Česká zemědělská univerzita v Praze
Provozně ekonomická fakulta

Návrh a tvorba internetových aplikací

Disertační práce

Autor: Ing. Vratislav Čermák

Školitel: Doc. Ing. Zdeněk Havlíček, CSc.

Katedra informačních technologií

Praha 2009

Poděkování:

Děkuji **Doc. Ing. Zdeňku Havlíčkovi, CSc.** za odborné vedení doktorské disertační práce a poskytnuté rady při jejím zpracování.

Dále bych chtěl podčkovat Doc. PhDr. Ivanč Švarcové, CSc. a kolegům z Katedry informačních technologií Provozně ekonomické fakulty za jejich pomoc, připomínky a rady, které mi pomohly při jejím zpracování.

Poděkování patří také mé manželce a rodičům za podporu v průběhu celého doktorského studia.

Souhrn

Předkládaná disertační práce se zabývá optimalizací vývojového procesu internetových aplikací a je rozdělena do dvou základních částí. V první části jsou analyzovány teoretické zdroje zabývající se zvláštnostmi a specifiky internetových aplikací a běžnými postupy jejich návrhu a tvorby. V práci jsou analyzovány činnosti jednotlivých projektových rolí a způsoby jejich zapojení do vývoje vlastní aplikace. Na základě průzkumu mezi internetovými agenturami a členy projektových týmů jsou formulovány požadavky na zefektivnění celého vývojového procesu.

Na základě získaných poznatků a provedeného průzkumu je navržena nová metodika pro efektivní tvorbu webových aplikací, která naplňuje požadavky definované v první části práce. Metodika se zaměřuje na vývojový proces, přesto je nezávislá na konkrétním technickém řešení a je tedy aplikovatelná v různých prostředích.

Použití metodiky je ověřeno a demonstrováno na konkrétních příkladech. Praktické ověření metodiky je provedeno na reálném projektu a porovnáno s konvenčním způsobem vývoje. V závěru jsou výsledky aplikace metodiky vyhodnoceny.

Klíčová slova

Efektivita, projektový tým, vývojový proces, webová aplikace, ETWA, metodika, projektová fáze, internetové technologie, WWW, XML kolektor, transportní vrstva.

Summary

The presented dissertation thesis focuses on the optimalization of the development process of internet applications. It is divided into two main parts. