou součástí složitých situací uvnitř organizace. Zbývá poslední charakteristika projektů, která se zdá být jednou z nejdůležitějších. *Projekty mají cíl*. Tento cíl se označuje jako trojí omezení, neboli *trojimperativ*, jelikož se skládá ze tří podmínek, které je třeba splnit – specifikace provedení, časový plán a rozpočet. Trojí omezení cíle projektu tedy popisuje vzájemné vazby základních prvků každého projektu. Přehledně je tato skutečnost znázorněna na obrázku 1.



Obr. 1: Trojimperativ [6]

Trojimperativ projektu odpovídá záměrně definici, kterou je v obchodním zákoníku vymezena smlouva o dílo. Každá smlouva totiž musí obsahovat specifikaci plnění, termíny a cenu. Bez toho ji nelze považovat za platnou smlouvu [10]. Každý projekt má unikátní rozložení trojimperativu. První zadavatel může preferovat co nejkratší termín na úkor navýšení rozpočtu, druhý naopak může

požadovat co nejnižší náklady na úkor kvality. V projektu však mohou hrát důležitou roli i další prvky, z nichž nejčastějším bývá kvalita projektu či uspokojení požadavků zadavatele. Z tohoto důvodu se v literatuře můžeme setkat také s označením *čtvero omezení*, které k trojimperativu přidává ještě prvek kvality [6].

Aby se projekt dal označit jako úspěšný, je potřeba splnit jeho cíl. Jinými slovy, výsledkem úspěšného projektového řízení je projekt splňující specifikaci provedení v požadovaném rozsahu, se všemi požadovanými výstupy, v dohodnutém termínu a v rámci rozpočtu. Je nutné dosáhnout všech těchto nezávislých podmínek současně. V případě, že nedojde ke splnění třeba i jedné části, nelze již hovořit o úspěšném projektu. Dosáhnout uspokojivé kvality požadované zadavatelem může jen dobré projektové řízení, které však znamená mnohem více než jen splnění zmíněného trojího omezení.

Projektové řízení je podle [1] uplatnění veškerých poznatků, znalostí, dovedností, nástrojů a technik v projektových činnostech s cílem splnit nebo překročit potřeby zájmových skupin a jejich očekávání od projektu. Pro splnění či překročení těchto potřeb je nutné neustále proti sobě vyrovnávat požadavky mezi:

- Rozsahem prací, časem, náklady a jakostí
- Zájmovými skupinami s různými potřebami a očekáváními
- Stanovenými a nestanovenými požadavky (potřebami)

Projektové řízení může být v různé literatuře označováno dalšími synonymy. Někdy se také používají anglické termíny. Často lze spatřit označení projektový management, management projektů, project management, řízení projektů apod. Opět uvedu také zjednodušenou formulaci, která řízení projektu definuje jako „uplatnění veškerých poznatků, dovedností, nástrojů a technik na aktivity (činnosti) projektu takovým způsobem, aby byly splněny požadavky na projekt“ [6].

Řízení projektů z oboru IT má některá svá specifika. Manažeři projektů v této oblasti musí ovládat určité znalosti navíc. Obtížným úkolem může být pro manažera sestavení a vedení projektového týmu vzhledem k vysoké pracovní specializaci lidí v tomto oboru. Typické je také rychle měnící se prostředí, které přináší stále nové technologie a změny. Řízení tedy musí držet krok s dynamickým prostředím a využívat rychlé procesy, které budou produkovat odpovídající produkty. Co se týče povahy IT projektů, na rozdíl od dalších oborů se jedná o velmi různorodé projekty zahrnující všechny možné obory lidské činnosti a různé obchodní funkce firem. Nejrozmanitější povahu mají softwarové projekty [6].

Softwarovým projektem rozumíme podle [10] plánovanou činnost směřující k dodání konkrétního softwarového produktu nebo služby za určitou dobu. Vznik softwarového projektu bývá vždy podmíněn potřebami zadavatele, který se chce vypořádat s určitým problémem v některém z jeho obchodních procesů nebo plánuje procesy zlepšit, aby tak zvýšil svou konkurenceschopnost. Cílem zadavatele je tedy dosažení určitého podnikatelského cíle.

Důležité je upřesnění významu a rozdílu často užívaných pojmů metodologie a metodika. *Metodologií* se rozumí vědní disciplína studující tvorbu a aplikaci metodik, způsoby řešení problému

a hledání odpovědí. Při vývoji software bývá název metodologie často nesprávně používán pro metodiku. Příčinou záměny je velmi častý nesprávný překlad pojmu z angličtiny. *Metodika* je obecně pracovní postup nebo nauka o metodě. Ve vývoji software představuje metodika souhrn doporučených praktik a postupů, které pokrývají celý životní cyklus vytvářené aplikace [9].

2.2 Význam projektového řízení

V dnešní době rychle roste počet řešených projektů i jejich složitost. Z tohoto důvodu si začíná stále více organizací i jednotlivců všímat oblasti řízení projektů. Dodnes skončí pouze třetina projektů úspěšným splněním požadovaného rozsahu, časového plánu i rozpočtu [6]. Podceňovat význam projektového řízení se nevyplácí. Řízení projektů pomáhá jak větší úspěšnosti jednotlivých projektů, tak i celé organizaci. Naštěstí se profese projektového řízení stále rozrůstá a její přítomnost je dnes zaznamenána prakticky ve všech možných oborech po celém světě.

Podle [6] se do osmdesátých let 20. století projektové řízení omezovalo na pouhé výkazy časového plánu a využití zdrojů. Oblastmi, kde se takto minimalistické řízení projektů využívalo, bylo stavebnictví a vojenské obory. Řízení projektů vzniklo původně v americké armádě. Tam byly také vyvinuty různé nástroje jako např. Ganttovy či síťové diagramy. V dnešní době, kdy se většina organizací setkává každý den s řešením projektů a využívá stále nové technologie, by asi žádná větší firma na trhu neobstála bez kvalitního projektového řízení.

Nárůst potřeby a významu řízení projektů dokládají i dostupné statistické údaje. Podle zprávy z roku 2001 se v USA ročně vydalo 2,3 bilionu dolarů na řešení projektů. To je pro představu přibližně čtvrtina hrubého domácího produktu země. V celém světě se ročně vydalo na projekty téměř 10 bilionů dolarů z hrubého produktu, který činí 40,7 bilionu dolarů. Dalším zajímavým číslem je více než 16 milionů lidí, kteří se v rámci své profese věnují řízení projektů. Na úspěchu podnikání se velmi výrazně podílejí projektoví manažeři. To si naštěstí dnešní společnost dobře uvědomuje [6].

Pokud by neexistovalo projektové řízení a projekt by nebyl žádným způsobem plánován a řízen, hrozí velké riziko, že se nedodrží termín předání, překročí se rozpočet, výsledný produkt nebude splňovat očekávání, kvalita neuspokojí požadavky zadavatele apod. Velkým přínosem projektového řízení je skutečnost, že díky němu lze včas zjistit možná ohrožení termínu, rozpočtu, kvality i rozsahu výsledného produktu. Řízení musí zahrnovat nepřetržité plánování, organizování, sledování a řízení všech hledisek projektu za účelem úspěšného dosažení cílů projektu [1].

Jak by na tom asi byla IT oblast bez projektového řízení? Jako důkaz mnoha problémů, které přináší nekvalitní projektové řízení či dokonce jeho absence, by mohla dobře posloužit často citovaná studie s názvem „CHAOS“. Touto studií zveřejnila v roce 1995 prestižní konzultační společnost Standish Group překvapivé informace o více než osmi tisících projektů z oblasti IT na území USA. Již název samotné studie napovídá chaotický stav sledovaných projektů. Ročně bylo vynaloženo více než 250 miliard dolarů na přibližně 175 000 projektů zaměřených na IT oblast. Úspěšnost projektů

však činila pouhých 16,2 %. Více než 31 % projektů bylo před jejich dokončením zastaveno. To znamenalo pro organizace a úřady ztrátu přes 81 miliard dolarů. Pokud by nic jiného nedokazovalo potřebu kvalitního projektového řízení, tak právě tato čísla jsou více než dostatečným důkazem. Pro lepší ilustraci cituji velmi výstižné prohlášení autorů studie: „Projekty zaměřené na vývoj softwaru jsou v chaosu a my se zkrátka již nemůžeme tvářit jako tři opice – neslyšet chyby, nevidět chyby a nemluvit o chybách“ [6].

Další následné studie od stejné společnosti dokazují postupné zlepšování statistických ukazatelů. Od roku 1994 do roku 2002 se podíl úspěšných IT projektů zvýšil z 16 % na 34 %. Úspěšnost se tedy zdvojnásobila, zatímco podíl neúspěšných projektů ve stejném období klesl na poloviční hodnotu a to z 31 % na 15 %. Co se týče nárůstu finančních nákladů v USA na IT projekty, ze zmiňovaných 250 miliard dolarů v roce 1994 se objem financí do roku 2002 zvýšil jen nepatrně na 255 miliard dolarů. Mnohem důležitější je skutečnost, že za tuto dobu se snížil objem financí vynaložených na neúspěšné či problémové projekty z hodnoty 140 miliard dolarů na 55 miliard dolarů [6]. Přestože podle [10] neúspěšnost softwarových projektů stále podstatně převyšuje neúspěšnost projektů tradičních inženýrských disciplín, jako je stavebnictví či elektrotechnika, navozují citované studie společnosti Standish Group optimistický pohled na budoucí vývoj softwarových projektů a efektivnější využití projektového řízení v IT oblasti.

Jaké shrnutí z předchozích údajů vyplývá? Svět IT zkrátka nemůže a zřejmě ani nechce podceňovat význam projektového řízení. V dnešní době je oblast řízení projektů v popředí zájmu většiny organizací. Stále však pokulhává nemalé množství organizací v praktickém dodržování potřebných metodik. Mnoho organizací potvrzuje, že jim projektové řízení přináší spoustu výhod. Zejména se jedná o lepší kontrolu nad zdroji, zkrácení doby vývoje, snížení nákladů, lepší vztahy s klienty, zlepšení produktivity práce, vyšší zisky apod.

2.3 Standardy, normy a certifikace

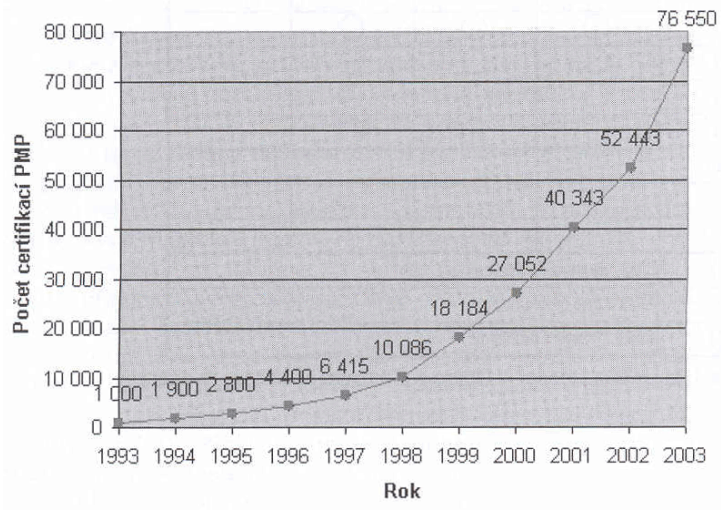
Projektový manažer by měl mít dobrý přehled o existujících normách, metodikách a omezeních, které se vztahují k jeho projektům. Jedině tak má šanci podle potřeb konkrétního projektu vybrat vhodnou metodiku a využít ji efektivně ve prospěch celého projektu. Plnění některých norem může dokonce vyžadovat přímo zadavatel projektu či legislativa.

Pro mnoho firem je velkým přínosem získání různých profesionálních certifikací, které jsou dnes důležitým faktorem zajišťování a uznávání kvality. Společnost, která vlastní renomovaný certifikát, je v očích svých potencionálních obchodních partnerů určitě významnější a důvěryhodnější. Při některých výběrových řízeních, zejména pokud jde o zahraniční zakázky, může zadavatel přímo vyžadovat, aby dodavatel vlastnil příslušné certifikace.

Pro zajištění kvalitního rozvoje projektového řízení, vývoj norem, jejich standardizaci a certifikaci vznikly mezinárodní instituce sdružující profesionály tohoto oboru. Za nejvýznamnější

instituce se považují Project Management Institut (PMI) v USA a International Project Management Association (IPMA) v Evropě. V naší republice byla zřízena Společnost pro projektové řízení (SŘP), která je národní organizací IPMA [8].

Čím je pro vývojáře webových aplikací konsorcium W3C, tím je pro projektové manažery institut *PMI*. Počátky této instituce sahají až do roku 1969. Nyní je PMI se svou členskou základnou o 240 000 členech ve více než 160 zemích vedoucí asociací pro profesi projektového řízení. Institut PMI nabízí certifikace Project Management Professional (PMP). Vzhledem k počtu držitelů této certifikace, která dosahuje hodnoty téměř 242 000, je PMP nejuznávanější certifikací v profesi projektového managementu [7]. Graf na obrázku 2 dokládá rychlý nárůst počtu získaných PMP certifikací. Ke získání certifikátu je potřeba doložit dostatečné zkušenosti s řízením projektů, souhlasit s etickým kodexem PMI a složit obsáhlou zkoušku. Některé společnosti platnou certifikaci PMP u všech svých projektových manažerů striktně vyžadují. Zvyšování složitosti IT projektů a jejich zasazení do globálního prostředí zvyšuje poptávku po zaměstnancích s prokazatelnými znalostmi a dovednostmi pro řízení projektů. Schopnost získat a udržet si určitou certifikaci je dnes velmi ceněná na trhu práce a při hledání práce je certifikát velkou výhodou. Několik studií již prokázalo, že organizace, které podporují programy odborných certifikací pro své zaměstnance, jsou ve svých činnostech mnohem efektivnější než firmy, které tyto programy certifikací nepodporují [6].



Obr. 2: Nárůst počtu certifikací PMP v letech 1993 – 2003 [6]

Institut PMI také vytvořil studijní materiál nazvaný PMBOK Guide 2004 (Guide to the Project Management Body of Knowledge). Jedná se o základní rámec a výchozí bod pro studium řízení projektů. Není to striktní metodika či norma, ale spíše takový průvodce projektovým řízením. Rozhodně ale projektovým manažerům pomáhá při jejich nesnadné práci. PMBOK dosáhl uznání jako mezinárodní standard. O tento materiál se opírá také další literatura jako např. [6].

V následující části alespoň stručně uvedu některé základní normy a standardy vztahující se k projektovému a procesnímu řízení, které zmiňuje [1].

Systémy řízení jakosti jsou dnes cílem mnoha organizací, které se snaží získat různé certifikáty jakosti. Mezi nejvýznamnější patří *normy ISO*, ze kterých vycházejí i některé naše ČSN normy. Získáním certifikátu organizace dokazuje svým potencionálním zákazníkům, že své produkty vytváří s definovanou kvalitou a za definovaných podmínek. Norma pro řízení kvality (jakosti) se označuje *ISO 9000* a předepisuje minimální požadavky, které musí organizace splnit pro dosažení certifikace. Norma popisuje nepřetržitý cyklus plánování, kontroly a dokumentování kvality v organizaci. Podle ohlasů organizací přináší zavedení ISO 9000 významné zlepšení kvality, spokojenější zákazníky a úsporu nákladů [6]. Normou pro hodnocení softwarových procesů je *ISO 15504*. Slouží k určení vyspělosti vývoje software v organizaci pomocí spolehlivých metod, které hodnotí stavy procesů a na základě zjištěných výsledků tyto procesy zlepšují.

Six Sigma je měřítkem pro téměř dokonalou kvalitu. Slouží ke snižování defektů v jakémkoliv vývojovém či výrobním procesu. Pravidla Six Sigma jsou velmi přísná. Požadují velmi jasné definice projektu a k jeho vlastnostem se staví velmi přísně. Možná díky tomu pojem Six Sigma dnes reprezentuje vysokou úroveň kvality.

Úroveň procesního řízení organizace (*Capability Maturity Model* – CMM) lze použít k určení stupně vývoje procesně zaměřených organizací. Slouží pro stanovení strategie zdokonalování řízení projektových procesů. K tomu je potřeba stanovit stávající úroveň vyspělosti organizace a definovat kritické oblasti. CMM má pět úrovní a každý přechod na vyšší úroveň znamená, že organizace zdokonalila řízení procesů.

Nástupcem CMM se v roce 2002 stal model CMMI (*Capability Maturity Model Integration*), který vznikl sloučením mnoha různých modelů a nesourodých prvků různých oborů do jednoho celku. Představuje způsob ohodnocení a popisu procesu vývoje v organizaci i způsob porovnávání s průmyslovými normami. Samozřejmě pomáhá dále tento proces rozvíjet a zlepšovat. Iniciativa CMMI existuje ve dvou podobách – postupná a kontinuální. Postupná CMMI je známější a stejně jako CMM má pět úrovní vyspělosti. Umožňuje srovnávat úroveň organizací a je vhodná pro inovace. Kontinuální CMMI umožňuje organizaci vybrat jen určitá zlepšení, která by nejlépe vyhovovala jejím podnikatelským cílům. Minimalizuje případná rizika, dovoluje lépe porovnávat procesy mezi jednotlivými projekty a rovněž usnadňuje přechod od jiných norem kvality. Model CMMI dokáže pro každou úroveň způsobilosti posoudit vědeckou metodou, do jaké míry organizace dosahuje daných cílů [13, 14].

2.4 Vývoj a řízení softwarových projektů

Úspěch softwarových projektů je podmíněn pěti vzájemně souvisejícími aspekty, které někdy bývají označovány jako pětiúhelník softwarového inženýrství [10]:

- Životní cyklus vývoje
- Modelovací jazyk

- Podpůrné nástroje
- Plánování projektu
- Řízení projektu

2.4.1 Životní cykly vývoje

Podle [10] je životní cyklus softwaru reprezentací softwarového procesu. Životním cyklem vývoje rozumíme změny, kterými softwarový produkt postupně prochází během své existence. Životnost takového produktu je dána od vzniku požadavku na jeho realizaci až po jeho vyřazení z provozu. Životní cyklus definuje fáze, jednotlivé kroky, aktivity, metody, nástroje a očekávané výstupy softwarového projektu.

2.4.1.1 Fáze životního cyklu vývoje

Analýza požadavků zahrnuje specifikaci požadavků zadavatele a analýzu budoucího softwarového produktu (dále také systém). Jedná se o nejdůležitější fázi vývoje, při které je podstatné formulovat požadavky zadavatele jasným způsobem. Analýza nebere žádným způsobem v úvahu, jak bude systém implementován, či na jaké platformě bude provozován.

Návrh systému navazuje na fázi analýzy a bere již v úvahu platformu, pro kterou bude systém implementován. Zahrnuje popis struktury systému, rozhraní komponent, uživatelského rozhraní, datový model apod. Mezi fází analýzy a návrhu je pouze slabá hranice, takže se často mohou tyto fáze prolínat.

Implementace je fáze zaměřená na programování systému a vychází z předcházející etapy návrhu. K implementaci může patřit také návrh konkrétních algoritmů či složitých databázových dotazů. Tím se vlastně programátoři prakticky podílejí i na etapě návrhu. Implementace bývá u mnoha projektů tou nejdelší fází, při které obvykle dochází ke vzniku nemalého počtu chyb. *Testování* a *ladění* mají za úkol tyto chyby odhalovat a opravovat. Tyto etapy bývají součástí všech fází, proto se většinou neuvádí jako samostatné fáze.

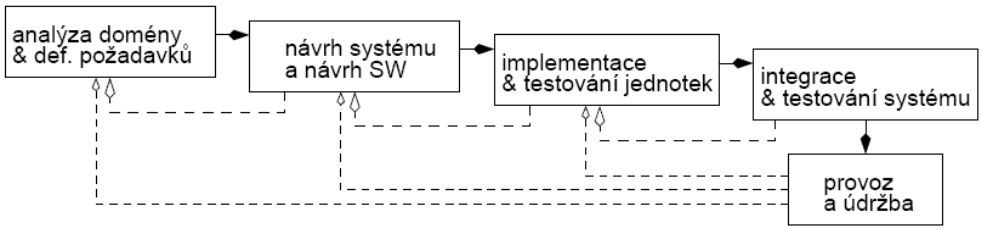
Integrace a *nasazení* zahrnuje sestavení systému z implementovaných a řádně otestovaných komponent. Funkčnost samostatných komponent nemusí znamenat, že po sestavení těchto komponent bude celý systém funkční. Z tohoto důvodu je součástí integrace testování, které se běžně označuje jako *integrační testování*. Při něm se funkčnost systému ověřuje postupně přidáváním dalších komponent. Po úspěšné integraci a otestování následuje nasazení systému u zadavatele. V této části přichází ke slovu opět testování. Tentokrát však obvykle nazývané jako *systémové testování*, které provádějí vývojáři za podmínek blízkých provozním podmínkám systému. Může se tak ověřit výkonnost systému, jeho bezpečnost, schopnost zotavení atd. Následuje ještě *přejímací testování*, jehož cílem je ověřit, že zákazníkovi je předáván produkt splňující všechny jeho původní požadavky. Touto etapou je systém předán zadavateli k užívání.

Provoz a *údržba* označuje etapu, ve které je systém používán a podle potřeb dále udržován a rozvíjen. Údržba systému bývá typicky plánovaná a náklady na ni jsou odhadnuty již v počátečních fázích životního cyklu vývoje systému. Opravná údržba má povahu odstraňování chyb a nedostatků odhalených při provozu. Adaptivní údržba má za cíl přizpůsobení systému např. změnám legislativy, vybavení zadavatele výpočetní technikou atd. Vylepšovací údržba se zaměřuje na rozvoj systému přidáváním dalších funkčností nebo zlepšováním kvality.

2.4.1.2 Modely životního cyklu vývoje

Životní cyklus software se modeluje pomocí *modelů životního cyklu* [10]. Tyto modely nepopisují konkrétní způsoby, jakými se má systém realizovat. Pouze definují jednotlivé fáze, jejich interakce a výstupy, popisují činnosti v průběhu procesu vývoje systému a umožňují průběžnou kontrolu. Jak budou jednotlivé fáze konkrétně provedeny, je ponecháno na každé organizaci. Konkrétní implementace se může lišit projekt od projektu, což vzhledem k unikátnosti každého projektu není nic neočekávaného.

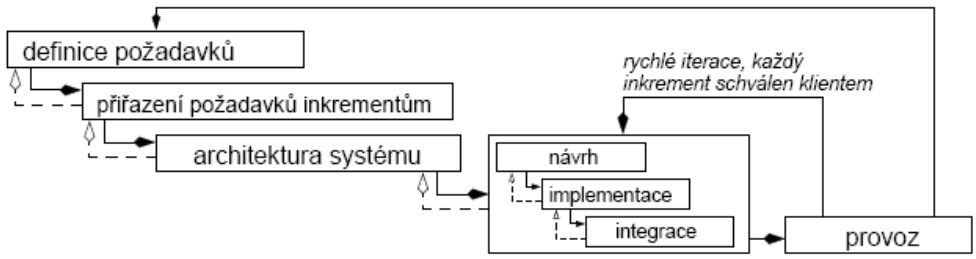
Model *vodopád* vznikl jako první model životního cyklu vývoje a byl úspěšně využíván k řešení spousty velkých projektů. Jak název napovídá, základem je následování jednotlivých fází postupně za sebou. Průchod všemi fázemi celého cyklu se provádí pouze jednou. Začátek následující fáze je dán dokončením fáze předchozí. Mezi těmito fázemi je často nutná zpětná vazba, která promítá provedené změny zpět do předchozí fáze. Zadavatel při použití tohoto modelu vstupuje jen do fáze analýzy a pak čeká na dokončení systému. Pro dnešní potřeby objektově orientovaného vývoje již tento model není vhodný a bývá nahrazen novějšími modely. Je možné setkat se s řadou různých variant tohoto modelu. Schéma klasického vodopádového modelu je zobrazeno na obrázku 3.



Obr. 3: Model životního cyklu vodopád se zpětnou vazbou [12]

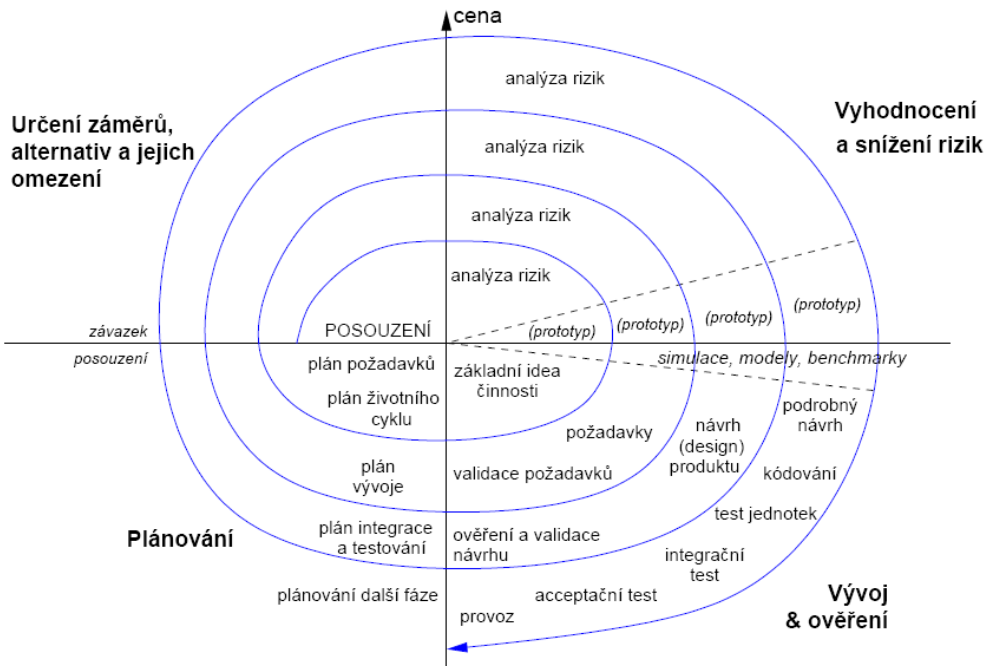
Iterační životní cykly s přírůstky (někdy označované jako inkrementální či evoluční) postupem času nahradily vodopádový model. Je pro ně charakteristické, že zahrnují opakované průchody jednotlivými fázemi s cílem rozšířit systém o nějaký přírůstek v každém opakování. Tento opakovaný průchod se příznačně nazývá *iterace*. Při každé iteraci vzniká nová verze vytvářeného systému obohacená o přírůstek v podobě nových funkčností, rozšíření či vylepšení. Každá tato nová verze má podobu celého systému s omezeným rozsahem funkčností ve spustitelném kódu, který se nazývá konstrukce (z angl. build). Výstup každé iterace lze tedy předvést zadavateli a jeho připomínky a další

požadavky promítnout do další iterace. Na obrázku 4 je zobrazeno názorné schéma celého procesu. Úloha zadavatele se v tomto modelu dostává do popředí a zvyšuje pravděpodobnost, že výsledný produkt bude odpovídat představám zadavatele. Typickým rysem iterací je krátká doba trvání. Obvykle se jedná o dny, týdny, maximálně měsíce. Tato vlastnost je výhodná právě pro řízení projektů, jelikož dobře umožňuje operativní plánování dalších iterací. Mezi nejvyužívanější iterační modely patří spirálový model, architektura řízená modelem (Model Driven Architecture – MDA) nebo RUP (IBM Rational Unified Process).



Obr. 4: Iterativní životní cyklus s přírůstky [12]

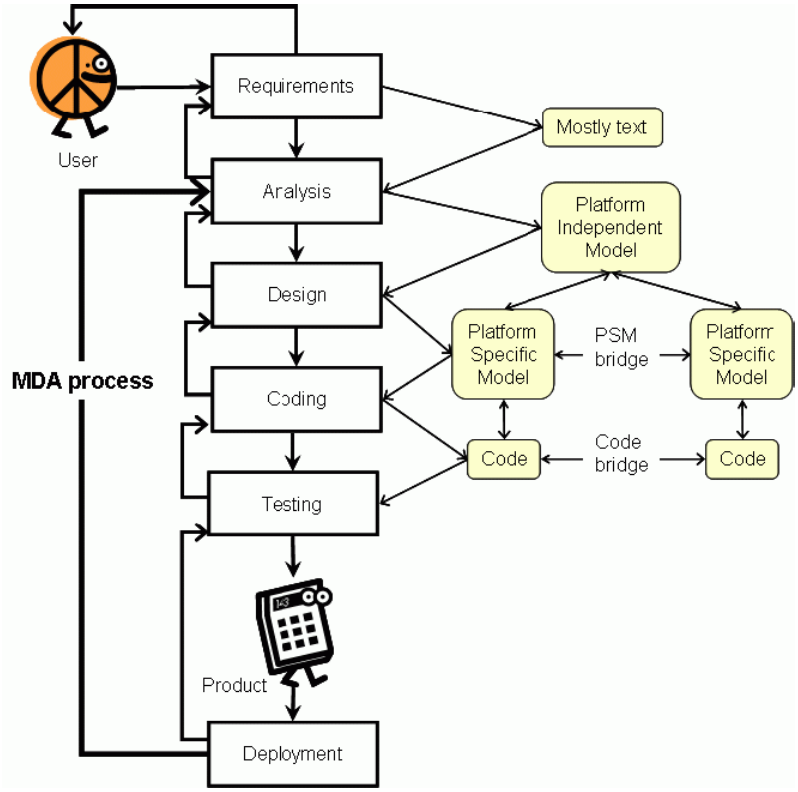
Spirálový model je spíše rámcem či referenčním modelem, který obsahuje jiné modely životního cyklu. Lze jej chápat jako způsob uvažování při vývoji software, které dává organizaci dostatečnou volnost při konkrétním využití modelu. Jak je vidět na obrázku 5, prezentuje se v podobě spirály. Ta prochází čtyřmi kvadranty, které představují fázi plánování, analýzu rizik, inženýrství a hodnocení zákazníkem.



Obr. 5: Spirálový model životního cyklu [12]

Vysoce iterativní charakter spirálového modelu je dán velkým důrazem na opakované plánování a hodnocení zákazníkem. Unikátností spirálového modelu je explicitní existence analýzy rizik, která se provádí při každé iteraci. Při první iteraci se získají od zadavatele požadavky na výsledný systém a projekt se naplánuje. V rámci analýzy rizik se posoudí předpokládané náklady a přínosy, rizika či hrozby, příležitosti plynoucí z řešení projektu atd. Výsledkem analýzy je rozhodnutí, zda se projekt bude řešit. Fáze inženýrství může podle různých interpretací představovat vlastní vývoj s výstupem v podobě konstrukce, zpracování analýzy požadavků a vytvoření prototypu nebo dokonce finální produkt realizovaný modelem vodopád. Po fázi inženýrství následuje hodnocení zákazníkem. Získané reakce ovlivňují další iteraci.

MDA je snahou o použití jazyka UML (z angl. Unified Modeling Language) jako spustitelné specifikace. Z reprezentace modelu v UML by se tedy vygeneroval celý softwarový systém. Podobně je tomu v hardwarovém inženýrství, kdy se generují podklady pro implementaci integrovaných obvodů ze specifikace zadané v jazyce VHDL (VHSIC Hardware Description Language). Model MDA přistupuje k vývoji systému jako k posloupnosti transformací z formální specifikace, přes návrh, až po spustitelný kód. Tento přístup by měl být výrazně automatizován. S výhodou se využívají různé CASE nástroje umožňující další strojové zpracování.

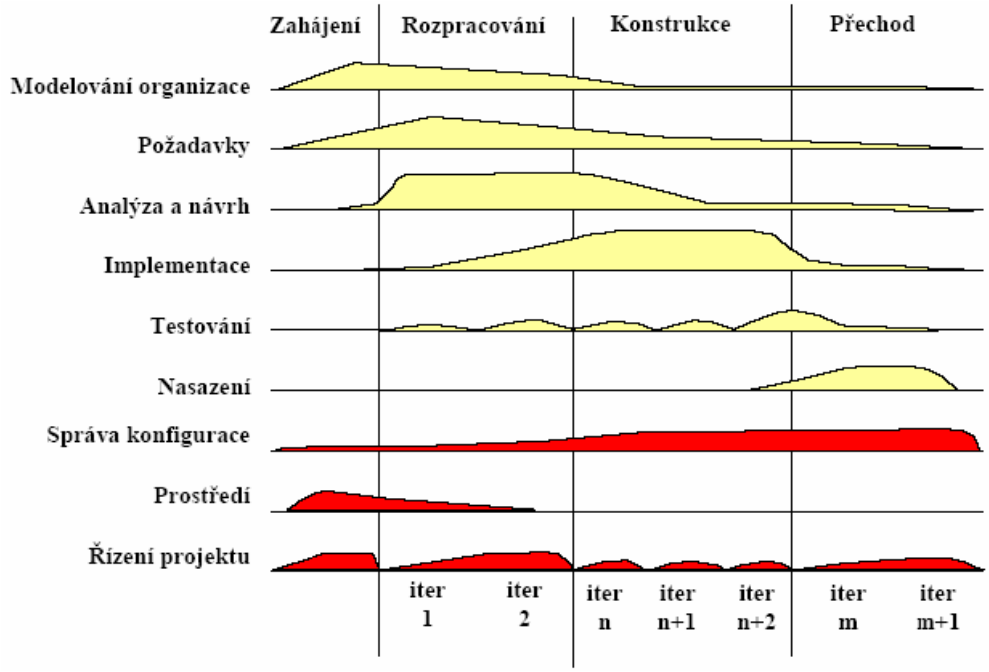


Obr. 6: Životní cyklus MDA [10]

Model *RUP* je v dnešní době jedním z nejznámějších modelů životního cyklu vývoje. Současně jej lze využít jako podpůrné prostředí, které vývojářům poskytuje dokumenty s online nápovědou,

šablony dokumentace a potřebné průvodce. Toto prostředí je součástí balíku CASE nástrojů dodávaných společností IBM. Dnes je RUP komerčním produktem firmy IBM, i když původně vznikl v tehdejší společnosti Rational. Existuje i otevřená verze, velmi blízká této komerční, s názvem Unified Process (UP), kterou vyvinuli autoři jazyka UML. Oproti UP je RUP úplnější, detailnější a dá se chápat jako konkrétní komerční podtřída, která rozšiřuje a modifikuje možnosti UP.

RUP nemá příliš jasnou strukturu a často bývá prezentován jako dvojdimenzionální, jak je patrné z obrázku 7. Horizontální směr označuje jednotlivé fáze životního cyklu – zahájení (inception), rozpracování (elaboration), konstrukci a přechod (transition). Tyto fáze představují dynamický aspekt životního cyklu, tj. jak je systém vyvíjen v průběhu času. Vertikální směr reprezentuje disciplíny, které se podílejí na životním cyklu – modelování organizace (business modelling), analýzu požadavků a návrh, implementaci, testování, nasazení a podpůrné aktivity jako je správa konfigurace a změn, řízení projektu a prostředí. Tyto disciplíny představují statický aspekt životního cyklu, tj. z jakých toků činností a aktivit se proces vývoje skládá. Toky činností probíhají souběžně. Objem prací jednotlivých činností se však liší podle aktuální fáze životního cyklu. Pro každou fázi je typická realizace několika iterací.



Obr. 7: Dvojdimenzionální znázornění RUP [10]

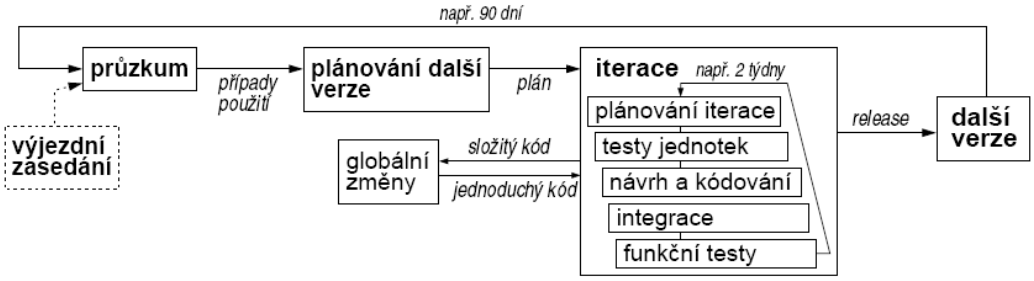
Výstupem každé iterace modelu RUP je spustitelná verze systému s omezenou funkčností. Každou iterací se funkčnost systému rozšiřuje, dokud se v rámci poslední iterace nedosáhne výsledné podoby systému. Obrázek 8 přehledným způsobem znázorňuje průběh iterace životního cyklu. Při každém průchodu proběhnou činnosti týkající se modelování. Následuje specifikace požadavků, analýza a návrh, implementace, testování a rozmístění (instalace). K tomu probíhá celá řada činností z

podpůrných toků, které se týkají správy konfigurací, řízení projektu a také přípravy prostředí, ve kterém je systém vyvíjen a nasazen [11].

Obr. 8: Iterace vývoje v rámci RUP [11]

Agilní vývoj je typem životního cyklu, který i přes svou revoluční povahu dobře zapadá mezi iterativní životní cykly. Vznikl jako reakce na potřeby rychlého vývoje, měnící se podmínky a zadání, které jsou typické např. u internetových projektů. Tento přístup se snaží o prosazení nových přístupů k vývoji softwaru, které upřednostňují fungující software před úplnou dokumentací, reakce na změny před striktním dodržováním plánu projektu, jednotlivce a interakce před procesy a nástroji, spolupráci se zákazníkem před vyjednáváním kontraktu. Průběžná spolupráce se zákazníkem také redukuje dokumentaci potřebnou pro přenos znalostí a požadavků. Tento přístup dává agilnímu vývoji charakter kreativního vývoje software. Fáze analýzy požadavků se u tohoto modelu označuje jako „uživatelské příběhy“ (z angl. user stories), kdy zákazník popíše, co by měl výsledný systém umět. Pro každý příběh se naplánuje doba trvání a předpokládané náklady. Zákazník si podle toho určí, co se bude implementovat v první iteraci vývoje. Pro agilní přístup je typický vývoj řízený testem, kdy vývojář ještě před implementací vytváří přejímací test. Ten zaručí, že zákazník dostane požadovanou funkčnost. Charakteristikou agilního vývoje také bývá programování ve dvojicích, kdy jeden programátor píše zdrojový kód a druhý jej kontroluje, případně navrhuje vylepšení apod.

Zřejmě nejznámějším představitelem agilního vývoje je v současné době extrémní programování. Doporučováno bývá pro případ rychle se měnících či nejasných požadavků na vyvíjený software. Použitelné je spíše pro malé týmy o maximálním počtu 10 lidí. Veškerý kód je psán ve dvojicích, což zaručuje kvalitu kódu a menší pravděpodobnost výskytu chyb. Nevýhodou může být absence dokumentace po skončení projektu. Jedinou formou dokumentace jsou případy použití a komentáře v kódu [12]. Schéma přístupu k extrémnímu programování je na obrázku 9.



Obr. 9: Extrémní programování [12]

2.4.2 Modelovací jazyk

Důležitým aspektem úspěchu softwarového projektu je kvalitní komunikace mezi vývojáři a zadavatelem. Svůj význam má samozřejmě také kvalitní komunikace mezi vývojáři navzájem. Prostředkem komunikace je jazyk [10]. Jelikož softwarové inženýrství je převážně o modelování, hovoří se v tomto případě o modelovacím jazyku. Ten slouží k vhodné reprezentaci vyvíjeného systému pro účely potřebné komunikace. Zadavateli slouží pro formulaci požadavků na budoucí systém, analytikovi pro specifikaci těchto požadavků, návrháři pro zachycení návrhu systému a programátorovi pro přesné zapsání implementace.

Jaké jsou požadavky na ideální modelovací jazyk? Měl by být použitelný v celém průběhu životního cyklu. Musí poskytovat dostatečné výrazové prostředky s minimem syntaktických konstrukcí. Neměl by obsahovat nejednoznačnosti. Hlavně by však měl být dostatečně srozumitelný pro širokou komunitu lidí. Bohužel žádný takový ideální modelovací jazyk neexistuje. V dnešní době je jako standard pro modelování přijímán jazyk UML, o kterém již byla zmínka u popisu modelu životního cyklu MDA. Jazyk UML je vizuálním modelovacím jazykem. Obvykle obsahuje také textové informace, které slouží pro upřesnění významu nebo v případě složitějších modelů slouží jako náhrada za nedostatečné výrazové prostředky. Při modelování vzniká model složený z typově různých diagramů. Syntaxe i sémantika symbolů použitých v diagramech je pevně dána definicí jazyka.

Modely systému lze klasifikovat podle různých hledisek. Podle fáze životního cyklu hovoříme o modelech analýzy, návrhu, implementace apod. Každý typ diagramu umožňuje modelovat pouze určité vlastnosti systému. Podle těchto vlastností rozlišujeme diagramy pro modelování struktury, chování, stavu systému a změn stavu. Vzhledem k zaměření této diplomové práce bude takto obecný popis modelovacího jazyka plně postačovat. Více informací lze nalézt v dostupné literatuře [10].

2.4.3 Plánování a řízení projektu

Řízení projektů je velmi obsáhlá disciplína a často bývá poměrně složitá. Samo o sobě však není projektové řízení zárukou úspěšného dokončení projektu. Vzhledem k unikátnosti každého projektu se některé metody mohou dobře osvědčit u projektu A, ale mohou vést k naprostému neúspěchu projektu B. Nejdůležitější osobou je projektový manažer, který na základě svých neustále rozvíjených

dovedností a zkušeností vybírá vhodné metody a vede projekt k úspěšnému dokončení [6]. Řízení projektů se skládá z různých manažerských činností, které musí pokrývat každou potřebnou aktivitu projektu. Tyto činnosti lze shrnout do procesu řízení o pěti krocích [5]:

- Definování
- Plánování
- Vedení
- Sledování (monitorování)
- Ukončení

Kroky definování a plánování nemusí být vždy nutně odděleny. Jejich pořadí také není úplně závazné. Krok *definování* je určen k definici projektových cílů. Běžnou praxí je začít s pracovní definicí, která je otevřena dalším společným jednáním a potřebným úpravám. Teprve během předběžného plánování vyjdou obvykle najevo některé důsledky původní definice. Konečná definice projektových cílů musí být měřitelná a dosažitelná.

Během kroku *plánování* se vytvoří plán projektu. Z něho je jasné, jak projektový tým splní cíle projektu. Plán je také základem pro sledování stavu projektu a jeho odchylek od původního plánu. Pro řízení projektu jsou zkrátka plánovací činnosti rozhodující. Dalo by se říci, že výsledný úspěch či neúspěch projektu závisí nejvíce na jeho kvalitním naplánování. Špatně provedená příprava se napravuje velmi obtížně. Pokud se navíc chybné plánování projeví až v průběhu realizace projektu, náklady na případnou nápravu bývají mnohonásobně vyšší.

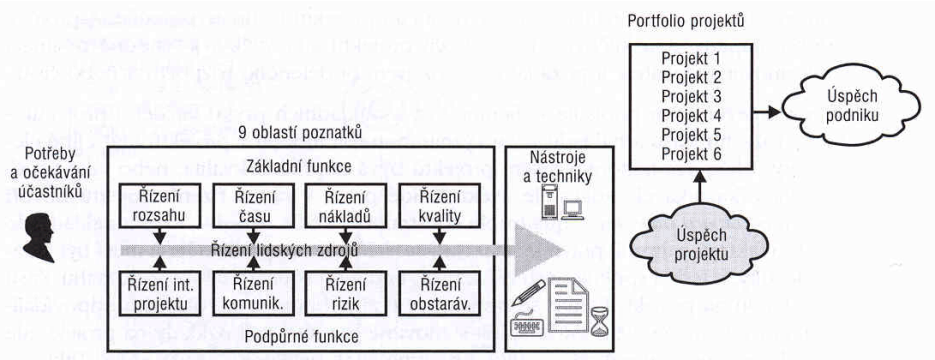
Vedení zahrnuje uplatnění manažerského stylu řízení lidských zdrojů, podřízených a dalších účastníků projektu. Při dobrém vedení svou práci všichni vykonávají efektivně a dodržují termíny.

Sledování slouží ke kontrole stavu a postupu projektových činností. Lze tak včas zjistit odchylky od plánu a opravit je. Tyto odchylky někdy mohou vést až ke změnám projektového plánu, k úpravě definice cílů či ke změně zdrojů. Málokterý projekt postupuje po celou dobu v souladu s plánem projektu. Téměř vždy je zapotřebí plán upravovat a často i měnit dohodnutou definici cílů.

Ukončení projektu zahrnuje ověření, že výsledek odpovídá definici cílů, a uzavření všech nedokončených činností a také dokumentace. Projekt může být definitivně ukončen až v momentě, kdy odpovídá aktuální definici cílů.

Projektové plánování a sledování projektu je každodenní činností projektových manažerů [10]. Prakticky se jedná o nepřetržitou činnost plánování, která zahrnuje odhad času, peněz, úsilí a zdrojů, které je potřeba vynaložit na projekt. Nejde jen o plánování, rozpočtování a sledování projektu. Mezi činnosti patří také analýza rizika, zajišťování kvality, řízení lidských zdrojů, řízení konfigurace atd.

Základní rámec celého řízení projektů z pohledu projektového manažera ilustruje obrázek 10. Tento obecný rámec obsahuje klíčové prvky – účastníky projektu, oblasti poznatků pro řízení projektů, nástroje a techniky pro řízení projektů, přínosy úspěšných projektů pro celou organizaci [6].



Obr. 10: Základní rámec řízení projektů [6]

Zmíněné *oblasti poznatků pro řízení projektu* charakterizují důležité schopnosti, které by si měl každý projektový manažer osvojit. Měl by mít dobré znalosti a zkušenosti ve všech devíti oblastech, protože každá je životně důležitá pro úspěch projektu.

Za *základní oblasti poznatků* lze považovat řízení rozsahu projektu, řízení času, řízení nákladů a řízení kvality projektu. Důvod je zřejmý vzhledem k již dříve uvedené definici cíle projektu, kde se objevily pojmy jako trojimperativ či čtvero omezení. Všechny základní oblasti poznatků vedou k naplnění konkrétních cílů projektu.

Další čtyři oblasti se označují jako *podpůrné oblasti poznatků*. Definují procesy, které pouze napomáhají k dosažení cílů projektu. Konkrétně se jedná o řízení lidských zdrojů, řízení komunikací, řízení rizik a řízení obstarávání v projektu. V následujícím textu alespoň stručně charakterizuji jednotlivé oblasti a uvedu některé z nástrojů a technik, které projektovým manažerům i jejich týmům pomáhají při řízení projektů.

Řízení rozsahu projektu zahrnuje definování a řízení veškerých činností, které jsou nezbytné k úspěšnému dokončení projektu. Mezi nejrozšířenější nástroje a techniky této oblasti patří stanovení rozsahu projektu, struktura rozpisu prací, pracovní příkazy, plán řízení rozsahu, analýza požadavků a řízení změn rozsahu.

Řízení času v projektu představuje odhad doby potřebné k dokončení činností, návrh časového plánu projektu a zajištění včasného dokončení celého projektu. K tomu pomáhají např. Ganttovy diagramy, síťové diagramy, analýza kritické cesty, plánování kritického řetězu, revize milníků atd.

Řízení nákladů se v projektu soustředí na přípravu projektového rozpočtu a jeho řízení. Používané nástroje a techniky jsou např. odhady nákladů, plán řízení nákladů, finanční software, návratnost investic, analýzy doby návratnosti, řízení portfolia projektů, čistá současná hodnota apod.

Řízení kvality projektu dohlíží na splnění původně stanovených požadavků a potřeb. Cílem je zajištění spokojenosti zadavatele. Mezi běžné nástroje a techniky patří již dříve zmíněná Six Sigma, diagramy řízení kvality, audity kvality, modely vyspělosti, statistické modely atd.

Řízení lidských zdrojů se zabývá nasazením pracovních sil do řešeného projektu a jejich efektivním řízením. Pro mnoho manažerů, vzdělaných spíše technicky, bývá řízení lidí tou nejtěžší

úlohou, kterou musí zvládnout. Při řízení napomáhají např. motivační techniky, empatické naslouchání, týmové smlouvy, matice přidělení odpovědností, cvičení při budování týmu atd.

Řízení komunikací představuje generování, shromažďování, distribuci a ukládání všech důležitých informací týkajících se projektu. V této oblasti jsou dostupné nástroje a techniky jako např. plán řízení komunikací, řízení konfliktů, výběr komunikačních médií, zpráva o stavu, šablony, weby projektů apod.

Řízení rizik zahrnuje včasné rozpoznávání rizik souvisejících s projektem, analýzu rizik a potřebné reakce na tato rizika. Pomocnými technikami jsou např. plán řízení rizik, hodnocení rizik, matice pravděpodobnosti a důsledků, metoda Monte Carlo atd.

Řízení obstarávání přestavuje pořizování zboží a služeb, které jsou pro řešení projektu potřeba, od vnějších organizací. K často používaným technikám patří smlouvy, žádosti o návrh nebo nabídku, vyjednávání o smlouvě, výběr zdroje, analýza „make-or-buy“ apod.

Devátou oblastí poznatků je *řízení integrace projektu*. Tato oblast je spíše průřezová, protože se vzájemně ovlivňuje se všemi ostatními oblastmi. Mezi používané nástroje a techniky patří např. metody výběru projektu, metodologie řízení projektů, analýza účastníků, plán řízení projektu, řízení konfigurace, software pro řízení projektů atd.

Je zřejmé, že projektoví manažeři nemají snadnou práci. Celá problematika řízení projektů je velmi široká a není úplně snadné do ni proniknout. Dobrý projektový manažer přistupuje ke každému konkrétnímu projektu s pečlivou rozvahou. Společně s ostatními klíčovými účastníky projektu definuje cíle a prostřednictvím vhodných nástrojů a technik se snaží dosáhnout úspěšného dokončení projektu. V dnešní době existuje naštěstí velké množství různého softwarového vybavení, které může manažerovi a jeho týmu v jejich nesnadném úkolu pomoci.

2.5 Programová podpora projektového řízení

Těžko lze jenom pomyslet, že by při vývoji IT projektů neexistoval nějaký software, který by projektové řízení usnadňoval. Vždyť dnes se používají počítačové programy snad v každém odvětví a sebemenším obchodním procesu. Programová podpora projektového řízení tedy má své opodstatnění a to nejen při řešení IT projektů.

Co by všechno měl umět takový software na podporu projektového řízení a v čem by mohl ulehčit práci manažerovi? Jak uvádí Milton D. Rosenau ve své knize [5] o softwaru pro řízení projektů: “Používat počítačový software k řízení projektů není totéž jako efektivně řídit projekt. Nicméně s pomocí tohoto softwaru může dobrý manažer projektu odvést lepší práci. Naopak ze špatného manažera projektu se nestane dobrý manažer, když po něm budete chtít, aby používal software k řízení projektů”. Software může manažera včas upozornit na možná rizika s ohledem na definované termíny a náklady. Umožní mu pečlivě celý projekt naplánovat, sledovat průběh a kontrolovat aktuální stav oproti plánu projektu. Software ale nedokáže řešit problémy, které nastanou

při realizaci projektu. Jak již v úvodu této práce zaznělo, řízení projektů se týká zejména řízení lidí. Počítačový program je mechanický nástroj, který lidské problémy nevyřeší.

Nevhodně zvolený nástroj může projektovému řízení spíše uškodit. Manažerovi by měl pomáhat a ne ubírat čas, který by měl raději věnovat svému týmu. Při výběru nástroje je důležité počítat s potřebným časem pro zaškolení členů týmu, aby software efektivně používali a rozuměli jeho výstupům. Velmi malé a jednoduché projekty obvykle použití projektových software nástrojů nevyžadují. Naopak by to práci spíše zdržovalo. Typickou chybou bývá nesprávný postup při výběru vhodného software. Chybný postup obvykle znamená vyhledání jakéhokoliv programu na řízení projektů, který poběží na současném vybavení týmu. Je doprovázen naivní představou, že se s tímto programem daný problém vyřeší. Správnou cestou je vždy nejdříve specifikovat řešený problém, pak vyhledat software vhodný na řešení tohoto problému a případně pořídit hardware pro bezproblémový chod vybraného software.

Komunity projektových manažerů a vývojářů software samozřejmě na potřebu vhodné softwarové podpory reagují. Institut PMI zveřejnil v roce 1999 zprávu Project Management Software Survey, ve které popisuje, porovnává a vyzdvihuje důležité rozdíly více než 200 softwarových nástrojů pro řízení projektů. Při projektovém řízení dosud příliš mnoho lidí využívá běžné kancelářské aplikace. Pomocí tabulkového procesoru či textového editoru stanovují rozsah projektu, časový plán a rozpočet, přiřazují zdroje, připravují projektovou dokumentaci atd. Dnes již existují stovky speciálních softwarových nástrojů, které nabízejí specifické funkce využitelné právě při řízení projektů. Dostupné jsou také komplexní nástroje určené pro celkové řízení projektů. V následující části uvádím shrnutí základních typů dostupných nástrojů. Rozděleny jsou podle rozsahu funkčnosti a ceny za software, jak je uvádí [6].

2.5.1 Nástroje nižší třídy

Pro řízení projektů poskytují spíše základní funkce a obvykle stojí maximálně 200 dolarů na jednoho uživatele. Jsou vhodné pro malé projekty a jednotlivé uživatele. Za zmínku stojí schopnost vytvářet Ganttovy diagramy, které se bez podpůrných nástrojů kreslí velmi obtížně.

Jako zástupce programů této skupiny lze uvést např. produkt Milestones Simplicity společnosti KIDASA Software, Inc., který se dá pořídit za 49 dolarů na jednoho uživatele. Pomocí průvodce časovým plánem umožňuje jednoduchými kroky vytvořit Ganttův diagram. Obsahuje nástroj pro vytvoření osnovy projektu, průvodce publikováním na Internetu, množství různých symbolů, flexibilní formátování atd.

Vzhledem k častému užívání běžných kancelářských produktů nabízí několik společností různé doplňky pro Microsoft Excel či Microsoft Access. Doplňky umožňují používat alespoň základní funkce pro řízení projektů v rámci dostupného software.

2.5.2 Nástroje střední třídy

Tyto nástroje již zvládají projekty většího rozsahu a současnou práci více uživatelů na více projektech. Pro obsluhu funkcí v pracovní skupině vyžadují některé nástroje instalaci samostatné serverové komponenty. Cenově se software této třídy pohybuje od 200 do 500 dolarů na jednoho uživatele. Samozřejmostí je zde dostupná podpora pro vytváření Ganttových a síťových diagramů, analýza kritické cesty, alokace zdrojů, sledování postupu prací na projektu, vytváření zpráv o současném stavu apod.

Nejpoužívanějším zástupcem nástrojů této třídy je Microsoft Project. Mezi další výrobce podobně zaměřeného software lze zahrnout společnosti Artemis, PlanView, Primavera a Welcom.

2.5.3 Nástroje vyšší třídy

Tyto špičkové nástroje bývají někdy označovány jako software pro řízení podnikových projektů. Často jsou integrovány s databázovým softwarem na podnikové úrovni a jsou také dostupné přes Internet. Licence se zpravidla k těmto produktům prodávají podle počtu uživatelů. Obsahují velice robustní funkce vhodné pro velké projekty, umožňují práci rozptýlených pracovních skupin, slučují informace z jednotlivých projektů, díky kterým lze vytvářet souhrnné pohledy na celkový průběh projektů v celé organizaci.

Své zástupce v této třídě nástrojů má např. společnost Microsoft se svým řešením Microsoft Enterprise Project Management Solution. Někteří výrobci nástrojů střední třídy dnes již nabízejí i podnikové verze. Na trhu lze nalézt také několik levnějších produktů, které jsou webově orientované, jako např. VMPi Enterprise Online za 12 dolarů za měsíc na jednoho aktivního uživatele.

Gigantická společnost typu Microsoft samozřejmě potřebuje efektivní způsoby projektového řízení a vývoje software. Není proto divu, že v této oblasti vyvíjí Microsoft stále optimálnější a užitečnější řešení. Velmi zajímavým počinem tohoto průkopníka v IT je řada produktů Visual Studio Team System [15]. Tomuto produktu, souvisejícím metodikám a efektivnímu přístupu k řízení a vývoji IT projektů se budu podrobněji věnovat v následující kapitole.

2.6 Efektivní softwarové projekty

V následující části bych rád shrnul některé poznatky z velmi zajímavé knihy od Sama Guckenheimera [15]. Sam je vedoucím plánovačem řady produktů Microsoft Visual Studio Team System (dále jen VSTS) a dříve se staral o produktovou strategii ve společnosti Rational Software Corporation (nyní součást IBM). Kromě významu a možností VSTS se pokusím tlumočit některé myšlenky tohoto uznávaného odborníka.

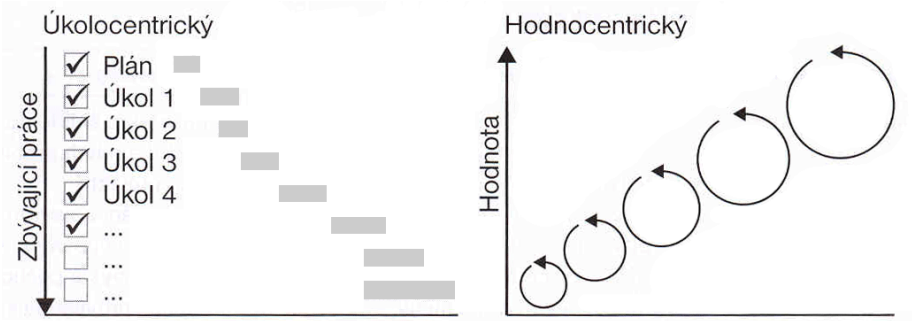
2.6.1 Teorie prověřená praxí

Za přínosné myšlenky ovlivněné praxí pokládám autorovo rozdělení a srovnání procesů podle „hodnotocentrického“ a „úkolocentrického“ přístupu. Dále bych zmínil užitečnost sledování denních činností, což poskytuje širší možnosti pro softwarové metriky a užitečné techniky převzaté z agilního vývoje projektů. Při pohledu na historii se zdá, že IT obor zpočátku spíše automatizoval staré ruční procesy. Zlom nastal v době, kdy se strategie začaly ubírat cestou nové reformy procesů s využitím automatizace. Tím se dobré procesy zbavují zbytečné režie. Zvyšuje se individuální produktivita všech rolí týmu. Ze všech členů týmu se stává výkonný a sehraný celek. Cíl je jediný: efektivní vývoj softwarových projektů.

Jak vypadá neefektivní vývoj? Spousta používaných softwarových metodik v praxi vyžaduje pracnou ruční realizaci. To velmi prodražuje a komplikuje sběr dat a sledování postupu. Požadované množství dokumentace, způsob vedení, provoz a údržba jsou zatíženy vysokými režijními náklady. Vznikají tak činnosti a nástroje, které jsou členové vývojového týmu nuceni používat, většinou s negativním ohlasem, a ještě to nepřiměřeně prodražuje a komplikuje softwarový vývoj. Převládá tzv. *úkolocentrický přístup*, jež nahlíží na softwarové inženýrství jako na jasně vymezenou činnost podobnou běžným inženýrským pracím.

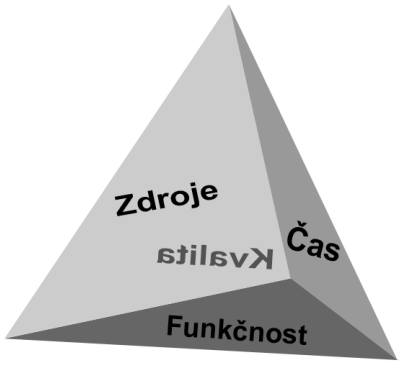
Jako reakce na zmíněné problémy a neefektivní vývoj přichází na svět odlišný přístup. Pružný a dostatečně efektivní způsob. Zvládá řešit případy, kdy vývojový tým musí v rámci každodenní práce přijmout nové myšlenky hodnoty pro zákazníka, změny, odchylky či reakce na aktuální situaci. Tato flexibilita musí být zachována také u projektů zajišťovaných externě, u projektů místních či zeměpisně rozptýlených. Převládá tzv. *hodnotocentrický* přístup, který se maximálně vyhýbá zbytečným režijním nákladům.

Úkolocentrický a hodnotocentrický přístup se v základu liší zejména výchozím měřítkem. Úkolocentrický přístup považuje projekt za pevný seznam oceněných úkolů, které je třeba dokončit. Výdaje měří podle plnění těchto úkolů. Hodnotocentrický přístup měří hodnotu, která je v daných okamžicích dodávána zadavateli. Na vstupy nahlíží jako na proměnlivé toky, nikoliv jako na pevnou zásobu. Tento základní rozdíl je názorně vidět na obrázku 11.



Obr. 11: Základní rozdíl mezi úkolocentrickým a hodnotocentrickým přístupem [15]

Úkolocentrický (work-down) přístup je způsob vedení projektu, ve kterém je centrálním motivem posloupnost konkrétních úkolů. Úkoly a jejich vazby jsou naplánovány podle dostupnosti a závislosti zdrojů. Takto vznikne výsledný projektový plán. Průběh projektu se sleduje jako postupné dokončování jednotlivých úkolů. U tohoto přístupu je zásadní, aby činnosti plánování a návrhu byly naprosto bezchybné. Během plnění plánu by se již neměly provádět žádné změny plánu. Všechny průběžné výsledky lze měřit. Výsledná kvalita se určuje splněním specifikace, která musí být správná již od začátku. Hlavním rysem úkolocentrického přístupu je řízení projektu skrze již dříve zmíněný trojimperativ. V tomto případě se však hovoří o tzv. „železném trojúhelníku“, resp. čtyřstěnu, pokud mezi proměnné čas, zdroje a funkcionalitu zahrneme také kvalitu. Železný trojúhelník, který znázorňuje obrázek 12, bere projekt jako pevnou sumu práce v klasickém smyslu úkolocentrického přístupu. Vazby proměnných jsou pevné a není tu prostor pro zvýhodňování některé z proměnných. Pokud je potřeba zvětšit jednu stěnu, musí se zvětšit také všechny ostatní stěny. Všechny stěny jsou spojeny, takže nelze natahovat jednu stěnu bez ohledu na ostatní.



Obr. 12.: Železný trojúhelník (nebo čtyřstěn) [15]

Tento přístup byl dříve často uplatňován a v mnoha případech je tomu i dodnes. Bohužel se ukazuje, že moc dobře nefunguje. Železný trojúhelník totiž předpokládá, že produktivita zdrojů je rovnoměrně rozložena, účinnost dokončování úkolů netrpí odchylkami a kapacita systému je plně využita. To jsou podmínky, které se vyskytují jen u mála projektů s nízkým rizikem. Ale u většiny realizovaných softwarových projektů tyto podmínky naplněny nejsou.

Stačí uvést příklad, kdy se zlepšením kvality zkrátí potřebný čas na realizaci projektu. Také lze bez změny objemu zdrojů nebo času výrazně zlepšit *tok (flow),* což je jakýsi plynulý pohyb hodnoty systémem. Hodnota rovnoměrně prochází jednotlivými fázemi transformace a s plynoucím výstupem se dostává k zákazníkovi jako fungující kód. Plynulý tok hodnoty v průběhu projektu je jednou z věcí, které musí manažer zajistit. Tzv. „tokový přístup k plánování“ vyžaduje, aby objem rozdělané práce byl co nejmenší. Tím se snižuje riziko nepředvídaných problémů a nutnost přepracování již hotové práce. To však vyžaduje každodenní měření toku dokončenosti. Manažer zajišťuje potřebnou rychlost průchodu jednotlivých pracovních činností od nápadu až po výstup ze systému. Pokud zná dostatečně

dobře systém, dokáže snadno zefektivnit vývoj zrychlením výroby, snížením realizačního času, včasným rozpoznáním překážek a prováděním takových opatření, které procesy ochrání a vhodně je využijí.

Zmíněných vlastností a nedostatků železného trojúhelníku si všímají agilní metody. Ke slovu se dostává efektivnější způsob vývoje softwarových projektů, tzv. hodnotocentrický přístup. Úkolocentrický přístup do procesu vývoje software vnesl řadu problémů, které v mnohých případech vedly k neúspěchu těchto projektů. Hodnotocentrický přístup má tedy obtížný úkol vyřešit stávající problémy a přitom nepřinést problémy nové.

Hodnotocentrický (value-up) přístup se podobá svým charakterem iterativním a agilním způsobům vývoje. Zaměřuje se na průběžné vytváření hodnoty, kterou lze zákazníkovi v daný okamžik dodat. To kromě jiných výhod přináší zvýšení návratnosti investic. Zákazník je více zapojen do častých konzultací. Tím je dosaženo spolehlivějších výsledků projektu. Díky většímu zapojení zákazníka do projektu se tento přístup nemusí zabývat zbytečně detailní a kompletní dokumentací, která znamená další režii a práci navíc pro členy vývojového týmu. Vzhledem k iterativnímu způsobu vývoje je třeba předpokládat nejistotu a možné dynamické změny v průběhu projektu. Je dobré tyto nejistoty předvídat a přizpůsobovat se momentálním požadavkům. Efektivnosti se dosahuje nasazováním strategií a metodik, které odpovídají aktuální situaci. Hodnotocentrický přístup se soustředí na jednotlivce a skupiny týmu. Vytvořením vhodného prostředí s dostatečnou motivací lze povzbudit k větší tvořivosti a vynalézavosti členů týmu. Dobrý výkon se podpoří skupinovou zodpovědností za výsledky a sdílením zodpovědnosti za efektivitu celého týmu.

Hodnotocentrický přístup se snaží odbourávat nevýhody přístupu úkolocentrického. Zejména velké množství zbytečné režie. K tomu potřebuje dostatečnou podporu nástrojů. Sbírat, udržovat a vyhodnocovat data bez režijních nákladů by nebylo možné, pokud by pro tyto činnosti neexistovaly vhodné nástroje s dostatečnou automatizací. Pokud je nutné vyhovět předpisům a umožnit audity, jsou tyto nástroje nepostradatelné pro správu změn a vedení dokumentace vyžadovanou auditem.

Hlavní rozdíly mezi úkolocentrickým a hodnotocentrickým přístupem jsou přehledně zachyceny v tabulce 1, jak je uvádí [15]. Při pečlivém srovnání zmíněných předpokladů je zřejmé, jak jsou tyto přístupy protikladné.

Výchozí předpoklad	**Úkolocentrický přístup**	**Hodnotocentrický přístup**
Plánování a změny	Plánování a návrh jsou činnosti, které musejí být bezpodmínečně bezchybné. Musíte je provést nejdříve, rozdělit zodpovědnost za plán, sledovat plnění plánu a pečlivě bránit tomu, aby se do něj nedostaly nějaké změny.	Změna je přirozená; smiřte se s tím. Plánování a návrh probíhá během celého projektu. Úvodnímu plánování a návrhu by proto mělo být věnováno jen tolik prostředků, abyste pochopili rizika a mohli provést další malý inkrement.
Primární měřítko	Dokončování úkolů. Protože známe kroky vedoucí ke konečnému cíli, můžeme měřit všechny průběžné výsledky a již získanou hodnotu vypočítat jako poměr spotřebovaného a celkového naplánovaného času.	Počítají se jen výsledky, které jsou pro zákazníka hodnotné (fungující software, hotová dokumentace atd.). Tok pracovních proudů musíte měřit frontami produkujícími hodnotu pro zákazníka a všechny průběžné míry musíte brát s odstupem.

Definice kvality	Splnění specifikace. Proto musí být správná hned na začátku.	Hodnota pro zákazníka. Její vnímání se může změnit (a pravděpodobně k tomu dojde). Zákazník může být schopen kvalitu formulovat teprve tehdy, když poprvé dostane fungující software. Možnosti proto ponechejte otevřené, přizpůsobte se neustálému dodávání a specifikaci nekonkretizujte předčasně.
Přijatelnost odchylek	Úkoly lze rozpoznat a odhadnout dopředu. Odchylkami se nemusíte zabývat.	Odchylky tvoří součást všech procesních toků, přirozených i umělých. Abyste dosáhli předvídatelnosti, musíte odchylky pochopit a snížit je.
Průběžné výstupy	K rozdělení návrhu a naplánování úkolů jsou nezbytné dokumenty, modely a jiné průběžné artefakty, které vedle toho umožňují měřit stávající pokrok.	Průběžná dokumentace by měla minimalizovat nejistotu a odchylky tak, aby se zlepšil tok výstupů. Cokoliv navíc je zbytečné.
Přístup k řešení problémů	Omezení času, zdrojů, funkcionality a kvality určují, co můžete dosáhnout. Když změníte jedno, musíte tomu ostatní přizpůsobit. Pečlivě hlídejte změny, aby se do plánu nedostaly takové, se kterými se nepočítá.	Omezení se mohou a nemusejí týkat času, zdrojů, funkcionality nebo kvality. Místo toho najděte hlavní slabé místo omezující tok hodnoty, pracujte na něm, aby přestal být nejdůležitější, a potom přejděte k dalšímu. Hladší tok zajistíte dalším snižováním odchylek.
Přístup k důvěře	Lidi je nutné sledovat a porovnávat se standardy. Vedení by mělo jednotlivce za jejich výkon při plnění plánu odměňovat.	Radost z dobře odvedené práce a týmové spolupráce je lepší motivací než individuální odměny. Důvěryhodné průhledné prostředí, kde všichni členové týmu mohou vidět celkovou výkonnost, funguje lépe než nařízení shora.

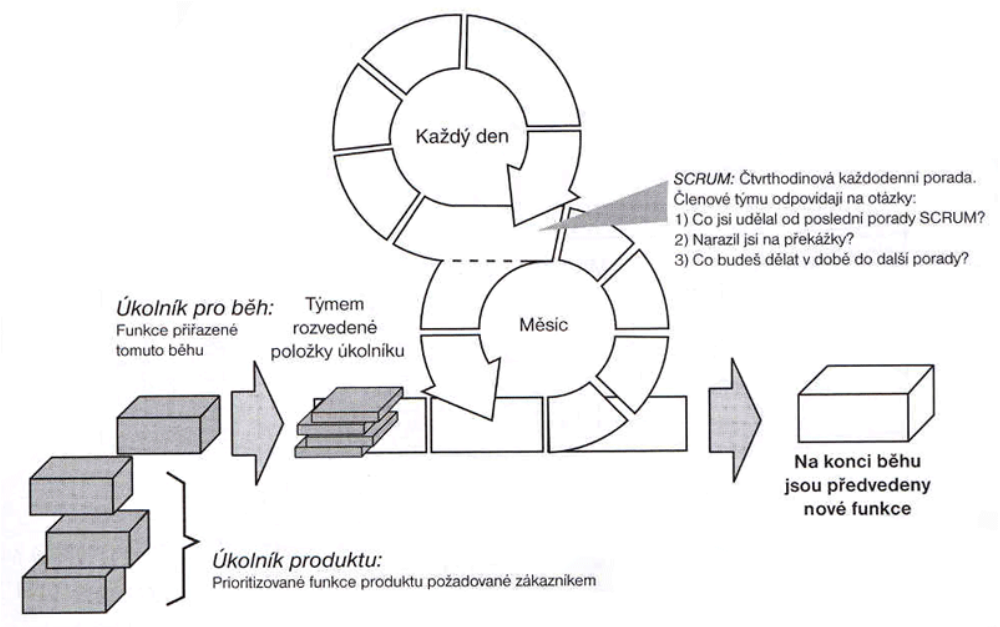
Tabulka 1: Hlavní rozdíly mezi úkolocentrickým a hodnotocentrickým přístupem [15]

Jak již bylo několikrát řečeno v předchozích kapitolách, většina softwarových projektů trpí nedodržením termínu. Na tomto místě je vhodné zmínit záludný kruh, do kterého se může vývojový tým dostat při zjištění časového skluzu. Taková zjištění a následná přeplánování či řešení problému příliš neprospívá efektivnímu toku. Začne se přeplánovávat, vzniká tlak na optimističtější odhady, které opět vedou k dalšímu zpoždění atd. Výsledkem je neřešitelný kruh optimistických odhadů a neustálého přeplánovávání. Kde je vlastně problém? Neumějí snad projektoví manažeři dobře plánovat a členové týmu řídit svůj čas? V některých případech možná ano, ale hlavní problém bývá spíše v rozporu mezi prioritami a očekáváním jednotlivých členů týmu. Jak k němu dochází?

Pokud porovnáme většinu metodik projektového řízení a samozřejmě i dostupných nástrojů, zdá se, že každý přístup má svá pracovní data o projektu uložena na mnoha místech. Za pracovní data v tomto případě považujme pracovní tabulky, projektové plány, databáze požadavků, databáze chyb, systémy pro správy testů, zápisy z jednání apod. Právě zmíněné rozptýlení informací na mnoho míst vede k nejasné představě o celém projektu. Je potřeba přistoupit k mnoha zdrojům a pracně dostat všechny potřebné údaje do časového plánu projektu. Navíc ve vhodné vyrovnané podobě. Pokud je zdrojů mnoho, nemusí být také už nalezená informace aktuální. Z toho vyplývá, že sjednocení a zviditelnění veškeré práce pro všechny členy týmu může být velice prospěšné. Je dobré, pokud každý člen týmu vidí, co může od kolegů očekávat a má tak ucelenou představu o průběhu projektu. To je samozřejmě podmíněno iterativním vývojem projektu, kvalitním plánováním a efektivní úpravou priorit pracovních úkolů podle dostupných zdrojů v daném cyklu vývoje.

Zmíněná fakta měla za následek snahu o využití jednotného transparentního *úkolníku produktu*. Ten obsahuje veškeré práce týkající se vyvíjeného produktu. Úkolník je tedy řada obchodních i technických funkcí, které mají svou prioritu a které musejí být do produktu zabudovány. Z jiného pohledu je to seznam všech schopností, funkcí, technologií, vylepšení či oprav, které se budou realizovat v dalších verzích produktu. Úkolník tedy vhodným způsobem nahrazuje více zdrojů informací o projektu a stává se jediným, udržovaným zdrojem informací o již provedené i zbývající práci. Tato transparentnost a jednotnost znamená ohromné zvýšení výkonu. Každý člen týmu vidí stejná data a celý projektový plán. Díky tomu si je také vědom, že na dokončení jeho pracovního úkolu čeká některý z kolegů. Tím vzniká vazba mezi týmovou a individuální zodpovědností. Ve spojení s každodenním sledováním toku vytváří jednotný úkolník v celém týmu atmosféru důvěry. Cesta k efektivnějšímu vývoji software je tedy otevřena.

Jednou z agilních metodik, která prosazuje jednotný úkolník, je metodika SCRUM (také označována jako „Scrum“). Princip metodiky SCRUM výstižně znázorňuje diagram na obrázku 13. V podkapitole věnované VSTS se zmíním mimo jiné o tom, jak myšlenku transparentního úkolníku využívá VSTS a rozšiřuje ji mnohem dále.



Obr. 13: Základní diagram metodiky SCRUM [15]

Každý projekt je unikátní a proto nemůže žádná metodika vyhovovat dokonale všem projektům. Výběr té nejvhodnější metodiky pro konkrétní projekt závisí na spoustě vlastností, např. velikost projektu, schopnosti týmu, zeměpisné rozmístění, rizika, předpisy apod. Vhodnou metodiku je následně dobré přizpůsobit konkrétním potřebám projektu a určit tak způsob dohlížení nad jejím dodržováním, průběh práce, opatření, šablony, reporty, oprávnění apod.

2.6.2 Visual Studio Team System

VSTS je řada produktů, které dohromady vytváří nástroj pro správu životního cyklu softwarového projektu. Dokáže se soustředit na potřeby víceúčelových rolí v rámci organizace. Jednotlivé produkty jsou zaměřeny na specifické potřeby cílových rolí. Lze si vybrat z pokročilých vývojářských nástrojů, vizuálních návrhářů pro návrh architektury, pokročilých nástrojů pro testování či nástrojů pro testování a nasazení databázových projektů. Samozřejmě existuje součást, která tyto produkty vhodně spojuje a také součást, která umožňuje všem členům týmu řídit a sledovat průběh svých projektů velice jednoduchým způsobem.

Celý balík VSTS je od začátku navržen takovým způsobem, aby efektivně využíval hodnotocentrický přístup. Musí umožnit postupy podle různých metodik a být dostatečně univerzální, aby jej bylo možné přizpůsobit specifickým potřebám různých organizací a projektů. Poskytuje také vše potřebné pro případné provádění auditů. Ze všech softwarových řešení, na které jsem narazil, považuji řadu produktů VSTS za nejzajímavější a nejefektivnější způsob vývoje, plánování a řízení projektů. Vzhledem k velikosti společnosti Microsoft je zcela pochopitelné, že se takový gigant bude snažit o co nejefektivnější zvládnutí procesů v softwarovém vývoji. Ale možná jsem jen příliš pozitivně ovlivněn zajímavou a přínosnou knihou [15].

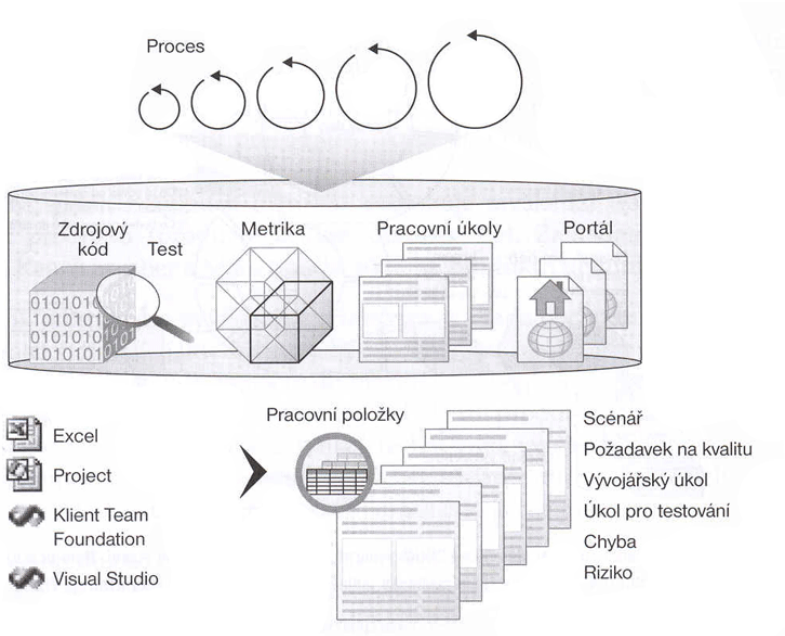
Výše v textu jsem se zmínil o nutnosti přizpůsobení metodice konkrétnímu projektu. VSTS s touto nutností počítá. Při vytváření projektů nabídne na výběr základní šablony metodiky a umožní její další přizpůsobení konkrétním potřebám projektu. Součástí VSTS je metodika Microsoft Solutions Framework (MSF), která zahrnuje následující dvě varianty:

- MSF for Agile Software Development
- MSF for CMMI Process Improvement

Jak již napovídá název, první varianta bude výhodnější pro agilní způsoby vývoje s důrazem na výsledek a bez nutnosti vytváření velkého množství dokumentace. Jde o pružný rámec schopný se přizpůsobit. Zaměřuje se na dodávání fungujícího software, většinou v kratších iteracích. Druhá varianta je zástupcem těžkopádnějších metodik se zaměřením na formálnější plánování, více dokumentace a pracovních výsledků. Tuto variantu využívají firmy, které se soustředí na optimalizaci svých procesů, audity a které chtějí vlastnit certifikaci. MSF ale celý rámec CMMI aplikuje agilně, bez zbytečné režie a s hodnotocentrickým přístupem. Obě varianty MSF jsou tedy hodnotocentrické a využívají ověřené postupy Microsoftu a oborových zkušeností.

V předchozí podkapitole byla zmíněna myšlenka jednotného transparentního úkolníku. VSTS používá jednotný úkolník pro veškerou práci týmu, ať už plánovanou, aktivní či hotovou, a to včetně historie provedených akcí a všech rozhodnutí, které se týkají činností během projektu. VSTS umožňuje uživatelům všechny tyto pracovní položky prohlížet či upravovat v několika různých nástrojích, např. ve Visual Studiu, Microsoft Excelu či Microsoft Projectu. Samozřejmě se také stará o neustálou synchronizací se společnou databází. Jedno úložiště dat, ale mnoho možností, jak s těmito

daty pracovat. Dokonce je možné sledovat pracovní položky v dnes již velmi rozšířeném komunikačním nástroji Microsoft Outlook. To však neznamená, že by bylo nutné používat více nástrojů. Úkoly, požadavky i chyby lze prohlížet na jednom místě. Výhoda přístupu VSTS k úkolníku je zřejmá. Jedna společná databáze umožňuje soudržnost informací na jednom místě. Tyto data jsou navíc přístupná z jednoho či více běžně používaných nástrojů. Potřebná data se již netříští mezi různé zdroje a tím odpadá nutnost vyhledávání a kopírování dat, což dosud neúměrně navyšovalo režii. Všichni členové týmu vidí tytéž pracovní činnosti, které jsou vždy aktuální bez ohledu na to, kdy byly naplánovány či upravovány. Sběr dat probíhá většinou automaticky, což snižuje další zbytečnou režii a umožňuje efektivnější sledování projektu. Obrázek 14 znázorňuje schéma, jak VSTS spojuje zdrojový kód, testování, pracovní položky i metriky do jednoho celku a umožňuje tak řídit celý proces vývoje. Pracovní položky jako např. scénáře, požadavky, úkoly pro vývojáře či testery, chyby, rizika apod. lze prohlížet a upravovat v běžných nástrojích, na které jsou členové týmu zvyklí. Konečně nejsou členové týmu nuceni k používání složitých nástrojů, mohou se již plně a efektivně věnovat své pracovní činnosti a pro potřeby projektového řízení používají běžné nenáročné nástroje.



Obr. 14: VSTS spojuje zdrojový kód, testování, pracovní položky a metriky dohromady

3 Analýza

Cílem této fáze je analyzovat požadavky na programovou podporu správy IT projektů s důrazem na týmovou spolupráci. Obvykle v této fázi analytik provádí definici problému, která vychází z úvodních jednání se zadavatelem. Sadou pohledů a otázkami v této definici ujasní se zadavatelem zadání projektu, které se rozpracuje ve formě úvodní studie. Tuto studii lze chápat jako předběžnou neformální specifikaci požadavků, která dává již celkem stabilní představu o funkčnosti, kterou by měl systém zadavateli nabízet. Dodavatel řešení provede předběžný plán a rozpočet projektu, analýzu rizik, studii proveditelnosti a následně předá zadavateli cenovou nabídku.

V případě, že zadavatel bude s nabídkou souhlasit, přichází na řadu nejdůležitější část celého životního cyklu vývoje – specifikace požadavků. Analytik ve specifikaci požadavků pomocí typů jednání vyjádří, jak bude systém používán. Samozřejmostí jsou důkladná jednání se zadavatelem, který se může ke specifikovaným typům vyjádřit. Vztah mezi budoucími uživateli systému a typy jednání doplňují diagramy případů použití (Use Case Diagram), které definují chování systému bez ohledu na vnitřní strukturu systému. Vznikne tak formální a úplná specifikace požadavků, ze které vychází návrh systému.

Během analýzy jsem procházel různá dostupná řešení a srovnával jejich výhody, nevýhody, uživatelskou přívětivost a složitost ovládání. Z načerpaných zkušeností jsem vytvořil přehled možných funkcí systému, které jsou podle mě důležité, nikoliv však příliš složité pro uživatele. Důležité pro mě byly poznatky z praxe, které jsem sám získal nebo které jsem čerpal z častých diskuzí s konzultantem diplomové práce. Jak již jsem zmínil v úvodu, konzultantem mé práce byla firma KomTeSa, zejména pak její jednatel Ing. Stanislav Piskač. S ním jsem často konzultoval požadavky na funkce systému. Požadavky se během dlouhé fáze analýzy několikrát měnily a přizpůsobovaly. Bohužel se stávalo, že jsem se z fáze návrhu musel vrátit zpět do etapy analýzy a provádět nemalé úpravy. Podobně tomu bylo i ve fázi implementace. Důvody vidím v nepříliš konkrétní specifikaci požadavků a málo detailním návrhu řešení. Dalším důvodem byla dodatečná doporučení na změny ze strany konzultanta projektu, většinou doprovázená příklady z praxe.

Po několika konzultacích specifikace požadavků jsem došel k následujícímu řešení. Navržená webová aplikace bude vycházet z některých funkčností intranetového informačního systému ESO9, který KomTeSa integruje svým zákazníkům, a sama jej už dlouhá léta používá pro vlastní chod firmy. ESO9 je velmi propracovaný informační systém. Z toho mimo jiné plyne jeho složitost a množství dostupné funkčnosti. Cílem mých dalších analýz bylo definovat pouze nutnou funkčnost aplikace, která umožní její propojení s informačním systémem a dovolí automatickou synchronizaci potřebných dat. Požadavkem na spolupráci aplikace s informačním systémem vstupují do projektu různá omezení a doporučené způsoby řešení. ESO9 má velmi dobře navržený datový model. Díky němu je informační systém univerzální a lze jej přizpůsobit potřebám prakticky jakékoliv organizace.

Vzhledem ke složitosti datového modelu ESO9 jsem se během analýzy pokusil definovat potřebné části stávajícího modelu, abych mohl při návrhu zajistit dostatečnou kompatibilitu obou systémů.

Po stránce dokumentační jsem do analýzy zahrnul neformální specifikaci požadavků a odpovídající diagram případů užití. Diagram představuje jednotlivé funkce systému, které by uživatelé měli mít k dispozici. Rozsah analýzy jsem se snažil optimálně přizpůsobit a definovat co nejvíce potřebné, přínosné funkčnosti, které by mohl systém uživatelům poskytovat. Při implementaci se však z časových důvodů zaměřím na realizaci těch nejnutnějších funkcí.

3.1 Neformální specifikace požadavků

Cílem projektu je programová podpora pro správu IT projektů. Oblast vyvíjeného software se tedy týká projektového řízení. Výsledná aplikace by měla umožňovat přehledným způsobem řídit projekty a v rozumné míře je také plánovat. Rozsahem se systém nepotřebuje vyrovnat profesionálním a drahým nástrojům typu Microsoft Project. Zadavatel požaduje nástroj, který jej nebude při správě projektů zbytečně zdržovat, ale naopak intuitivním a jednoduchým ovládáním zjednoduší a zefektivní projektové řízení. Organizace se zabývá vývojem projektů spíše menšího a středního rozsahu. Proto je soustředěna pozornost více na procesy operativního charakteru a agilní metody vývoje software. Především tedy zadávání úkolů a sledování jejich stavu a plnění.

Výsledný systém by měl mít podobu webové aplikace, která bude bez problému funkční na běžných kancelářských počítačích s internetovým prohlížečem a připojením do internetové, případně intranetové sítě. Aplikace bude provozována na běžném webovém serveru, který bude přístupný z internetu, případně intranetu. Aplikace bude využívat klasické třívrstvé architektury. Na webovém serveru bude aplikační logika, na databázovém serveru celá databáze a klienti se k aplikaci připojí pomocí webového prohlížeče. Předpokládaný počet uživatelů využívajících aplikaci ve stejnou chvíli je maximálně několik desítek.

3.1.1 Přehled očekávaných funkcí

Systém musí být dostatečně zabezpečený a uživatelská oprávnění by měla být navržena vhodně tak, aby každý uživatel měl přístup jen k těm funkcím, které skutečně potřebuje a naopak by neměl mít možnost provádět v systému takové činnosti, které mu nejsou z podstaty jeho role na projektech či na konkrétním projektu povoleny. Prvotní funkcí je tedy zabezpečené *přihlášení uživatele* do systému.

Ještě před specifikací dalších funkcí je třeba definovat význam termínu proces v tomto systému. *Procesem* může být úkol, událost, zpráva či poznámka uživatele. Svou podstatou se jedná o obecné procesy během života projektu, ale kvůli jejich specifičnosti je lépe je z pohledu uživatele rozdělit na zmíněné typy procesů.

3.1.1.1 Úkoly

Po přihlášení má každý uživatel možnost *vyzvednout si úkoly*, které mu byly v rámci různých projektů přiděleny. Má dostupný *přehled všech úkolů* s možností sledovat jejich stav. Po vzoru jednotného úkolníku obsahuje přehled všechny úkoly všech projektů a všech uživatelů. Uživatel má možnost přehled úkolů *filtrovat*. Může si tak zobrazit pouze úkoly, které má řešit, které zadal k řešení, které se týkají konkrétního projektu, zákazníka atd. Nově přidělené úkoly v přehledu všech úkolů má vhodně vyznačeny, dokud u nich nepotvrdí jejich *přečtení*. Uživatel si může vybrané záznamy označit *záložkou*, aby měl důležité úkoly vždy na začátku přehledu a tak si je stále připomínal.

Uživatel může zadat *nový úkol*. Zadá potřebné údaje jako zadání, termín, kterého projektu a zákazníka se úkol týká atd. Úkol pak může předat k řešení jednomu konkrétnímu uživateli. Zároveň má možnost přidělit úkolu tzv. pozorovatele, kteří mají právo na čtení úkolu, ale úkol přímo neřeší. Pozorovatelem může být jeden obecný subjekt, tj. uživatel či zákazník nebo také celá uživatelská skupina. Úkol má charakter procesu, takže mu lze přiřazovat vstupy a výstupy. *Vstupem úkolu* mohou být soubory (dokumenty, diagramy, grafika apod.) nebo jiný obecný proces, tj. úkol, událost, zpráva či poznámka uživatele. Na základě události (např. obchodní jednání) může vzniknout úkol a tuto vazbu je dobré zachytit. *Výstupem úkolu* mohou být pouze soubory (dokumentace, schémata apod.). Uživatel může *editovat úkol*, pokud k tomu má dostatečné oprávnění. *Smazat úkol* může pouze uživatel, který úkol zadal.

Každý úkol může být rozdělen na více *podúkolů*. Každý úkol tedy může mít definovánu vazbu na nadřazený úkol. Výkazy by se pak měly provádět přímo na podúkolech. Příkladem může být obecný úkol, který dostane k řešení hlavní programátor. Ten vytvoří jednotlivé podúkoly a předá je k řešení jednotlivým programátorům. Svůj odpracovaný čas může vykazovat do jednoho z podúkolů, který předá k řešení sám sobě. Nadřazený úkol je pak pouze obecným zadáním a případně také úložištěm vstupních dat, ze kterých mohou čerpat podúkoly.

Vzhledem k potřebě pružného řízení projektů by uživatelská oprávnění neměla příliš uživatele omezovat, pokud např. potřebují zadat kolegovi nějaký úkol. Řešitel úkolu by měl mít také možnost rozdělovat práci svým kolegům prostřednictvím podúkolů. Je třeba ošetřit např. vhodnou uživatelskou skupinou, aby běžní uživatelé nemohli úplně všechno, ale také aby nebyli příliš omezováni uživatelskými oprávněními.

3.1.1.2 Události

Události představují pro uživatele možnost zachycení běžných obchodních případů a v případě potřeby jejich dohledání. Událostí je *několik typů*: zápis z jednání, obchodní schůzka, telefonický kontakt, e-mail došlý, e-mail odeslaný, vědomost, hesla, obecná událost. Pomocí události si může uživatel naplánovat obchodní schůzku se zákazníkem a přímo na jednání pořídit zápis, který předá ke čtení vývojářskému týmu. Události slouží i celé firmě pro ukládání různých vědomostí, důležitých hesel, důležité komunikace atd.

Každý uživatel má k dispozici *přehled událostí*. Díky tomu má dostupné informace o dění ve firmě a v jednotlivých projektech. U některých událostí je důležité, aby si jej všichni cíloví uživatelé přečetli. Událost lze předat ke čtení podobně jako úkol osobě nebo uživatelské skupině. Aby nedocházelo ke čtení rozepsaných událostí, je třeba zobrazovat nové události až po zveřejnění ke čtení. Výpis událostí by měl mít možnost vhodného *filtrování*. Uživatel si tak může snadno vyhledat všechny zápisy z jednání s konkrétním zákazníkem, proběhlou e-mailovou komunikaci apod. Tato možnost je důležitá i pro případ, že by došlo k nesouladu mezi původním požadavkem zákazníka a jeho současným tvrzením k některému bodu projektu.

Ne všechny události mohou být veřejné. Proto by měl uživatel při zadání *nové události* určit, kdo má událost právo číst, případně označit událost jako soukromou apod. *Smazat událost* může pouze uživatel, který událost založil. Je třeba navrhnout vhodný způsob řešení *editace událostí*. Pokud si již někteří uživatelé událost přečetli, nemělo by být možné událost měnit, nebo by si měli uživatelé událost přečíst znova. K události lze přikládat různé soubory jako *přílohy* ke stažení. Každou událost uživatel může označit jako *přečtenou* a již ji nevěnovat pozornost. Má však i možnost událost označit *záložkou* a pak ji bude mít v přehledu událostí stále na prvním místě, dokud záložku nezruší.

3.1.1.3 Zprávy

Zprávy slouží pro vnitropodnikovou komunikaci uživatelů. Každý uživatel má k dispozici *přehled zpráv*, které mu byly zaslány a které sám zaslal jinému uživateli. Zprávy, které příjemce dosud nepřečetl, jsou vhodně zvýrazněny. Nové příchozí zprávy má uživatel ještě více zvýrazněny a zobrazují se v přehledu vždy na začátku.

Příchozí zprávu má uživatel možnost označit jako *přečtenou*, takže na ni již nebude upozorňován a příjemce uvidí, že jeho odeslaná zpráva byla úspěšně doručena a přečtena. Na příchozí zprávu lze také jednoduše *odpovědět* odesílateli. Každou zprávu je možné označit *záložkou* podobně jako úkol nebo událost. Ke zprávám je možné přikládat různé soubory jako *přílohy* ke stažení.

Při psaní *nové zprávy* lze nastavit, zda se týká určitého projektu, konkrétního zákazníka či uživatele. V přehledu zpráv lze díky tomu *filtrovat* zprávy podle hledaného tématu či vazby na určitý projekt nebo osobu. *Editovat odchozí zprávu* lze pouze do té doby, než bude příjemcem označena jako přečtená.

3.1.1.4 Poznámky

Poznámky plní funkci oblíbených žlutých nalepovacích papírků, ale v modernější elektronické podobě. Poznámky jsou pouze osobní, takže každý uživatel v *přehledu poznámek* vidí jen své uložené poznámky. Co se týče dostupných funkcí v rámci poznámek, postačí velmi jednoduché možnosti.

Uživatel může napsat *novou poznámku*, kde vyplní text poznámky a případně ještě termín či subjekt, kterého se poznámka týká. Příkladem může být „Text: Zavolat dopoledne dodavateli, jestli je

schopný dodat UPS do druhého dne; Týká se: Zákazník X; Termín: 30.1.2008“. Poznámky lze také *editovat* a *mazat*. Uživatelé nemají přístup k poznámkám ostatních uživatelů.

3.1.1.5 Adresář

Každý uživatel má přístup do *adresáře subjektů*. Ten nabízí přehled všech aktivních subjektů uložených v systému. Subjekty mohou být různého typu: uživatel, zákazník, dodavatel, partner apod. Pro základní funkčnost postačí subjekty typu uživatel a zákazník. Uživatel si může vyhledat potřebný kontakt na zákazníka, konkrétní pracovníky apod. K tomu slouží možnost *filtrování* výpisu subjektů.

Někteří uživatelé by měli mít možnost přidat *nový subjekt* typu zákazník. Pokud se vytváří projekt pro nového zákazníka, nebude tento zákazník zatím v systému uložen. Přidávat uživatele by měl ale pouze administrátor. Adresář slouží primárně spíše k prohlížení aktuálních záznamů, je tedy ponecháno na zvážení, jaké možnosti správy bude adresář nabízet a co bude přenecháno do části správy a nastavení systému.

3.1.1.6 Projekty

Uživatelé si mohou prohlížet *přehled všech projektů* a jejich *detail*. Mají tak přehled, co všechno firma momentálně realizuje. Výpis je možné *filtrovat* a vyhledávat tak konkrétní projekty.

Systém by měl umožňovat *správu projektů*. Ta zahrnuje možnost vytvářet *nové projekty*, editovat stávající projekty, ukončovat projekty nebo je celé mazat, resp. nastavit jim příslušný stav. K těmto funkcím by neměli mít přístup všichni uživatelé. Je potřeba navrhnout vhodný způsob bezpečnostní politiky nejen pro správu projektů. Při návrhu je potřeba zohlednit i potřebnost zastupitelnosti uživatelů v případě nemoci apod. V praxi často nastává případ, kdy uživatel potřebuje zadat nový úkol, ale projekt, resp. zakázka, ještě není v systému uložena. Měl by mít tedy možnost tento případ nějakým způsobem řešit.

3.1.1.7 Kalendář

Kalendář představuje pro každého uživatele možnost zobrazit si rychle kalendář a plánovat si např. schůzky apod. Cílovým stavem kalendáře je zachytit ke každému dni přehledně všechny uživatelovy schůzky, úkoly, výročí atd. Kalendář by tedy shrnoval některá data z uložených procesů vztahujících s k přihlášenému uživateli.

3.1.1.8 Správa a nastavení

Administrátor má přístup ke správě uživatelů a uživatelských skupin. Uživatelské skupiny mají různá oprávnění, která jsou nadřazená oprávněním vyplývajícím z role uživatele na daném projektu. Uživatele lze přiřazovat do různých rolí jako např. vedoucí projektu, analytik, návrhář apod. Je ponecháno na zadavateli, zda se pro každý nový projekt bude sestavovat samostatný tým nebo zda bude složení týmu jasné z přiřazení do uživatelské skupiny. Administrátor bude mít přístup i ke správě dalších obvyklých číselníků jako např. číselník zákazníků, států, používaných měn apod.

3.1.1.9 Sledování projektů a výkazy

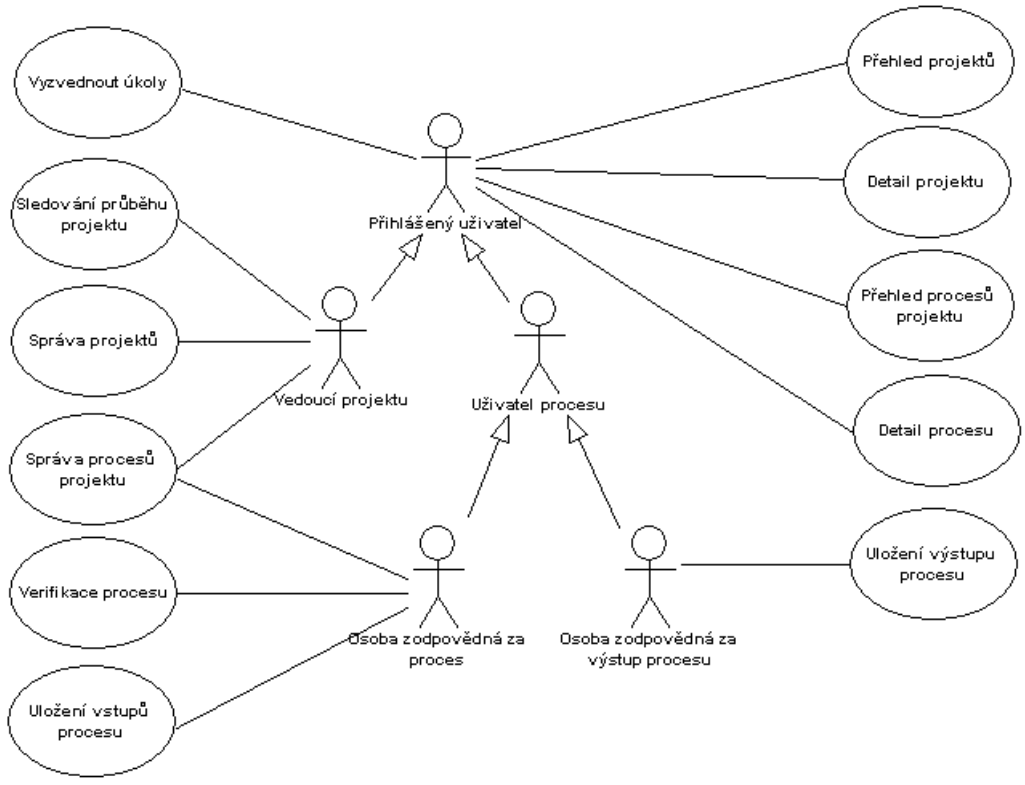
Pro účely projektového řízení a plánování by měl systém umožnit sledovat průběh jednotlivých úkolů a celých projektů. Sledování by měl mít většinou na starosti vedoucí projektu. Stejně tak plánování a řízení projektu. Tyto činnosti by měl systém podporovat a zefektivňovat. Uživatel, který zadal kolegovi úkol, by měl mít také možnost sledovat, jestli jej kolega převzal a splnil.

Systém by měl rovněž umožnit uživatelům provádět výkazy práce. Každý řešitel úkolu by měl odhadovat dobu řešení a tu skutečnou pečlivě vykazovat. Výkazy objemu prací jsou další z důležitých vlastností, které by měl systém umožňovat sledovat.

3.2 Diagram případů užití

Diagram případů užití je na obrázku 15. Vychází z neformální specifikace požadavků a představuje chování systému z pohledu uživatelů. Různá uživatelská oprávnění lze aplikovat pomocí uživatelských skupin.

Diagramy reprezentují systém na nízké úrovni podrobností, aby byl přehledný jak pro zadavatele, tak i pro vývojáře. Zachycují tedy základní možnosti aktivit, které mají uživatelé systému k dispozici. Ve fázi návrhu by mohly sloužit pro zachycení dynamické struktury systému. Pro detailnější návrh by ale musely být rozpracovány do vyšší úrovně podrobností.



Obr. 15: Use Case Diagram – chování systému z pohledu uživatelů

4 Návrh

Fáze návrhu vychází z předchozí fáze analýzy. Formálně definuje, jak vyvíjený systém vytvořit. Návrh by měl obsahovat zejména dynamický a statický model systému, návrh uživatelského prostředí a návrh databáze. Nejběžnějším způsobem je pochopitelně modelování pomocí nejrůznějších diagramů UML.

Snažil jsem se navrhnout systém v co největším možném rozsahu bez ohledu na to, jestli se budou všechny navržené části implementovat. To by mělo zaručit dobré možnosti pro další rozvoj systému v budoucnu. Z existující dokumentace návrhu lze vycházet při realizaci nových programových přírůstků.

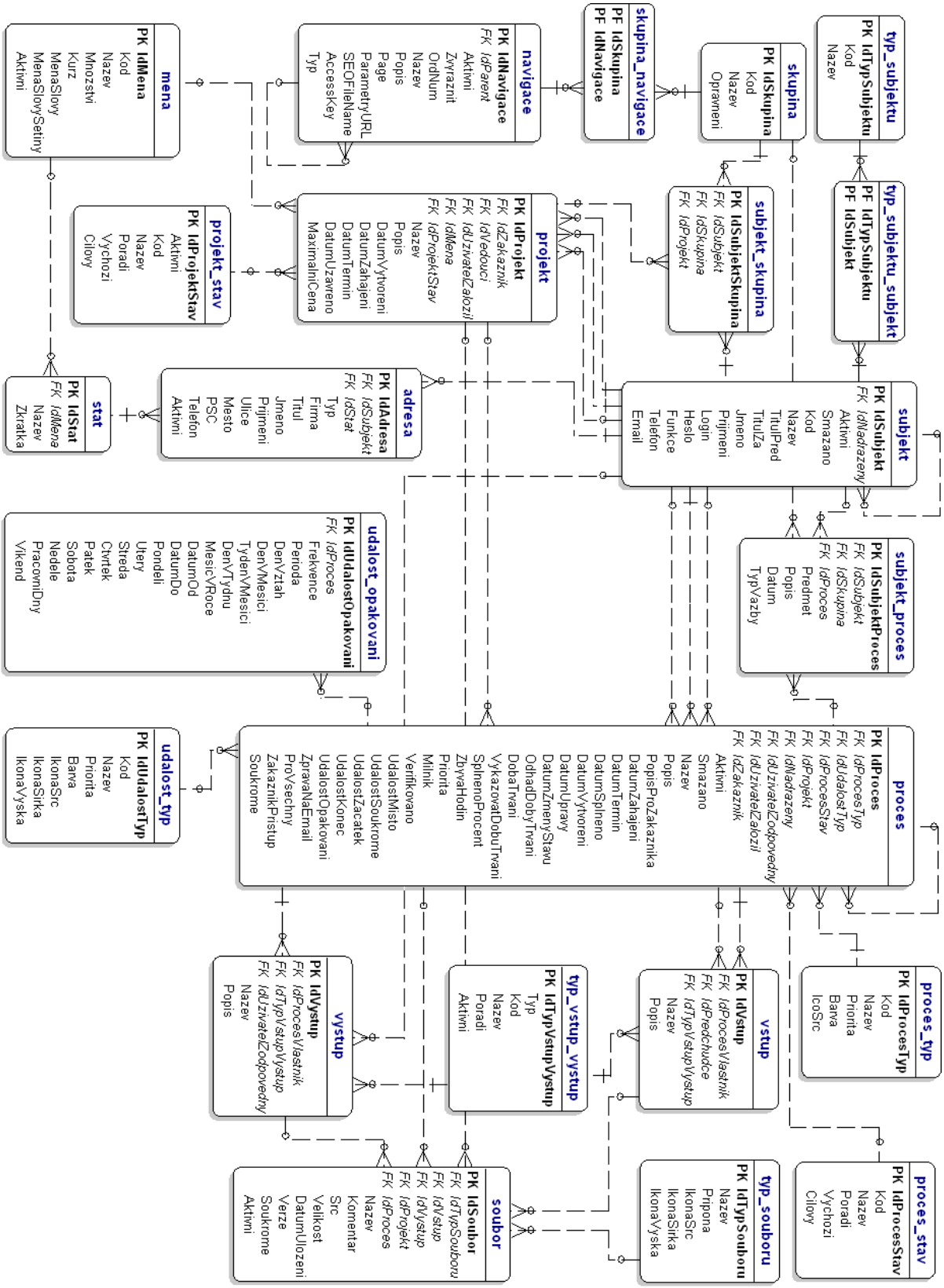
Podobně jako fáze analýzy byla komplikována častými změnami požadavků, byla i fáze návrhu zatížena neefektivním a častým upravováním datového modelu. Mnoho změn bylo nutné provádět během seznamování se s datovým modelem informačního systému a jeho možnostmi. Z důvodu potřebné kompatibility jsem byl nucen vytvořit co možná nejuniverzálnější návrh datového modelu. Časté doplňování a úpravy modelu byly také zapříčiněny nedostatečně detailní specifikací požadavků. Dokonce i během implementace jsem se musel vrátit na úroveň datového modelu a upravovat jej. Takové vracení se k předchozím etapám není vhodné pro efektivní vývoj systému. Vede jen k dalším a dalším změnám, frustruje vývojáře a způsobuje problémy v projektovém řízení.

Pro potřeby této diplomové práce jsem do fáze návrhu zahrnul návrh datového modelu a návrh uživatelského prostředí (GUI). Pro návrh dynamické struktury využívám diagramy případů použití realizované ve fázi analýzy.

4.1 Návrh databáze

Schéma datového modelu je na obrázku 16. Diagram je znázorněn ve formě, která by se dala označit za E-R diagramu (Entity-Relationship Diagram). Schéma jsem namodeloval v programu DeZign for Databases v4.2.0. Rozdíly od běžněji užívaného E-R diagramu mohou být v odlišném značení kardinality vztahů a absenci textového popisu u vztahů. Rovněž vztahy M:N se zde modelují rovnou další entitou, což může výsledné schéma činit méně přehledné. Velká výhoda zmíněného programu spočívá v možnosti vygenerovat databázový skript, který podle schématu vytvoří databázi pro cílový DBMS (DataBase Management System).

Během implementace systému jsem potřeboval do některých tabulek doplnit několik sloupců. Většinou se jednalo o sloupec, který označoval záznam jako aktivní. Do schématu datového modelu jsem již tyto úpravy nepromítal. Vygenerovaný skript pro vytvoření databáze, včetně později doplněných sloupců, je dostupný v adresáři aplikace „/db/create.sql“. Více informací o použití tohoto skriptu uvádím v kapitole o nasazení a instalaci aplikace.

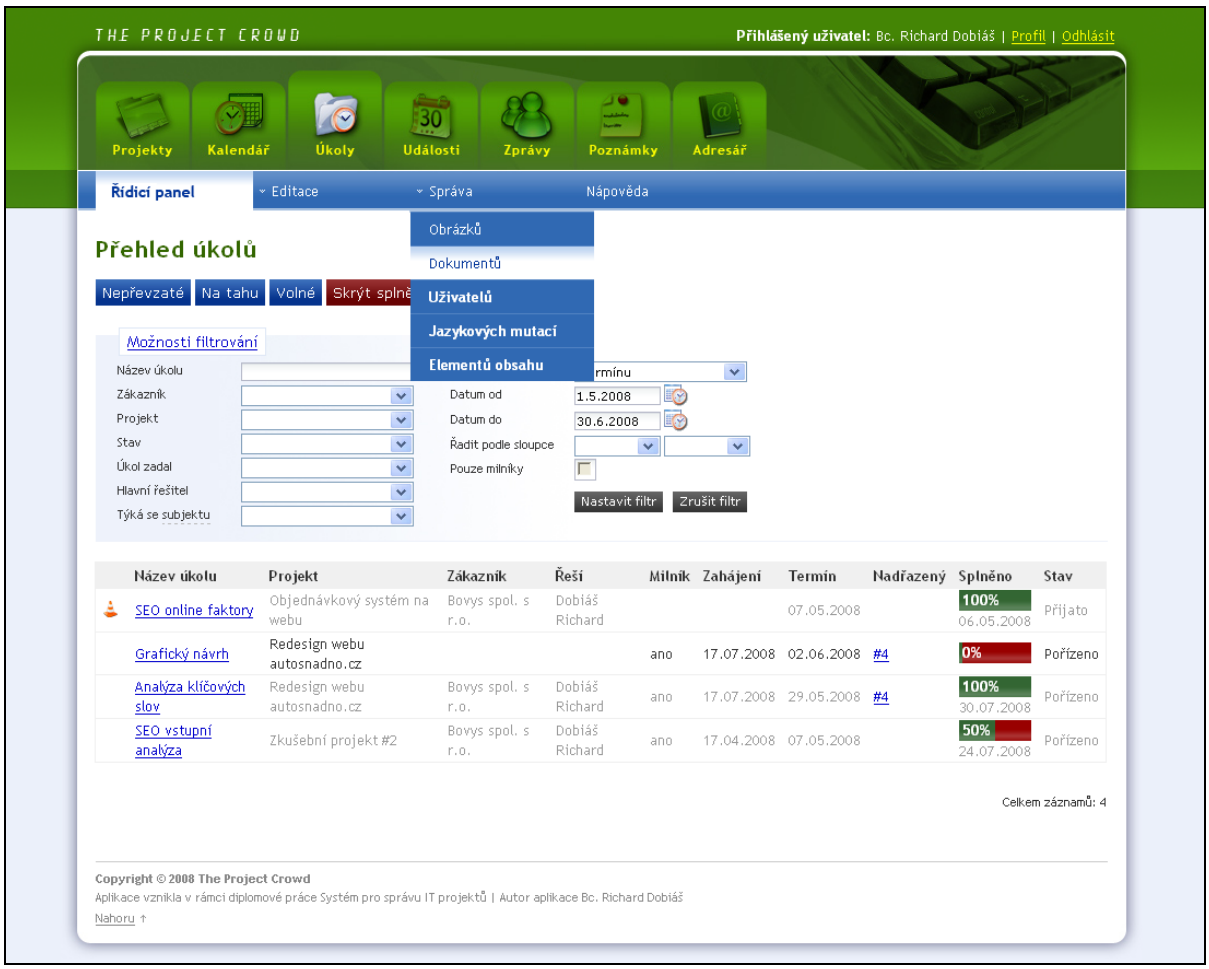


Obr. 16: Návrh datového modelu

4.2 Grafické uživatelské rozhraní

Návrh uživatelského rozhraní definuje prostředí, které bude uživateli sloužit k ovládání celého systému. Definuje menu a různé formuláře, které bude uživatel používat k požadovaným činnostem. Velmi užitečný může být programátorům návrh sledů obrazovek, ze kterého je velmi přehledným způsobem jasné, co se bude uživateli v jeho současném stavu zobrazovat a jaké má další možnosti.

Vzhledem ke skutečnosti, že vyvíjený systém bude implementován formou webové aplikace, zahrnul jsem do této fáze návrh designu jednotlivých www stránek, globální navigace a běžných formulářů. Výsledný návrh je na obrázku 17. V následujícím textu stručně uvedu některé důležité prvky uživatelského rozhraní s odkazem na zmíněný návrh vzhledu aplikace.



Obr. 17: Návrh designu vyvíjené aplikace

Uživatelské rozhraní bude v hlavičce webu obsahovat globální navigaci společnou pro všechny uživatele. Pro větší přehlednost a příjemnější pocit uživatele jsou položky navigace navrženy jako grafická tlačítka s tematickou ikonou. Pro dynamický efekt se bude tlačítko při přejetí myši zvýrazňovat. Stejné zvýraznění bude mít tlačítko aktuálně zobrazené stránky.

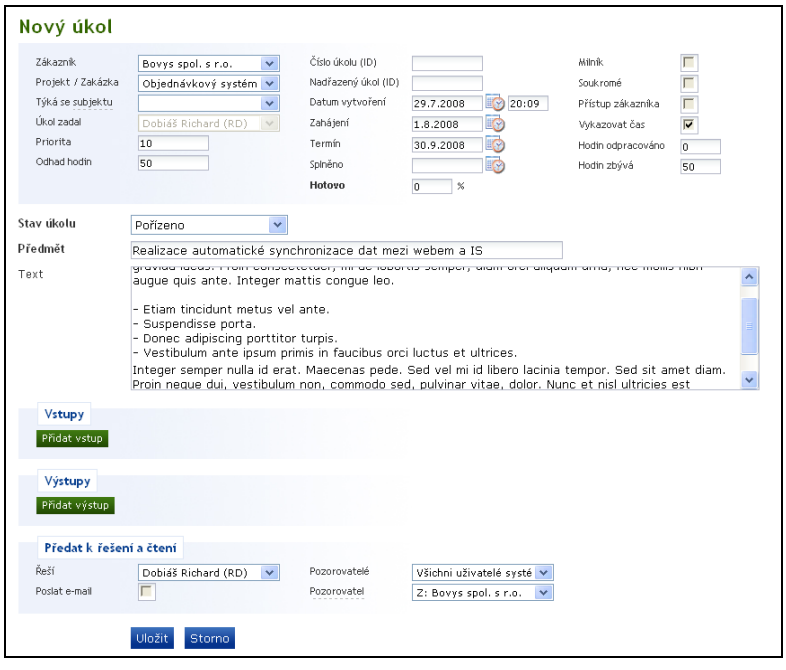
Pod globální navigací bude umístěno rozbalovací menu. To rozšiřuje základní navigaci a zpřístupňuje uživateli některé funkce pro nastavení a správu systému. Menu bude vygenerováno pro každého uživatele individuálně podle toho, k jakým funkcím má dostatečné oprávnění.

Pod hlavičkou webu, která obsahuje obě zmíněné navigace, je vždy zobrazen nadpis aktuálně zobrazené stránky. Uživatel tak má přehled, ve které části aplikace se momentálně nachází. Pod nadpisem stránky jsou obvykle dostupná funkční tlačítka pro ovládání potřebných akcí. Tlačítka pro přidávání záznamů či zapínání funkčností jsou obarveny do zelené barvy. Tlačítka pro smazání či vypínání funkčnosti jsou obarveny do červené. Tlačítka pro nastavení filtru jsou šedá. Ostatní tlačítka jsou nastylovány do modré barvy. Uživatel tak snadno odliší povahu tlačítek.

Výpisy záznamů z databáze budou v případě tabulkových dat zobrazeny přehlednou tabulkou. Důležité řádky tabulky budou vhodným způsobem zvýrazněny, např. podbarveny tematickou barvou podle typu záznamu. Např. nepřečtené události budou podbarveny světlým odstínem červené barvy. K tomuto způsobu označení je vhodné ještě doplnit jasnou informaci o tom, proč je řádek tímto způsobem zvýrazněn. První sloupec tabulky může sloužit pro zobrazení různých ikon, které budou znázorňovat např. typ události nebo aktivní záložku.

Nad výpisem záznamů bude formulář umožňující filtrování výpisu. Po kliknutí na nadpis filtru se celý filtr schová, resp. zobrazí. Schování filtru musí mít za následek vypnutí filtrování, jinak by mohlo docházet k matení uživatele v případě zapomenutého aktivního filtru apod.

Uživatelské rozhraní pro přidání nebo zobrazení, resp. editaci záznamu, bude velmi podobné obrazovce s výpisem záznamů. Místo tabulky záznamů budou na stránce formulářová pole. Celkový vzhled všech karet a výpisů záznamů je podobný, aby si uživatel zvykl na jednotný koncept a snadno se v něm orientoval. Ukázka karty úkolu je na obrázku 18.



Obr. 18: Návrh rozhraní pro kartu úkolu

5 Implementace

Fáze implementace vychází z etapy návrhu systému a zabývá se samotným vývojem aplikace, tedy jejím programováním. Cílem implementace je vytvoření funkčního kódu systému. Implementace systému by měla vždy probíhat podle detailního plánu implementace [30]. Tento plán specifikuje implementační strategii, rozvrh programování přírůstků, jejich testování, integraci a také integrační testování. Detailní plán je samozřejmě možné vytvořit až po návrhu systému, kdy je pomocí typů jednání (případů užití) jasně dáno, jaké funkčnosti budou realizovány. Důležité je určení pořadí přírůstků, které probíhá na základě ohodnocení hodnotami vývojové riziko a zákazníkova priorita. Typy jednání s vysokým rizikem a malou prioritou se realizují jako první. Typy s vysokým rizikem i prioritou budou mít pravděpodobně vysoké nároky na zdroje a tak vyžadují velkou pozornost. Pomocí ohodnocení vznikne pořadí přírůstků, a tak je možné vytvořit detailní plán implementace.

V dalších podkapitolách se věnuji popisu použitých technologií při realizaci výsledné aplikace. Dále některým zajímavým vlastnostem systému a také problémům, na které jsem během implementace narazil. Jak jsem již dříve zmiňoval, výsledná aplikace má být schopná reálného nasazení u společnosti KomTeSa, která moji práci konzultovala. Z tohoto důvodu jsem byl směřován do jistého způsobu vývoje, který je v této firmě zavedený.

Aplikace je implementována jako internetová, resp. intranetová. Tento způsob je velmi výhodný. Pro správný chod systému postačuje uživatelům obyčejný internetový prohlížeč a připojení k serveru, na kterém je systém provozován. Všichni uživatelé přistupují ke stejným datům uloženým ve společné databázi. Jakákoliv údržba aplikace se provádí pouze na serveru, bez nutnosti aktualizace jakéhokoliv programu na uživatelských stanicích. Prostřednictvím sítě Internet lze k aplikaci přistupovat odkudkoliv z celého světa a z jakéhokoliv počítače, který má alespoň běžný webový prohlížeč. Konkrétní techniky použité při implementaci uvedu v příslušných podkapitolách.

5.1 Programování na straně serveru

Cílová aplikace má být funkční jako intranetová, resp. internetová aplikace. K tomu je potřeba, aby systém běžel na webovém serveru a byl přístupný z klientských stanic přes internetový prohlížeč. Vývoj webových aplikací se stává stále populárnějším. Zlepšují se možnosti současných technologií a přichází na řadu technologie nové.

Vývoj webových aplikací v KomTeSe probíhá především ve skriptovacím jazyku PHP s využitím vlastního jednoduchého frameworku, šablonovacího systému Smarty a databáze MySQL. Všechny tyto technologie jsou dostupné jako Open Source a díky tomu jsou velmi známé a používané po celém světě.

5.1.1 Jazyk PHP

PHP je velmi populární skriptovací jazyk, který se zpracovává na straně serveru. Výstupem bývá většinou webová stránka, kterou server pošle klientovi do prohlížeče. Jednoduchost jazyka PHP zapříčinila masové rozšíření mezi vývojáře a dnes je velmi oblíbenou technologií pro vývoj webových aplikací. PHP již od verze 3 podporuje objektově orientované programování. PHP5 však tuto podporu kompletně přepracovala a přinesla plnohodnotný objektově orientovaný vývoj aplikací v jazyce PHP. Vzhledem k mým zkušenostem s vývojem webových aplikací v PHP jsem uvítal možnost realizace projektu v tomto jazyce.

5.1.1.1 PHP framework

Pokud má být vývoj aplikací efektivní a výsledný kód udržovatelný, je dobré používat vhodný framework. Ten ujednotí způsob programování, zpřehlední zdrojový kód a pomůže při týmové spolupráci. Pro PHP existuje spousta frameworků. Od těch nejpropracovanějších až po ty nejjednodušší. K těm známějším a propracovanějším patří např. Zend Framework.

Přestože má použití frameworku spoustu výhod, najde se pár nevýhod, kvůli kterým jsem se dosud jejich nasazení vyhýbal. Předně je třeba říci, že nasazením frameworku se zvýší spotřeba výpočetního výkonu aplikace. V případě velmi robustních frameworků mohou být dokonce výsledné aplikace celkově dost pomalé. Další nevýhodou je nutnost vybrat ten nevhodnější framework pro určitý typ projektů a naučit se jej efektivně používat. V mém případě v tomto vidím asi ten největší problém. Každý PHP programátor si oblíbí nějaký framework a považuje jej za ten nejlepší.

Spousta PHP frameworků, se kterými jsem se setkal, využívala plně architekturu MVC (Model-view-controller). To je samozřejmě vhodné, neboť se oddělí vrstvy prezentace, logiky aplikace a dat. Pro část View, tedy vrstvu prezentace a interakce aplikace s uživatelem, slouží HTML stránka. Část Model zapouzdřuje věcnou a datovou logiku. Obsahuje tedy velké množství programových komponent. Část Controller je mezivrstvou mezi View a Model. Zajišťuje zpracování událostí v prezentační vrstvě a spuštění příslušné funkčnosti z vrstvy Model. Příkladem frameworku využívající výhody MVC může být již zmíněný Zend Framework.

Existují frameworky s poněkud odlišným přístupem k vývoji internetových aplikací. Např. framework PRADO (PHP Rapid Application Development Object-oriented) je komponentový, událostmi řízený framework pro vývoj webových aplikací v PHP5. Svým konceptem jde dále než jen k oddělení vrstev dle MVC. Vytváří strukturované aplikace a používá událostmi řízené programování. Vývoj s PRADO je možné přirovnat k vývoji v ASP.NET. Pokud ale programátor nebude při vývoji opatrný, může být výsledná aplikace pomalá nebo může dojít k překročení dostupné paměti [28].

KomTeSa používá pro vývoj webových aplikací vlastní PHP framework. Je velice jednoduchý, ale jeho použití usnadňuje práci a zdrojové kódy aplikace jsou velmi přehledné. Zřejmě i díky využití objektově orientovaného programování. Framework tvoří několik tříd, které si mezi sebe rozdělují

potřebnou funkčnost vyvíjené aplikace. Nechybí univerzální třída pro práci s databází, často používané funkce pro ověření přihlášení či správu uživatelů systému. Díky použití šablonovacího systému Smarty je docíleno oddělení vrstev Model, View a Controller. Controller zastupuje soubor index.php, na který se směřují požadavky z prezentační vrstvy. Podle požadované akce se zavolá některá z metod globálního objektu určeného pro zpracování logiky aplikace. Vrstva Model tedy zpracuje požadavek, vytáhne potřebná data z databáze a předá je do vrstvy View. Tu představují šablony Smarty, které si data převezmou a vytvoří výstupní HTML stránku.

5.1.2 Šablonovací systém Smarty

Systémy pro práci se šablonami se s oblibou používají k oddělení aplikační a prezentační logiky. Oddělí se tedy jejich přílišná závislost. To umožňuje lepší spolupráci členů týmu na jednom projektu. Programátor aplikační logiky se soustředí pouze na kvalitní kód tříd a efektivní práci s databází. Připraví datové struktury, které potřebuje výstupní stránka. Kodér si pak prostřednictvím šablony data zpracuje a naformátuje na výstup v potřebné podobě. Výhodou je i možnost použití různých šablon na stejné datové struktury bez nutnosti zásahu do aplikační logiky.

Smarty je zřejmě nejrozšířenějším a nejpopulárnějším šablonovacím systémem pro PHP. Smarty je jednoduché a velmi rychlé. Při prvním spuštění skriptu dojde ke kompilaci a další spuštění je již velmi svižné díky nastavitelným možnostem kešování. Pokud kodér změní šablonu, dojde automaticky znovu ke kompilaci. Syntaxe Smarty je velmi jednoduchá. Použití šablon a vývoj aplikací se Smarty se dá ovládnout během několika hodin. Šablona se převážně skládá z HTML kódu a v potřebných místech se vkládají speciální značky Smarty. Na obrázku 19 je část kódu šablony, kterou jsem v aplikaci použil pro zobrazení konkrétní zprávy uživateli. Značky Smarty jsou od HTML odděleny složenými závorkami.

```
<div id="sysmessage" class="{if $err_level eq 1}app-message
                            {elseif $err_level eq 2}app-warning
                            {elseif $err_level eq 3}app-error{/if}">

    <a id="sysmessage-close" href="#">Zavřít<span></span></a>
    <div class="title">{if $err_level eq 1}Zpráva systému
                       {elseif $err_level eq 2}Varování
                       {elseif $err_level eq 3}Chyba{/if}</div>
    <div class="message">{$message}
    {if $err_no ne ""}
        <div class="code">Chybový kód: <br />{$err_no}</div>
    {/if}
    </div>

</div>
```

Obr. 19: Šablona Smarty kombinuje HTML a speciální značky

5.1.3 Databázový systém MySQL

MySQL je relační databázový systém. Svou obrovskou oblibu si získal zejména ve spojení s PHP pro vývoj webových aplikací. Mezi výhody patří např. rychlost, výkonnost, jednoduchost a dostupnost Open Source licence. Používá standardní dotazovací jazyk SQL. Bohužel nemá některé užitečné vlastnosti jako jeho větší kolegové typu MS-SQL či Oracle. Přestože poslední verze MySQL 5.0 přinesla spoustu rozšíření a vylepšení, dosud MySQL nepodporuje např. možnost vnořených SQL dotazů. V MySQL si lze vybrat pro každou databázi jeden z několika způsobů uložení dat. Práce s tabulkami typu MyISAM je rychlejší a umožňuje Full-text indexování a vyhledávání. Podporu referenční integrity však poskytuje MySQL pouze v případě použití tabulek typu InnoDB. Ty naopak neumožňují Full-text. Je tedy důležité zvolit výhodnější a optimálnější způsob podle potřeb konkrétního projektu.

Pro svou aplikaci jsem zvolil InnoDB tabulky. Díky tomu odpadá nutnost řešit referenční integritu na aplikační úrovni. Kvalitní návrh datového modelu zaručí, že vymazání či editace záznamu se automaticky promítne do propojených tabulek.

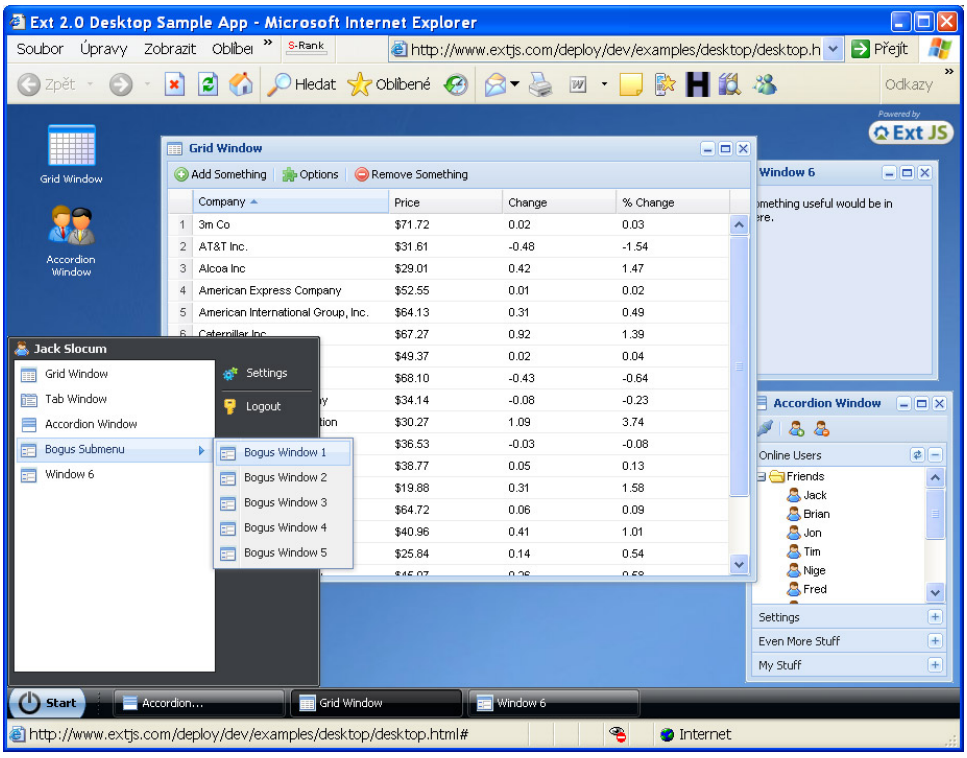
Po diskuzi s konzultantem projektu jsem začal zvažovat, jestli by nebylo lepší zvolit engine MyISAM. V aplikacích tohoto typu se totiž příliš záznamy nemažou. Naopak může mít smazání nepříjemné důsledky. Stane se i situace, kdy zaměstnanec omylem vymaže zakázku. Referenční integrita v tu chvíli vykoná své. Všechny úkoly, události, výkazy práce či další data ohledně této zakázky jsou nenávratně pryč. V tuto chvíli přichází na řadu obnovení zálohy, pokud tedy nějaká existuje. Jak řešit takové případy v aplikaci? Buď nedovolit mazání, ale pouze deaktivaci záznamu např. nastavením příznaku ve speciálním sloupci, nebo záznam před trvalým odstraněním zazálohovat včetně všech záznamů z propojených tabulek. Vzhledem k tomu, že v aplikaci používám pro ukládání souborů filesystem operačního systému a nikoliv možnosti databáze pro uložení souborových dat, potřebuji mít stále záznam o souboru na disku v databázi. Při trvalém mazání záznamů z databáze by se musely také mazat fyzické soubory z disku, jinak by se z úložiště dat stával odpadkový koš. Samotná referenční integrita mazání souborů z disku nezajistí.

Po dlouhých úvahách jsem se rozhodl k takovému řešení, kdy stisk uživatele tlačítka „smazat“ neprovede trvalé odstranění záznamu, ale pouze jej zneaktivní. Uživateli se však celá akce jeví jako skutečné smazání. V aplikaci jsem však realizoval odstranění záznamu jak trvalým odstraněním, tak i zneaktivněním. V konfiguračním souboru aplikace se pomocí konstanty nastavuje, který způsob má aplikace používat. Kdyby se tedy v budoucnu mazání řešilo druhým způsobem, stačí změnit jednu konstantu.

5.2 Programování na straně klienta

Vývoj webových aplikací je samozřejmě neodmyslitelně spojen s jazyky, které se spouštějí na straně klienta. Webová aplikace se z pohledu uživatele chová stejně jako běžné www stránky, na které jsou dnešní uživatelé dostatečně zvyklí. Pro správný chod aplikace postačuje pouze internetový prohlížeč s případnými rozšířeními.

Vývoj nelze zastavit a tak běžné statické www stránky jsou nyní nahrazovány dynamickými weby, které uživateli přináší nadstandardní komfort. Ať už se jedná o líbivá grafická uživatelská rozhraní, inteligentní kontroly formulářů bez nutnosti odezvy serveru či zvýšení rychlosti načítání stránek díky Ajax technologii. Za zmínku určitě stojí také stále využívanější technologie FLASH či ještě lépe nastupující Adobe AIR, Adobe Flex a Microsoft Silverlight, které přináší takovou malou revoluci do tvorby webových řešení. Díky těmto novým technologiím se do popředí zájmu dostávají tzv. RIA (Rich Internet Application) aplikace. Ty se snaží o to, aby se webové aplikace podobaly co nejvíce těm desktopovým, ať už na úrovni vzhledu, chování nebo i vyššího uživatelského komfortu [16]. Budoucnost je těmto novým řešením nakloněna. Lze tedy očekávat stále interaktivnější a zajímavější aplikace, které v mnohém předčí ty desktopové. Navíc budou dostupné z celého světa přes obyčejný internetový prohlížeč. Zajímavé srovnání těchto technologií popisuje např. [17]. Před nástupem těchto technologií přicházelo v úvahu pro zvýšení uživatelského komfortu pouze použití skriptovacího jazyka JavaScript či jeho novější způsob využití ve formě Ajax.



Obr. 20: Ukázka RIA aplikace vytvořené pomocí JavaScript frameworku ExtJS [24]

Jako zastánce dobré přístupnosti webu nejsem zatím příliš nakloněn výše zmíněným technologiím jako je Flash apod. Přesto, že budoucnost bude patřit zřejmě právě jim, pro realizaci klientské části aplikace jsem použil osvědčené technologie, které se ještě stále pro tvorbu intranetových či internetových aplikací běžně používají. Jsou bezbariérové, relativně rychlé, jednoduše se implementují a udržují.

5.2.1 Značkovací jazyk (X)HTML

XHTML (eXtensible HyperText Markup Language) je značkovací jazyk určený pro tvorbu hypertextových dokumentů, které se používají v prostředí WWW (World Wide Web). XHTML nahrazuje původní HTML jazyk, jehož vývoj již byl ukončen verzí 4.01. Pro zdůraznění významu XHTML stručně zmíním historii vývoje značkovacích jazyků pro tvorbu WWW stránek [18, 19].

Metajazykem pro deklaraci různých typů dokumentů se stal SGML (Standard Generalized Markup Language), který je mezinárodním standardem pro popis značkovaného textu [20]. Nejznámější aplikací je právě již zmíněný jazyk HTML. Po úpravě SGML vznikl pro dnešní dobu velmi populární metajazyk XML, který je vhodný pro deklaraci různých typů strukturovaných dat. Díky jeho univerzálnosti se často využívá pro komunikaci a přenos dat mezi různými systémy. Aplikací jazyka XML, který má velmi podobnou sémantikou jako jazyk HTML, vznikl jazyk XHTML. Dá se tedy říci, že jakýkoliv XHTML dokument je současně validním XML dokumentem.

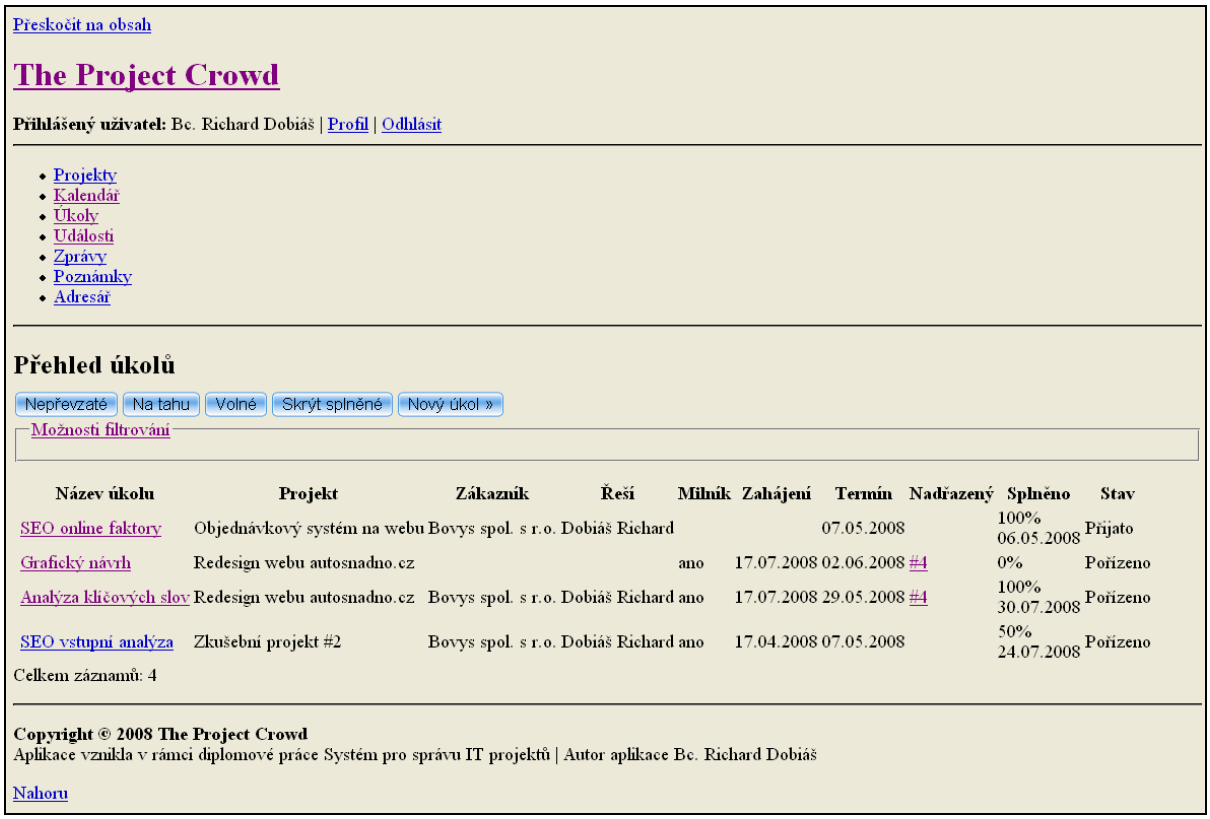
Historie vytváření webových stránek začíná jazykem HTML 2.0, který vydala standardizační společnost IETF (Internet Engineering Task Force). Tato verze jazyka HTML byla velmi strohá. Obsahovala prvky pro definici struktury dokumentu, ale neumožňovala žádné, lidskému oku lahodící, grafické ztvárnění stránky. Z tohoto důvodu přišla verze HTML 3.2, která byla obohacena o prvky sloužící pouze k definici vzhledu stránky. Od této verze se ujímá vývoje jazyka HTML konsorcium W3C (World Wide Web Consortium). Pro dobrou přístupnost dokumentů z různých zařízení však HTML prvky pro definici vzhledu nejsou vhodné. Vydání další verze HMTL 4.0 se soustředilo na maximální možnosti definice struktury dokumentu. Formátování vhledu přechází na nový jazyk pro popis způsobu zobrazení webových stránek, tzv. CSS (Cascading Style Sheets). Později vychází revidovaná verze HTML 4.01, která vývoj jazyka HTML ukončuje a stává se výchozím bodem dalších jazyků pro tvorbu webu, které na HTML navazují.

Snahou W3C bylo prosadit univerzálnější metajazyk XML jako hlavní značkovací jazyk nejen pro web. Výsledkem byl vznik jazyka XHTML, který svojí specifikací odpovídá jazyku HTML 4.01, ale integruje pravidla jazyka XML. Oproti HTML je ořezán a soustředí se pouze na definici struktury dokumentu. Díky tomu by měl být nezávislý na zobrazovacím zařízení, na kterém je webová stránka zobrazena. Rozšíření webu na velké množství alternativních zařízení bohužel ukázalo, že některá zařízení nedokáží podporovat všechny vlastnosti jazyka XHTML. Proto vznikly jakési podmnožiny, které bylo třeba definovat a standardizovat. Specifikace *Modularization of XHTML* rozděluje prvky

do modulů, ze kterých se pak skládají značkovací jazyky. Pro specifická zařízení lze tedy vytvářet nové moduly, upravovat stávající, nebo skládáním modulů vytvářet celé značkovací jazyky. Verze XHTML 1.1 je již definována pomocí modulů. Starší verze XHTML 1.0 moduly ještě neobsahuje, ale od verze 1.1 se příliš neliší. Více o tomto tématu lze nalézt např. v [19].

Pro realizaci aplikace jsem využil jazyk *XHTML 1.0 Strict*. V dnešní době je to stále nejpoužívanější značkovací jazyk pro tvorbu WWW stránek. Při psaní kódu jsem pečlivě dbal na čistou a dobře postavenou strukturu dokumentu a samozřejmě i validní kód. To zaručí, že žádný běžně používaný prohlížeč nebude mít problémy se správným zobrazením stránek. Využil jsem moderní přístup, kdy se pomocí XHTML definuje pouze struktura dokumentu, aniž by se uvažovalo jakékoliv rozmístění jednotlivých prvků na stránce. Formátování a rozmístění prvků je realizováno pomocí jazyka CSS, o kterém se zmíním v další podkapitole. Použití tabulek je omezeno pouze pro zobrazení výpisu tabulkových dat. Rovněž rozmístění formulářových prvků není řešeno pomocí tabulek. Důvody, proč se vyhýbám nevhodnému použití tabulek, vysvětlím v podkapitole věnované použitelnosti a přístupnosti webu.

Zdrojový kód, který dorazí k uživateli do prohlížeče, je velmi úsporný a obsahuje pouze nezbytné prvky. Šablona, kterou se nastyluje vzhled stránky, se načítá pro každou stránku ze stejného místa. Prohlížeč si ji po prvním použití uloží do své vyrovnávací paměti. Jak vypadá výstup jedné stránky bez použití šablony pro nastylování vzhledu lze posoudit na obrázku 21.



Obr. 21: Struktura dokumentu bez nastylování vzhledu

5.2.2 CSS layout aneb kaskádové styly

CSS (Cascading Style Sheets), česky tabulky kaskádových stylů, je jazyk popisující způsob zobrazení webových stránek. Kaskádové styly se dají popsat jako šablony, kterými lze měnit vzhled dokumentu. Obsah a jeho struktura zůstává stále stejný, ale použitím různých CSS můžeme výslednou podobu dokumentu libovolně měnit a formátovat. Jazyk CSS byl vyvinut, stejně jako XHTML, konsorciem W3C. Podobně jako má (X)HTML své různé verze, tak i CSS má více svých podob. Existuje CSS1, CSS2, dále momentálně nejrozšířenější CSS 2.1 a ve vývoji je verze CSS3 [21].

Jak již bylo řečeno u jazyka HTML, během vývoje se do značkovacího jazyka pro strukturování dokumentu dostaly takové prvky, které definovaly vzhled. Žádoucí je však takový způsob tvorby, který oddělí vzhled dokumentu od jeho struktury a obsahu. Když se tomuto úkolu vyhnulo HTML, muselo přijít něco nového. Tím je jazyk XHTML, určený pro čisté strukturování dokumentu, a jazyk CSS pro definici vzhledu.

V době vzniku CSS převládal při tvorbě webů tzv. *tabulkový layout*. Rozmístění prvků na stránce a definitivní vzhled se řešil tabulkou, případně více tabulkami. Velmi nepraktické a nevhodné, nicméně masivně rozšířené. Jiný způsob tehdy snad ani nebyl. Naštěstí díky rozšíření CSS přišel nový modernější způsob, tzv. *CSS layout*. Pomocí (X)HTML jazyka se vytvoří správná struktura dokumentu, prvkům se přiřadí správná sémantika podle jejich skutečného významu v dokumentu a o zbytek se postará CSS šablona. Strukturovaný dokument zobrazený na obrázku 21 lze tedy pomocí CSS nastylovat tak, aby získal požadovaný vzhled aplikace, který je pro porovnání na obrázku 17. Díky tomuto přístupu lze dokonce na jeden dokument aplikovat více CSS šablon a umožnit uživateli přepínat si je. Obsah zůstává stále stejný, ale vzhled si uživatel může změnit podle svých preferencí. Velmi praktické a efektivní. Není divu, že v dnešní době se všechny moderní weby realizují tímto způsobem.

Možnosti CSS jsou obrovské. S obsahem dokumentu lze pomocí kaskádových stylů provádět téměř cokoliv. Od skrývání vybraných částí dokumentů, přeskupení a rozmístění prvků obsahu, až po dynamické efekty, které se spustí např. přejetím kurzoru myši přes vybraný prvek stránky. Pokud provedeme krátké shrnutí, výhody CSS jsou zřejmé:

- Přináší mnohem větší možnosti formátování vzhledu.
- Umožňuje vytvoření centrální CSS šablony, kterou využijí všechny stránky.
- Zajišťuje konzistentní styl zobrazení prvků pro každou stránku.
- Striktní oddělení struktury obsahu od vzhledu dokumentu.
- Urychluje načítání webových stránek.
- Poskytuje možnost definovat různé styly pro různá výstupní zařízení.

Nevýhodou při použití CSS může být rozdílná podpora některých vlastností v prohlížečích. Ne všechny prohlížeče podporují kompletní sadu vlastností CSS 2.1 a některé si specifikaci CSS vykládají svým svérázným způsobem. Dochází tak k rozdílnému formátování vzhledu v různých

prohlížečích. Zejména Microsoft Internet Explorer je specifický tím, že se speciálně pro něj vymýšlejí různé „hacky“, kterými lze obejít chyby či odlišný způsob reprezentace některých vlastností. Jednoduchost použití CSS se tím trochu komplikuje, ale v dnešní době je již většina problémů prohlížečů s interpretací CSS známá a jsou pro ně snadno dostupná řešení. Situace se samozřejmě zlepšuje i s příchodem nových verzí prohlížečů. Ve chvíli, kdy všechny prohlížeče budou plně a správně podporovat CSS připravované verze 3, přijde nová doba v tvorbě webů. CSS3 umožní např. automatické rozdělení textu do určeného počtu sloupců, nahrazování a generování obsahu, pokročilé formátování dynamických částí dokumentu, více interakce pro uživatele webových stránek atd.

Pro realizaci webové aplikace jsem použil CSS layout. Snažil jsem se zachovat co největší čistotu struktury dokumentu a pomocí CSS šablony nastylovat stránky do působivé podoby uživatelského rozhraní.

Ze zajímavých technik stojí za zmínku např. *nastylování globální navigace webu*. Využívá technik nahrazení textu obrázkem v kombinaci s jednoobrázkovým rolloverem, které popisuje např. Petr Staníček ve svých článcích [31]. V HTML zůstává pouze nečíslovaný seznam, ve kterém má každá položka jednoznačný identifikátor. V CSS se pak položka menu napozicuje a následně nastyluje překrytím textu obrázkem, aniž by se porušila funkčnost odkazu. Působivého efektu, kdy se po přejetí myši přes odkaz položka zvýrazní, je dosaženo velmi jednoduše. Levá polovina obrázku obsahuje ikonu pro neaktivní a pravá pro aktivní položku menu. Uživateli se u položky menu zobrazuje pouze levá polovina obrázku. Při přejetí myši nad odkazem se obrázek posune a uživatel vidí jen pravou polovinu. Velmi efektivní řešení, které přináší do aplikace působivou interakci, ale nezpomaluje práci se stránkou. Ukázku realizace této techniky znázorňuje obrázek 22. V horní části je navigace bez CSS šablony a v dolní části je menu již nastylováno.

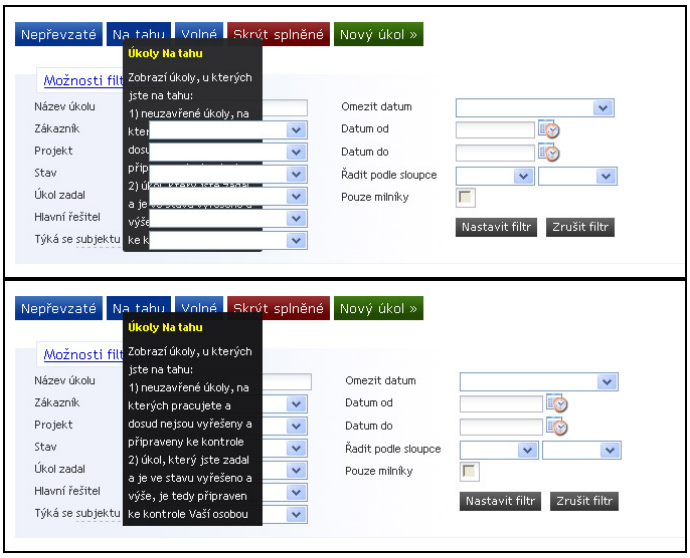
Obr. 22: Elegantní nastylování navigace pomocí CSS

Globální navigace není jedinou navigací na webu. Při návrhu jsem uvažoval dvě navigace z několika důvodů. Globální navigace bude společná pro všechny uživatele a zobrazí se všem přihlášeným uživatelům bez výjimky. Shrnuje nejčastěji používané funkce na dostupném místě. Druhá navigace by se měla generovat podle uživatelského oprávnění. Pro každého uživatele by tedy

mohla vypadat trochu jinak. Měla by obsahovat kompletní rozcestník ke všem funkcím, které uživatel může v aplikaci použít. Navigace musí být realizována ve formě rozbalovacího menu. V tomto případě jsem využil plně možností CSS. Rozbalovací menu pro svou funkčnost nepotřebuje JavaScript nebo něco podobného. Řešení využívá podporu pseudotřídy „:hover" v prohlížečích. Internet Explorer bohužel tuto pseudotřídu nepodporuje, takže je třeba použít berličku v podobě jednoduchého kódu JavaScriptu. Ten se však nevkládá nikam do stránky, ale jako externí soubor s definicí chování pro prohlížeč (htc soubor). Podobný princip řešení rozbalovacího menu pomocí CSS popisuje ve své knize Eric Meyer [32].

Obr. 23: Implementace rozbalovacího menu pomocí CSS

Při realizaci aplikace jsem se setkal s několika problémy, které pramení z rozdílné interpretace CSS různými prohlížeči. Jedním z nich je např. problém s překrýváním elementu <select> v prohlížeči Internet Explorer 6. Pro vykreslení tohoto ovládacího prvku totiž využívá grafické prostředí operačního systému, a proto vždy tento prvek zobrazí v nejvyšší vrstvě. CSS hodnotu pro pořadí vrstev ignoruje. Internet Explorer 7 již tento problém nemá. Ukázku problému si lze prohlédnout na obr 24. Způsobů řešení lze nalézt několik, ale žádný způsob mi nepřipadá vhodný.



Obr. 24: Vykreslení prvku <select> v prohlížečích Microsoft IE6 a IE7

5.2.3 JavaScript

JavaScript je objektově orientovaný skriptovací jazyk, který je interpretován na straně klienta. Používá se na WWW stránkách k ovládání různých interaktivních prvků GUI, kontrole formulářových uživatelských vstupů či dynamickým efektům [22]. Ačkoliv název JavaScript navozuje dojem pevné spojitosti s jazykem Java, není tomu tak. Tyto jazyky spojuje kromě názvu pouze podobná syntaxe. Původní název zněl LiveScript a na JavaScript byl přejmenován pouze z marketingových důvodů. Tvůrcem tohoto dnes velmi populárního a rozšířeného jazyka je společnost Netscape. Microsoft, jak bývá zvykem, nezůstal pozadu a zapojil se do vývoje svou vlastní verzí označenou jako JScript.

JavaScript se stal velmi populárním v době, kdy přišlo dynamické HTML. Tehdy nebyly dostupné žádné jiné technologie, které by umožňovaly nějakou interakci a dynamičnost na webu. Pomocí JavaScriptu se začaly i řešit takové konstrukce, které obecně není vhodné realizovat tímto způsobem. Docházelo k přílišnému zapojení do struktury dokumentu a mnohdy i k porušení sémantiky některých prvků. Dodnes je možné nalézt weby, které svou navigaci generují pomocí JavaScriptu. Celá navigace je pak nepřístupná vyhledávačům a také uživatelům, jejichž prohlížeč JavaScript nepodporuje nebo jej má zakázaný. Další problémy způsobují různé nekompatibility verzí JavaScript a různých verzí prohlížečů.

Postupem doby tedy opět dochází ke snaze soustředit se na správnou strukturu dokumentu, maximálně zpřístupnit všechen potřebný obsah stránek a definici vzhledu a uživatelského rozhraní realizovat vhodným způsobem pomocí CSS a JavaScript, případně dalšími technologiemi. Např. dynamická menu lze nyní řešit velmi elegantním způsobem, kdy se v XHTML nadefinuje struktura navigace pomocí vnořených nečíslovaných seznamů a o zbytek se postará CSS. Tento způsob je bezbariérový a popsal jsem jej v předchozí podkapitole.

Snahy posunout vývoj webů k čistším technikám a celý proces vývoje zjednodušit, má za důsledek vznik mnoha zajímavých JavaScript frameworků. Za zmínku stojí *MooTools framework*, jehož některé funkčnosti jsem použil i pro vývoj aplikace. MooTools je kompaktní, modulární, objektově orientovaný JavaScript framework, který umožňuje psát výkonný, flexibilní a na konkrétním prohlížeči nezávislý kód. Způsob psaní kódu je velmi elegantní, dobře zdokumentovaný a respektuje striktně důležité standardy [23]. Na obrázku 25 je znázorněn HTML kód připravený pro práci s MooTools. Jak je patrné, jedná se pouze o čisté HTML. Žádný zbytečný kód JavaScriptu přímo ve struktuře dokumentu. Obrázek 26 znázorňuje kousek MooTools kódu, kterým se danému prvku přiřadí potřebná dynamická funkčnost. Tento JavaScript kód lze umístit do externí souboru, takže se do struktury HTML dokumentu nedostává žádný další přebytečný kód a nezvětšuje se zbytečně velikost souboru. V tomto konkrétním případě se bude celý prvek s identifikátorem “test” schovávat či zobrazovat v závislosti na tom, na jaký odkaz uživatel klikne. Navíc dochází ke schovávání, resp. zobrazování prvku velmi efektním zasouváním, resp. vysouváním. Velmi

jednoduché a efektivní řešení pro vývojáře, velmi působivé a zajímavé chování prvků webu pro uživatele.

```
<a id="slideout" href="#">slideout</a> | <a id="slidein" href="#">slidein</a> | <a id="toggle" href="#">toggle</a>

<div id="test">
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt aliqua.
</div>
```

Obr. 25: MooTools nezasahuje přímo do HTML kódu [23]

```
var mySlide = new Fx.Slide('test');

$('slidein').addEvent('click', function(e){
            e = new Event(e);
            mySlide.slideIn();
            e.stop();
});

$('slideout').addEvent('click', function(e){
            e = new Event(e);
            mySlide.slideOut();
            e.stop();
});

$('toggle').addEvent('click', function(e){
            e = new Event(e);
            mySlide.toggle();
            e.stop();
});
```

Obr. 26: JavaScript využívající framework MooTools [23]

Podobnou funkčnost jsem implementoval do své aplikace. Narazil jsem však v některých prohlížečích na problém, pokud je schovávaný prvek umístěn na stránce pomocí CSS absolutním pozicováním. Opera verze 9.51 provede místo zasunutí prvku a tedy jeho schování něco jako posun celého bloku směrem nahoru. Stejně se chová i Firefox 2.0.0.15, ale ten ještě navíc ignoruje důležitou vlastnost CSS pro nastavení pořadí vrstev, takže blok se nevykresluje v nejvyšší vrstvě a tedy nad všemi ostatními prvky stránky. Internet Explorer verze 6 se chová stejně jako Opera, ale neprovede efekt zasunutí, resp. vysunutí. Prvek se zkrátka jen schová, resp. zobrazí. Internet Explorer 7 již efekt zasunutí, resp. vysunutí zvládá. Na stránkách MooTools jsem již našel ohlášení zmíněného „bugu“ při absolutním pozicování, ale řešení jsem dosud neobjevil. To jen dokazuje, jak může být problematické zkombinovat všechny potřebné technologie a zachovat přitom co největší kompatibilitu vůči různým prohlížečům a dokonce i vůči jednotlivým verzím konkrétního prohlížeče. Problém jsem částečně vyřešil umístěním bloku k hornímu okraji stránky, takže blok se zasunutím nahoru přeci jen před uživatelem schová.

Dalším velmi zajímavým JavaScript frameworkem je např. ExtJS [24]. V případě ExtJS přerůstá JavaScript knihovna do výkonného nástroje, kterým lze realizovat rozsáhlé a velmi zajímavé RIA aplikace srovnatelné s desktopovými programy. V úvodu této kapitoly byla zobrazena ukázka

jedné RIA aplikace realizované právě pomocí ExtJS. Uživatel si díky dostupné funkčnosti a vzhledu aplikace připadá jako kdyby pracoval s desktopovou aplikací nebo dokonce přímo s prostředím operačního systému. Těžko uvěřit, že celá tato webová aplikace funguje pouze s využitím JavaScriptu. Celý balík ExtJS je dost obsáhlý a jeho správné ovládnutí není úplně triviální. Z časových důvodů jsem se tedy rozhodl využít pouze některé jednoduché funkce z výše zmíněného frameworku MooTools.

5.2.4 Použitelnost a přístupnost

Při realizaci webů či webových aplikací naráží pečlivější vývojáři na pojmy použitelnost a přístupnost webových stránek. Jako zastánce těchto důležitých vlastností při tvorbě webů se v této podkapitole pokusím stručně shrnout vlastnosti dobře použitelného a přístupného webu. K důležitým vlastnostem uvedu vždy praktickou ukázku ilustrující, jak jsem se s touto konkrétní vlastností vypořádal při realizaci aplikace. Vzhledem k zaměření práce se pokusím tyto vlastnosti nastínit co nejstručněji.

Použitelnost webu je vskutku zajímavá oblast, která se bere v úvahu při tvorbě kvalitního webu. Cílem dobré použitelnosti je používání webových stránek bez nutnosti přemýšlet. Použitelný web je tedy takový web, který se návštěvníkům dobře používá. Dobře se na něm orientují, najdou rychle, co hledají, neztrácí se v nepřehledném množství informací a z webu mají jednoduše dobrý pocit [26]. Uznávaný odborník na použitelnost webu Steve Krug považuje výrok "Nenuťte uživatele přemýšlet" za nejdůležitější zákon použitelnosti [27]. Každá webová stránka by měla být maximálně intuitivní, pochopitelná a samovysvětlující. Pokud uživatel přijde na stránku a bez přemýšlení je mu jasné, o čem stránka je, jak ji používat a kde najde, co hledá, je to správně. Pokud uživatel přijde a začne si pokládat otázky, kam se to vlastně dostal, kam vede tento odkaz a proč je tahle věc pojmenována tímto způsobem, a kde je vlastně navigace, je to špatně. Pro potřeby této diplomové práce si vystačím se základními pravidly přehledného uživatelského rozhraní, které uživatele ničím nepřekvapuje, neklade mu do cesty žádné zbytečné překážky a už vůbec jej při práci nefrustruje. Více informací o použitelnosti lze nalézt např. ve výborné knize Steve Kruga [27].

Přístupnost webu je zaměřena více na uživatele se specifickými potřebami. Dá se říci, že přístupný web je bezbariérový web, který je dobře použitelný i pro handicapované uživatele. Přesněji pro uživatele, kteří mají specifické potřeby, protože označení handicapovaný uživatel může vyvolávat dojem, že jde pouze o zdravotně postižené občany. Na webu může být handicapovaným uživatelem např. barvoslepý, člověk s poruchou zraku či úplně nevidomý, člověk s poruchou učení či soustředění, důchodce, uživatel s rukou v sádře, uživatel alternativního zobrazovacího zařízení a další uživatelé se ztíženými pracovními podmínkami. Abych budoucím uživatelům aplikace nekladl do cesty zbytečné bariéry, snažím se o dodržení všech zásadních pravidel přístupnosti [25]. Pro více informací o přístupnosti webu lze využít skvělou a jedinou českou knihu na toto téma, kterou napsal David Špinar [26].

Obecná *pravidla použitelnosti* mluví např. o umístění loga stránky vlevo nahoře. Pokud uživatel klikne na logo, dostane se na úvodní stránku. Globální navigace je snadno rozeznatelná od ostatních odkazů a dalšího obsahu. Na všech stránkách je tato navigace přítomná a umístěná vždy na stejném místě. Na stránkách se uživatel snadno orientuje a hned je mu jasné, co může na webu dělat. V jakékoliv části webu by měl mít uživatel přehled, na které stránce se nachází. Všechny odkazy by měly být dostatečně odlišitelné od ostatního textu na stránce, nejlépe podtržením. U tlačítek by mělo být na první pohled jasné, že na ně lze kliknout. Celé ovládání by mělo být maximálně intuitivní a uživatel by neměl narazit na nečekaná překvapení a problémy. Všechny tyto důležité vlastnosti jsem při vývoji aplikace vzal v úvahu.

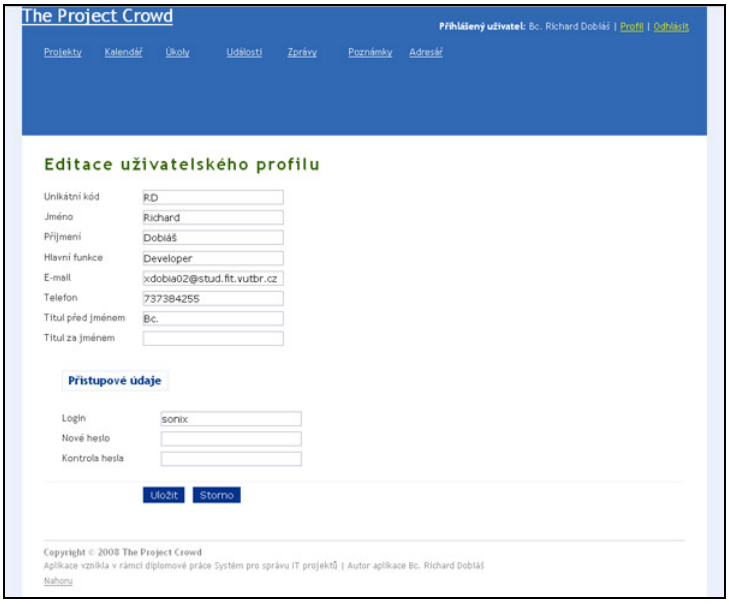
Velikost písma na stránkách je definována pomocí relativních jednotek. Uživatel si tak může pomocí standardních funkcí prohlížeče zvětšovat či zmenšovat písmo. Při změně velikosti nedochází ke ztrátě obsahu, funkčnosti či čitelnosti textu. Z hlediska přístupnosti je toto důležitá vlastnost aplikace. Uživatelé určitě ocení, že při dlouhodobější práci s aplikací mohou zvětšit písmo, aby nenamáhali příliš oči. Barvy popředí a pozadí textu jsou vůči sobě dostatečně kontrastní. Text je tedy dobře čitelný i pro uživatele, kteří mají zhoršený zrak. Opět důležitá vlastnost pro bezbariérovost webu a také pohodlí uživatelů.

Další pravidlo říká, že pokud jsou na webu použity netextové prvky, musí mít *textovou alternativu*. Toto řeším efektivně pomocí CSS, kdy všechny informace jsou textové a teprve přes ně se např. překrývají obrázky. Některé ukázky této techniky jsou zobrazeny v části o CSS, konkrétně se jedná především o stylování navigace. Snažil jsem se o co největší přístupnost jak navigace, tak i dalších částí dokumentu. Proto je vygenerovaná stránka poslaná uživateli prakticky jen čistý HTML text s pečlivě definovanou strukturou dokumentu a správnou sémantikou všech prvků. O zbytek se starají k tomu určené technologie jako je právě CSS.

Webová aplikace musí být funkční i v případě vypnutých rozšíření jako je např. *JavaScript*. Uživatel nemusí mít k dispozici některé nadstandardní funkčnosti a pohodlí, ale s aplikací může dále pracovat. Jeden dopad této podmínky jsem při vývoji aplikace opomněl. Některá funkční tlačítka v aplikaci odkazují na další stránky pomocí událostí onclick či onkeypress a po vypnutí podpory JavaScript jsou nefunkční. Framework, který jsem k vývoji aplikace využil, tento nešvar běžně používal, takže jsem jej bohužel převzal a nepříjemný důsledek jsem objevil až později. Některá tato tlačítka by šla předělat na běžné odkazy, takže by se závislost na JavaScriptu snížila či úplně odstranila. V případě intranetové aplikace sice lze po uživatelích požadovat podporu a zapnutí některých rozšíření, ale lépe by bylo udržet webovou aplikaci přístupnou i bez těchto rozšíření, pokud není nezbytně nutné využívat funkčnost některého doplňku.

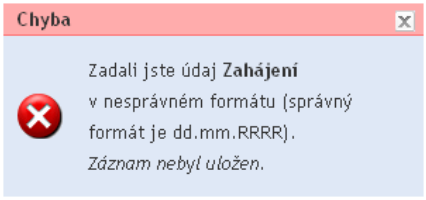
Webová aplikace je použitelná i v případě *vypnutého zobrazování obrázků*, či v případě úplně *vypnutých kaskádových stylů*. Na obrázku 27 je náhled stránky s vypnutými obrázky. Jak je zřejmé, všechen obsah je i nadále dostatečně čitelný. Náhled stránky s vypnutými CSS byl uveden již dříve v části o (X)HTML, konkrétně na obrázku 21. Pro nastylování vzhledu stránek na obrazovce

uživatele je použit pouze jeden CSS předpis. Pro nastylování vzhledu stránky pro tisk existuje další CSS šablona, která vhodně formátuje výstup stránky do tiskové podoby.



Obr. 27: Stránka s vypnutými obrázky nesmí omezit uživateli čitelnost obsahu

V aplikaci jsou ve velké míře využívány *formuláře*. Každý formulářový prvek má dle pravidel přístupnosti vhodný popisek, který vystihuje požadovaný obsah vstupního pole. Pokud se uživatel při vyplňování formuláře dopustí chyby, je aplikací jasně upozorněn, ve které části se dopustil chyby a dostane i jednoduchou nápovědu, v jakém tvaru by měla být hodnota zadána. Ukázka této nápovědy je na obrázku 28.

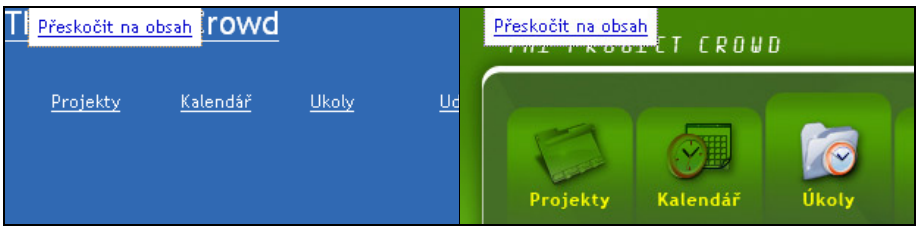


Obr. 28: Hlášení o chybně zadané hodnotě do formuláře

Aplikace by nebyla příliš dobře použitelná, pokud by se při výskytu chybně zadané položky uživatel vrátil na stránku, kde by mu z formuláře všechny pracně zadané údaje zmizely. Pokud tedy aplikace vrátí uživatele na formulář bez uložení zadaných dat, na straně serveru si ošetří, aby se uživatelem vložené hodnoty neztratily a ve formuláři se znovu objevily. Častým nešvarem nevhodně řešeného zpracování formulářů bývá hláška prohlížeče „Platnost stránky vypršela“, která uživatele velmi mate a často si není jistý, co má dělat. Stačí když se odešle formulář a uživatel klikne na tlačítko „Zpět“. To je způsobeno tím, že formulář zpracovává stejný skript, který generuje výstup do

prohlížeče. V tu chvíli se dostává do historie navštívených stránek a prohlížeč požaduje opakované odeslání formulářových dat, aby mohl stránku zobrazit. V mé aplikaci jsem zpracování formulářů řešil takovým způsobem, aby se uživatel při kliknutí na tlačítko „Zpět“ žádné takové hlášky nedočkal.

Ovládání webové aplikace by mělo být možné *bez použití myši*. Někteří uživatelé mohou momentálně pracovat pouze s klávesnicí nebo jim takovýto způsob práce připadá rychlejší a jsou na něj více zvyklí. Snažil jsem se tedy umožnit snadné ovládání webu i pro příznivce klávesnic. Pro položky navigace jsem definoval atributy „accesskey“, formuláře jsou bez problému přístupné přes klávesu TAB. Navíc je na začátku stránky umístěn odkaz *„Přeskočit na obsah“*, který po aktivaci odkazu přeskočí navigaci a hlavičku webu a skočí rovnou na hlavní obsah stránky. Využil jsem velmi pěkné techniky popsané v článku Davida Špinara [29] a tento odkaz jsem šikovně skryl za titulek webu. Teprve klávesou TAB se odkaz dostane do popředí a uživatel jej vidí. Viditelný odkaz je znázorněn na obrázku 29. Jakmile uživatel znovu stiskne TAB či někam klikne myší, odkaz se opět schová. Velmi efektní a velmi bezbariérové řešení.



Obr. 29: Zobrazený odkaz pro přeskočení menu (vlevo stránka bez CSS, vpravo s CSS)

5.3 Konkrétní způsoby řešení

V této kapitole bych rád stručně popsal některé obecné způsoby řešení, které jsem při implementaci aplikace použil. Většinou se jedná o běžně užívané techniky, které se používají při vývoji webových aplikací.

Funkce aplikace jsou uživateli přístupné až po úspěšném přihlášení. Proces *autentizace* jsem řešil obvyklým způsobem. Uživatel zadá svůj login a heslo. Skript vyhledá v tabulce uživatelů záznam s předaným loginem. Následně se porovnají zašifrované tvary uloženého a zadaného hesla. Pokud je heslo zadáno správně, skript vytvoří SESSION obsahující informace o přihlášeném uživateli. Při jakékoliv práci s aplikací se jednoduchou funkcí kontroluje platnost této SESSION. Pokud se kontrola nezdaří, aplikace uživatele odhlásí a upozorní jej na nutnost přihlášení. Pro uchování informace o platném přihlášení uživatele se na straně klienta používají Cookies. Uživatel tedy musí mít v prohlížeči Cookies povoleny, jinak se mu nepodaří do aplikace přihlásit.

Využití SESSION v aplikaci je celkem časté. Vzhledem k tomu, že protokol HTTP je bezstavový, není se čemu divit. Webová aplikace si potřebuje udržovat nějaký aktuální stav, ve kterém se uživatel právě nachází, potřebuje si pamatovat zadané údaje, nastavení apod. Použití

SESSION je velmi vhodné u zpracování formulářů. Pro zpracování všech formulářů v aplikaci jsem použil následující postup. Uživatel v aplikaci vyplní formulář a odešle jej na server. Pokud skript objeví v zadaných hodnotách chybu, uloží všechny uživatelem vyplněné hodnoty do SESSION a pošle uživateli zpátky formulář s chybovým hlášením a nápovědou, jakou hodnotu je třeba opravit. Pokud existuje SESSION, formulář se předvyplní hodnotami, které uživatel naposledy zadal. Díky tomuto způsobu nemusí uživatel opakovaně vyplňovat všechny položky formuláře, ale pouze opraví tu chybně zadanou hodnotu.

SESSION se používá také pro uložení všech filtrů, které si uživatel nastaví. U každého výpisu záznamů je možnost nastavit filtrování a nechat si tak zobrazit jen požadované záznamy. Přehled úkolů, událostí či zpráv má takový filtr k dispozici. Aplikace si nastavení filtrů pamatuje nezávisle na právě zobrazeném přehledu. Všechna nastavení aktivních filtrů se uchovávají pomocí SESSION. Jakmile uživatel filtr zruší, příslušná SESSION zanikne.

Ve většině formulářů je obsaženo vstupní pole formátu datum. Jak uživateli zpříjemnit práci s datovým typem datum bývá někdy oříšek. Každý uživatel může být zvyklý zapisovat datum v jiném formátu, ale formulář požaduje jednotný způsob zadání. Datumová vstupní pole se většinou předvyplňují aktuálním datem, díky kterému uživatel vidí požadovaný formát. Pro větší pohodlí uživatele při zadávání jiného datumu jsem využil hotové komponenty Kalendář, kterou použitý framework obsahuje. Vedle vstupního pole je umístěna ikonka znázorňující kalendář. Uživatel na ni klikne a otevře se mu nové malé okno s typickou tabulkou kalendáře. Po kliknutí na nějaký den se datum ve správném formátu vyplní do vstupního pole formuláře. Po odeslání formuláře ke zpracování se samozřejmě zadané datum kontroluje. Kontrola dokáže odhalit nesprávný formát zadané hodnoty, tj. pokud datum nebylo zadáno ve formátu „dd.mm.rrrr“, uživatel je upozorněn na špatně zadaný formát. Skript zpracovávající formulář dokáže odhalit i neexistující datum, tj. pokud uživatel zadá např. „30.2.2008“, dostane chybové hlášení o neplatném datu.

Rozsáhlejší webové aplikace mohou mít problémy s rychlostí. Důvodů, proč je aplikace pomalá, může být spousta. Často se aplikace začne zpomalovat po několika měsících provozu. Záleží samozřejmě na frekvenci využívání, množství ukládaných dat, počtu aktivních uživatelů pracujících ve stejnou dobu atd. Předně je třeba kvalitní datový model a optimalizované databázové dotazy. Při návrhu datového modelu jsem se snažil o dosažení co nejoptimálnějšího rozdělení dat. V databázových dotazech spojuji tabulky přes cizí klíče, které mají vytvořeny indexy. Samozřejmě v dotazech připojuji jen potřebné tabulky a pečlivě vybírám sloupce, které je třeba z databáze číst. Pro rychlejší zpracování dotazů jsou vytvořeny indexy na všech sloupcích, kde může existence indexu vést k rychlejšímu nalezení požadovaných dat. Práce s databází je jedna z nejdůležitějších částí, které by se měla věnovat pozornost v případě optimalizace webové aplikace.

Rychlost závisí také na způsobu implementace. Programátoři mohou psát kód úsporně, nebo zbytečně neefektivně. Mohou používat algoritmy, které se na konkrétní problém hodí dobře, nebo který se nehodí vůbec. Kvalita programových konstrukcí by se při optimalizaci měla tedy brát také

v úvahu. Pokud se k vývoji použije nějaký dostupný framework, je důležité, jak dobře je framework napsán. Špatně zvolený framework nezachrání ani velmi dobře napsané algoritmy aplikace. Při vývoj jsem použil velmi jednoduchý a přehledný framework. Má však jistou slabinu. Při použití Smarty šablon je velmi důležité nastavení *optimálního kešování* (od angl. slova „cache“ neboli vyrovnávací paměť). Bez kešování by generování stránek bylo znatelně pomalejší. Pokud se např. rekurzivně generuje velmi obsáhlý strom kategorií, může proces trvat i několik sekund. S použitím cache se zkrátí doba na setiny sekundy. Z tohoto důvodu je dobré mít na webovém serveru nainstalovaný nějaký PHP kešovací systém. Známé jsou např. APC nebo Memcache, které dokumentuje přímo PHP manuál.

Použití šablon může být ještě záludné v jedné věci. Při sestavení stránky se vygeneruje potřebný obsah a doplní se do šablony. Šablona se tak sestaví, zpracuje a teprve poté se odesílá výsledná stránka ke klientovi. Pokud je generovaný obsah velký, může být doba odezvy delší. Bez použití šablon by se dal generovaný obsah posílat do uživatelova prohlížeče postupně z výstupního bufferu. U webových aplikací je také dobré myslet na to, že některé prohlížeče zobrazí tabulky až po jejich úplném načtení. Při realizace aplikace jsem tabulky využil pouze pro výpis záznamů, které musí mít tabulkový formát, resp. bez tabulek by jejich zobrazení bylo velmi nešikovné. Zbývající výstup stránek je maximálně optimalizován. Do prohlížeče dorazí pouze čistý HTML dokument a pomocí společných externích souborů se každá stránka styluje a obohacuje o nadstandardní funkčnost. Externí soubory si prohlížeč uchovává ve své vyrovnávací paměti, stejně tak i obrázky, takže v tomto ohledu je realizace výstupu navržena velmi efektivně.

5.4 Neimplementované části

Oproti specifikaci požadavků a návrhu systému jsem některé funkčnosti v aplikaci nerealizoval. Důvodů je hned několik. Prvním důvodem je časová náročnost projektu. Implementovat takový systém v celém rozsahu by zvládl spíše několikačlenný projektový tým. Pro jednoho programátora je to nemožný úkol. Druhým důvodem jsou některé spory při analýze a návrhu. Pro potřeby propojení aplikace s informačním systémem je potřeba ve všech fázích projektu postupovat velmi opatrně. Vhodnější bude ponechat realizaci specifičtějších funkcí a potřebných politik vývojářskému týmu, který má větší zkušenosti se zmíněným informačním systémem a bude mít dostatek časové kapacity na vypracování podrobnějších specifik. Politikami se zde myslí např. bezpečnostní politika, systém uživatelských oprávnění apod. ESO9 má uživatelská oprávnění řešena velmi důkladně a univerzálně pomocí šablon. Převzít tento způsob řešení bylo nad časové možnosti fáze analýzy. S důkladnější specifikací souvisí i třetí důvod. Pokud bych momentálně provedl realizaci některých částí, mohl bych přidělat spoustu práce týmu, který bude na vývoji systému pokračovat. Příkladem je již zmíněné řešení uživatelských oprávnění. Pokud bych jej vyřešil po svém, promítne se toto řešení do celého systému, což se předělává velmi pracně a zbytečně rostou náklady na projekt.

Bezpečnostní politiku a uživatelská oprávnění jsem realizoval minimálně. Momentálně aplikace nekontroluje, co uživatel může nebo nemůže a to ani s ohledem na uživatelskou skupinu, do které je přiřazen. Soustředil jsem se jen na základní omezení, která vyplývají z charakteru funkcí systému. Např. u zpráv nelze editovat odchozí zprávu, pokud ji příjemce již četl. Úkol může smazat pouze uživatel, který jej zadal. Stejně tak to platí s událostmi. Poznámky jsou osobního charakteru, takže u těch jsou zobrazení a možné úpravy omezeny na aktuálně přihlášeného uživatele. Oprávnění jsou prozatím realizována na této základní úrovni, kdy se aktivity uživatele omezují pouze minimálně.

Podle specifikace požadavků měla být dostupná funkčnost sledování projektů. Tuto komplexní funkčnost jsem nerealizoval, protože je velmi obsáhlá. ESO9 umožňuje sledovat stav úkolů z pohledu všech uživatelů. Manažer vidí, kdo má kolik úkolů k řešení, které již vyřešil, jestli čeká na dokončení úkolu kolegy apod. Obecně celý proces zadávání a kontroly úkolů je docela složitý. Úkol má mnoho stavů od „pořízeno“ až po „vyřešeno“. Mezi tím dochází k dokončení a předání ke kontrole, následně se řešení integruje, testuje atd. Uživatel vidí, u kterých úkolů je „na tahu“. Tj. pokud má nějaký úkol řešit a dosud není ve stavu „vyřešeno“, je „na tahu“. Pokud uživatel zadal někomu úkol a ten jej dovedl do stavu „vyřešeno“, zadavatel úkolu musí řešení zkontrolovat a „na tahu“ je tedy zadavatel. Řešit takové funkčnosti je pro aplikaci potřebné, nicméně z časových důvodů jsem ponechal všechny možnosti sledování otevřeny pro další vývoj. Zatím je pouze možné úkolu nastavit hodnotu splnění v procentech, odhad zbývající doby řešení úkolu a vykazovat již odpracovanou dobu. Z těchto údajů by bylo možné zpracovávat výkazy a reporty. V přehledu úkolů je výrazným způsobem zobrazen aktuální stav splnění úkolu.

Kalendář jako možnost sledovat všechny osobní schůzky, úkoly atd. jsem již nestihl implementovat. Tato funkčnost je na realizaci svým rozsahem náročná a zahrnuje práci se všemi uloženými procesy filtrovanými podle uložených hodnot termínů, zahájení apod. Pro dostatečnou rychlost kalendáře by navíc bylo potřeba využít možností Ajax technologie, což by znamenalo další čas potřebný pro dokonalejší nastudování tohoto způsobu využití JavaScriptu.

Nastavení a správa systému je soubor důležitých funkcí pro úpravy chování systému. K těmto funkcím by měl mít ale přístup pouze administrátor. Některé běžné funkce má v sobě již implementován framework. Tyto funkce však vycházejí z jiného datového modelu. Z tohoto důvodu jsem přenechal úpravy těchto funkcí vývojářům v KomTeSe, kteří celý framework budovali.

Poslední zajímavou funkčností, kterou jsem měl původně v úmyslu s velkým nadšením realizovat, je „Řídicí panel“. Ten by měl být vstupní stránkou do aplikace a shrnutím důležitých dat na jednom místě. Řídicí panel by mohl obsahovat seznam nepřevzatých úkolů, nepřečtených událostí, aktuální denní plán, kalendář, upozornění na blížící se termíny úkolů, schůzky, nově příchozí zprávy apod. Inspiraci jsem hledal např. ve službě Google Analytics, kde je řídicí panel řešen velmi efektivně. Panel si lze poskládat z různých funkčních bloků podle přání. Řídicí panel mé aplikace by tedy mohl sdružovat všechna možná upozornění a data, které by chtěl mít uživatel stále na očích. Bohužel implementace tohoto řídicího panelu se mi již nevešla do časového plánu.

6 Testování

Fáze testování má za úkol ověřit správnost kódu systému a otestovat, zda se systém chová podle zadaných požadavků. Žádný software není bezchybný. Během implementace se může do kódu dostat spousta chyb kvůli nepřesné komunikaci programátorů, množství situací, ve kterých program musí pracovat korektně atd. Testování by mělo tyto případné chyby objevit.

Již během vytváření detailního plánu implementace v předchozí etapě vývoje se také vytváří detailní plán testování. Ten zahrnuje testování přírůstků, testování jejich integrace a samozřejmě i testování uživatelských vstupů [30]. Vzhledem k rozsahu projektu a časovému fondu jsem plán testování nevytvářel. Testování jsem prováděl průběžně při dokončení každé ucelenější funkčnosti a nalezené chyby jsem hned opravoval.

Častou příčinou existence chyb byla má nepozornost při některých konstrukcí či sestavování SQL dotazů. Další chyby způsobovalo nedostatečné ošetření zadaných vstupů přes formulář. Záludné chyby také vznikaly při kopírování částí programových konstrukcí, které bylo třeba s úpravami využít v jiných částech systému. Při tomto způsobu se často něco přehlédne.

Během testování a běžné práce v aplikaci jsem došel k závěru, že je vhodné skutečně vytvořit detailní plán testování a následně provést důkladné metodické otestování všech funkcí systému.

6.1 Testování webové aplikace

V případě webových aplikací se testování ještě rozšiřuje na kontrolu stejného zobrazení a funkčnosti ve všech potřebných internetových prohlížečích. Během vývoje jsem kontroloval správnou funkčnost aplikace v prohlížečích Opera 9.51, Firefox 2.0.0.15, Microsoft Internet Explorer 6 a Microsoft Internet Explorer 7. Až na drobné rozdíly ve vykreslení některých prvků je výsledná aplikace použitelná v jakémkoliv z těchto prohlížečů. Vzhledem k použitým technologiím a vhodným výběrem některých technik realizace se lze domnívat, že by aplikace měla bez problému fungovat ve všech novějších prohlížečích. Testování u webů může jít ještě dále než jen k pouhému porovnání výsledku ve více prohlížečích. Mám na mysli testování použitelnosti a přístupnosti webu. Obě tyto oblasti se testují zvlášť a jiným způsobem.

Testování použitelnosti může zahrnovat odbornou studii, která shrne běžné nedostatky, které by mohly snižovat použitelnost webu. Mnohem zajímavější je provést testy použitelnosti přímo s uživateli. Cílová skupina lidí dostane sadu úkolů. Sleduje se, jak si uživatelé s úkoly na webu poradí. Odborné vyhodnocení testů pak vede k definici problémů, které uživatelé měli, a samozřejmě i k doporučením, jak tyto problémy nejvhodněji řešit. V případě webové aplikace můžeme zjistit, že jisté funkčnosti by měly být dostupné přes jedno kliknutí, formulář by měl obsahovat jasnější popisky, navigace by měla být umístěna mnohem blíže k obsahu apod. Po realizaci navrhovaných

úprav bude mít aplikace spokojenější uživatele. Během realizace jsem se snažil, aby se výsledná aplikace uživatelům používala co nejsnadněji a během práce se stránkami měli dobrý pocit a nenaráželi na zbytečné překážky.

Testování přístupnosti webu se zaměřuje na kontrolu webových stránek z pohledu splnění pravidel přístupného webu [25]. Existují automatické nástroje, které mohou některé nedostatky odhalit. Pokud ale má být testování úplně, musí se provést hlubší analýza. Při vývoji aplikace jsem bral v úvahu všechny důležitá pravidla a snažil se je plně dodržovat. Výsledná webová aplikace by tedy měla být maximálně bezbariérová.

Některé konkrétní zajímavé ukázky z testování použitelnosti a přístupnosti jsem uvedl v podkapitole věnované tématu použitelnost a přístupnost.

7 Nasazení

Nasazení je závěrečnou fází procesu vývoje softwarového projektu. Následovat může už jen ukončení a zhodnocení projektu, které je spíše administrativního rázu. Případně by byl možný další vývoj v podobě údržby a dlouhodobého rozvoje systému o nové funkčnosti. To už by ale bylo zřejmě předmětem nového projektu. Hlavním cílem etapy nasazení je instalace systému a předání celého projektu zadavateli. Součástí této fáze může být dokončení uživatelské dokumentace a zaškolení zadavatele v užívání systému [30].

7.1 Funkční demo na webu

Funkční webovou aplikaci jsem pro snadné vyzkoušení nainstaloval na webový server a zpřístupnil na adrese http://diplomka.komtesa.com/. Přístupové údaje pro vstup uživatele:

- login: demo
- heslo: vutbr

7.2 Minimální požadavky

Realizovaná aplikace má minimální požadavky na systém jako běžná webové aplikace. Pro správný chod aplikace postačuje webový server s běžným nastavením a podporou PHP minimálně verze 4 (doporučeno PHP5). Kvůli šablonovacímu systému Smarty je doporučen PHP kešovací systém a vhodné nastavení Smarty systému. Po práci s daty je potřeba databázový systém MySQL minimálně verze 4 (doporučeno MySQL5). Na straně klienta postačuje běžný internetový prohlížeč s podporou CSS 2.1, zapnutým JavaScript a povoleným ukládáním Cookies.

7.3 Instalace

Instalace je velmi snadná. Na webový server je potřeba nahrát všechny soubory aplikace. Pro adresář „/files/“ je nutno nastavit přístupová práva pro zápis. Adresář „/tpl/templates_c/“ vyžaduje rovněž přístupová práva pro zápis. Soubor „/const.php“ je potřeba upravit dle potřeb. Většinou postačí nastavit přístupové údaje k databázi. Dále je třeba vytvořit databázi a spustit v ní sql skripty „/db/create.sql“ a následně „/db/inicializacni-data.sql“. Po instalaci je vhodné ze serveru adresář „/db/“ odstranit. Nyní je aplikace připravená ke spuštění a dostupná přes běžný webový prohlížeč.

8 Závěr

Cílem mé diplomové práce bylo seznámit se s problematikou projektového řízení se zaměřením na IT oblast. Při procházení různých zdrojů jsem narazil na velmi zajímavé poznatky a dostupná řešení, o kterých jsem dříve neměl tušení. Oblast řízení projektů je velmi obsáhlá a překvapivě ne úplně správně pochopena u mnohých projektových řešitelů. Po úvodním seznámení s plánováním a řízením projektů jsem provedl analýzu požadavků na software, který by měl při řízení projektů pomáhat zefektivnit důležité operativní procesy. Systém jsem po dlouhých analýzách a nekonečných návrzích implementoval. Z možných funkčností jsem implementoval jen základní a to z toho důvodu, že rozsah analýzy a návrhu výrazně převyšuje možnosti programování jednoho vývojáře.

Již od začátku jsem měl k dispozici konzultanta, který mé analýzy upřesňovat a posouval blíže k praxi. Záměrem bylo vytvořit webovou aplikaci, která bude propojitelná s intranetovým informačním systémem ESO9. Díky tomu bude možné přistupovat k důležitým datům a procesům odkudkoliv, kýmkoliv a bez nutnosti nakupovat další uživatelské licence informačního systému.

Mým úkolem nebylo vytvořit během několika měsíců komplexní robustní aplikaci pro řízení projektů. To ani není v silách jednoho člověka. Soustředil jsem se tedy na dobrou analýzu a návrh systému, který bude snadno rozšiřitelný a bude umožňovat další vývoj v zázemí celého vývojářského týmu. Aplikaci jsem realizoval v rozsahu, který jsem byl schopný v rámci diplomové práce zvládnout.

Nedokončil jsem celý projekt v takovém rozsahu, jaký jsem si na počátku představoval. Důvody popisuji v následující podkapitole. Některé důvody jsem zmínil přímo u jednotlivých fází projektu. Ačkoliv jsem nedosáhl všech cílů, které jsem si při výběru zadání projektu předsevzal, myslím, že jsem vybudoval dostatečný základ dobrého projektu, který si zaslouží pozornost a který bude po dalších rozšíření nasazen do reálného provozu a bude efektivně plnit svůj účel.

Během diplomové práce na téma projektového řízení jsem si dostatečně uvědomil, jak moc důležité je kvalitní plánování a řízení. Snažil jsem se všechny podstatné vlastnosti zachytit do této práce, aby si každý manažer, vývojář či IT odborník uvědomil základní důležitou myšlenku: *„Začněme konečně brát projektové řízení vážně!“*

8.1 Chyby a problémy při řešení projektu

Při realizaci projektu jsem se setkal s několika nepříjemnými problémy, které jsem byl nucen řešit. Prvně je třeba říci, že jsem neodhadl náročnost celého projektu. Navrhnul jsem tak rozsáhlý systém, který nejsem schopen sám v rámci diplomové práce realizovat v plné specifikaci. Během analýzy docházelo k častým změnám požadavků. Jak ze strany konzultanta, tak i ze strany mé. Když mě napadla vhodná funkčnost z hlediska dobré použitelnosti aplikace, snažil jsem se ji zahrnout do nových požadavků.

Další problémy se projevovaly při implementaci. Zřejmě z důvodu nedostatečně detailní analýzy a návrhu jsem často přemýšlel, jak vlastně má být konkrétní funkčnost řešena, jaké má mít vlastnosti, omezení a jak k ní přistupovat. Z důvodu kompatibility s informačním systémem jsem musel využívat spoustu konzultací, abych zjistil, jestli je potřeba některé vlastnosti řešit konkrétním způsobem. Protože zmíněný informační systém neznám moc dobře, bylo potřeba vždy zastavit a potřebnou vlastnost dohledat. Celý tento proces velmi zdržoval a krátil tak časový fond projektu.

Přes všechny problémy při řešení projektu vznikl slibný základ, který bude nadále rozvíjen celým týmem pod záštitou konzultanta mé diplomové práce. Snažil jsem se postavit takový systém, který umožní snadný rozvoj do budoucna. Pokud jsem při řešení dospěl k závěru, že přílišně detailní implementace určité funkčnosti vloží do dalšího vývoje bariéry, ponechal jsem raději tuto funkčnost otevřenou a implementovanou pouze v nutném základu.

8.2 Přínos projektu

Již žádný stres během projektového vývoje. Výsledná webová aplikace by měla přinést více chuti do úkonů nutných ke správnému řízení projektu. V jednom programu má uživatel možnost číst a zadávat nové úkoly, sledovat aktuální dění ve firmě, zapsat si rychlou poznámku, zaznamenat do událostí telefonický rozhovor se zákazníkem, dopisovat si s kolegy přes vzkazy, vyhledat měsíc starou komunikaci, kontrolovat včasné plnění úkolů atd.

Snažil jsem se o rychlou dostupnost všech možných funkcí, které IT pracovník během dne potřebuje. Navíc by mělo být vše velmi přehledné a jednoduché. Cokoliv je potřeba, je dostupné z jednoho místa, přes jednu aplikaci. Zalíbila se mi myšlenka jednotného úkolníku, který používá VSTS. Proto jsem aplikaci realizoval tak, aby měl každý přehled o svých úkolech, ale také o práci svých kolegů a obecně celé firmy. Tento přístup utužuje vývojový tým a produkuje kvalitní projekty.

Detailně jsem se zaměřil na co nejkvalitnější způsob pro realizaci aplikace, která má být dostupná prostřednictvím webu. Již dlouhá léta se zabývám vývojem webů, webových aplikací a konzultační činností v oblasti webu a podnikání na Internetu. Všechny mé zkušenosti a znalosti jsem vnesl do tohoto projektu. Drobnosti, kterým jsem se z časových důvodů nevěnoval detailněji, lze celkem snadno dovést k úplné dokonalosti.

Díky mým pracovním povinnostem mám již i nějaké zkušenosti s vedením projektů. Setkávám se často s problémy, které vedou k nesplnění termínu či překročení původního rozpočtu. Vím o skutečnostech, které vedou k nechuti vývojářů provádět řádné sledování a vykazování odvedené práce. Jsou mi také jasné důvody, proč nebyl splněn některý úkol či proč se všechny zainteresované osoby neseznámili se zápisem z porady. Všechny tyto zkušenosti jsem bral v úvahu při návrhu a realizaci aplikace. Snažil jsem se vytvořit jednoduchý a přehledný nástroj, který bude všem členům týmu pomáhat a nikoliv jim přidělávat starosti a zbytečnou práci.

Podle mých zkušeností je velmi důležitá grafická podoba a dobrá použitelnost uživatelského rozhraní. Vytvořil jsem tedy zajímavý design webu a zapracoval ho do příjemného prostředí, které uživatele nebude odrazovat nebo dokonce frustrovat svou složitostí a nepřehledností. Navíc je uživatelské rozhraní realizováno způsobem, který umožňuje bez problému kdykoliv změnit vzhled celé aplikace. Snadné by také bylo realizovat přepínání vzhledu, takže by si každý uživatel mohl přizpůsobit aplikaci svým potřebám. Při realizaci rozhraní jsem se snažil o co největší přístupnost, tzn. aby aplikace žádnému uživateli nekladla do cesty jakékoliv bariéry.

8.3 Možná rozšíření projektu

Vzhledem k povaze tohoto projektu jsou cesty pro rozšíření otevřeny dokořán. Nejdříve by mělo dojít k detailnějšímu dokončení všech potřebných funkčností, které aplikace bude potřebovat pro využití v reálné praxi. Pro vytvoření navržené aplikace a její další rozvoj je zapotřebí několikačlenný tým, který se projektu bude plně věnovat. Během realizace mě napadaly různé možnosti, jak aplikaci vylepšit. Pokusím se tedy stručně nastínit možné směry dalšího vývoje.

Pro dnešní dobu je typická integrace různých řešení. Uživatelský kalendář by mohl být propojen se službami typu Google iCalendar, aby uživatel mohl se svými kolegy či přáteli sdílet např. termíny schůzek, výročí apod. Vzhledem k velké oblibě aplikace Microsoft Outlook bych doporučoval rovněž integraci systému s tímto užitečným pomocníkem.

Již od začátku projektu bylo počítáno s tím, že se aplikace bude napojovat na vnitropodnikový informační systém. Bylo by tedy vhodné vytvořit univerzální rozhraní, přes která budou oba systémy komunikovat. Samozřejmě je důležité vyřešit oboustrannou automatickou synchronizaci. Aplikaci lze pak zpřístupnit např. externím programátorům, kteří nebudou mít přístup do informačního systému, ale mohou se účastnit projektových procesů prostřednictvím webové aplikace.

Přímo nutným rozšířením bude definice a implementace vhodných bezpečnostních politik. Informační systém, který byl uvažován pro pozdější propojení, má řešena uživatelská oprávnění velmi dobře, ale složitě. Pro webovou aplikaci bude dobré definovat nutnou podmnožinu těchto omezení a vhodným způsobem je implementovat.

Projektové řízení je také o sledování a vykazování práce. V této oblasti má vyvinutá aplikace ještě spoustu nevyužitého místa. Projektový vedoucí by měl mít dostupné funkce pro přehledné sledování stavu projektu, různé statistiky, reporty, hlášení rizik apod. Členové týmu by měli mít možnost vykazovat svou odvedenou práci, za kterou jsou odměňováni. Informační systém bývá obvykle propojený se systémem docházky, která slouží pro zúčtování a odměňování zaměstnanců. Tyto systémy bývají velmi složité. Pro potřeby webové aplikace by postačovalo umožnit uživatelům vykazovat jednoduchým způsobem jejich odpracované hodiny a data následně zpracovat již docházkovým systémem.

V neposlední řadě by bylo ideální do aplikace přizvat i zadavatele projektů, tedy zákazníky. Systém by mohl zákazníkovi předkládat průběžné reporty o stavu odvedené a zbývající práce, dosavadních nákladech, upozorňovat na překročení termínu či rozpočtu apod. Pro zjednodušení procesů spojených se zadavatelem bych doporučoval také zavést do aplikace předávání a přebírání procesů projektu. Pokud programátor splní nějaký úkol, zanese tuto informaci do systému a zákazník hned může vidět výsledek. Pokud souhlasí, potvrdí jej a tím může dojít k profinancování odvedené práce. Šlo by o takové elektronické schvalování, kdy lze jakoukoliv akci dohledat a prokázat. Aplikace již od počátku s možným přístupem zákazníka počítá. Implementuje také poznatek z praxe, kdy je vhodné vytvářet dvě varianty popisu úkolu či obecně procesu. Jednu variantu pro vývojový tým, který používá svoji terminologii a zkrácené zápisy. Druhou variantu pro zákazníka, který má obvykle s pochopením zápisů problém a požaduje takovou verzi popisu, kterou pochopí každý laik.

Projektové řízení by mělo co nejvíce zefektivnit obchodní procesy v jednotlivých fázích projektu. Budoucnost bude zřejmě v integraci všech potřebných řešení do jednoho užitečného celku. Začal jsem s vývojem jednoduché aplikace, která těžko může dosáhnout kvalit velkých řešení typu MS Project nebo VSTS. Rozšiřovat je tedy možné prakticky do nekonečna. Je ale třeba zvážit, zda v určitém momentě nebude levnější a optimálnější přejít k větším dostupným řešením. Přeci jen některá hotová řešení mají za sebou několikaletý vývoj, kterému se věnují početné týmy lidí.

Literatura

[1] Martínek, Z., Kreslíková, J.: *Management projektů [Studijní opora]*. Vysoké učení technické v Brně. Fakulta informačních technologií, 2007, 98s. Dokument dostupný na URL https://www.fit.vutbr.cz/study/courses/MPR/private/mpr-opora-v2.pdf (prosinec 2007)

[2] Katolický, A., Šlechtová, Y.: *Critical Chain (CCPM)*. Dokument dostupný na URL http://www.ifm.zcu.cz/CCPM1.htm (prosinec 2007)

[3] Podpěra, D.: *Jak se udržet naživu při řízení projektů*. Dokument dostupný na URL http://www.zive.cz/default.aspx?article=108646 (prosinec 2007)

[4] *Faktory úspěchu řízení IT projektů*. Dokument dostupný na URL http://www.unicorn.eu/cz/press/clanek.php?id=5274 (prosinec 2007)

[5] Rosenau M. D.: *Řízení projektů*, Computer Press, 2007, ISBN 978-80-251-1506-0

[6] Schwalbe K.: *Řízení projektů v IT*, Computer Press, 2007, ISBN 978-80-251-1526-8

[7] *About PMI*. Dokument dostupný na URL http://www.pmi.org/WhoWeAre/Pages/About-PMI.aspx (prosinec 2007)

[8] Václavková, R.: *Řízení projektů vědy a výzkumu [Studijní opora]*. Dokument dostupný na http://moodle.vsb.cz/moodle/file.php/750/Material/Studijni_opora_-_projekty.pdf (prosinec 2007)

[9] *Metodologie*. Dokument dostupný na URL http://cs.wikipedia.org/wiki/Metodologie (prosinec 2007)

[10] Bartík, V., Květoňová, Š., Zendulka, J.: *Analýza a návrh informačních systémů [Studijní opora]*. Dokument dostupný na URL https://www.fit.vutbr.cz/study/courses/AIS/private/ (prosinec 2007)

[11] Vondrák, I.: *Úvod do softwarového inženýrství [Studijní opora]*. Dokument dostupný na URL http://vondrak.cs.vsb.cz/download/Uvod_do_softwaroveho_inzenyrstvi.pdf (prosinec 2007)

[12] Petrlík, L.: *Základy softwarového inženýrství [Studijní opora]*. Dokument dostupný na URL http://www.kiv.zcu.cz/%7Eluki/vyuka/zswi/predn/zswi2005.pdf (prosinec 2007)

[13] *Capability Maturity Model Integration*. Dokument dostupný na URL http://en.wikipedia.org/wiki/Capability_Maturity_Model_Integration (leden 2008)

[14] *CMMI usnadňuje kvalitní vývoj softwaru*. Dokument dostupný na URL http://www.cw.cz/cwarchiv.nsf/clanky/DDB466AB4321C786C125702E00485F6E?OpenDocument (leden 2008)

[15] Guckenheimer S., Perez J.: *Efektivní softwarové projekty*, Zoner Press, 2007, ISBN 978-80-86815-62-6

[16] *Rich Internet Application.* Dokument dostupný na URL http://www.symbio.cz/slovnik/rich-internet-application.html (červenec 2008)

[17] Cajthaml, M.: *Silverlight nebo Flash?* Dokument dostupný na URL http://www.symbio.cz/clanky/silverlight-nebo-flash.html (červenec 2008)

[18] *XHTML*. Dokument dostupný na URL http://en.wikipedia.org/wiki/XHTML (červenec 2008)

[19] Snížek, M.: *XHTML - vývoj (X)HTML a jeho možnosti*. Dokument dostupný na URL http://interval.cz/clanky/xhtml-vyvoj-x-html-a-jeho-moznosti/ (červenec 2008)

[20] Burget, R., Hruška, T.: *Internetové aplikace (WAP) II. (část SGML, HTML, CSS, DOM) [Studijní opora]*. Dokument dostupný na URL https://www.fit.vutbr.cz/study/courses/WAP/private/opory/OporaWAP2SGMLHTMLCSSDOM.pdf (červenec 2008)

[21] *Cascading Style Sheets*. Dokument dostupný na URL http://cs.wikipedia.org/wiki/Cascading_Style_Sheets (červenec 2008)

[22] *JavaScript*. Dokument dostupný na URL http://cs.wikipedia.org/wiki/JavaScript (červenec 2008)

[23] *MooTools, a compact javascript framework*. Dokument dostupný na URL http://www.mootools.net/ (červenec 2008)

[24] *Ext JS: Cross-Browser Rich Internet Application Framework*. Dokument dostupný na URL http://www.extjs.com/ (červenec 2008)

[25] *Pravidla tvorby přístupného webu*. Dokument dostupný na URL http://www.pravidla-pristupnosti.cz/ (červenec 2008)

[26] Špinar, D.: *Tvoříme přístupné webové stránky*, Zoner Press, 2004, ISBN 80-86815-11-0

[27] Krug, S.: *Web design: Nenuťte uživatele přemýšlet,* 2. vydání, Computer Press, 2006, ISBN 80-251-1291-8

[28] Rosa, J.: *PRADO*. Dokument dostupný na URL http://blog.php-group.cz/wp-content/uploads/2008/04/prado-jan-rosa.pdf (červenec 2008)

[29] Špinar, D.: *Přeskakovací odkazy nejsou jen pro nevidomé*. Dokument dostupný na URL http://pristupnost.nawebu.cz/weblog/blogpost.php?post=94 (červenec 2008)

[30] *Metodika projektového řízení objektově orientovaného vývoje*. Dokument dostupný na URL http://www.fit.vutbr.cz/study/courses/MPR/public/metodika-oo/index.html (červenec 2008)

[31] Staníček, P.: *WellStyled.com: Archiv příspěvků*. Dokument dostupný na URL http://www.wellstyled.com/archive.html (červenec 2008)

[32] Meyer, E.: *Eric Meyer o CSS – pokračujeme s kaskádovými styly profesionálně!* Zoner Press, 2005, ISBN 80-86815-17-X

Seznam příloh

Příloha 1. CD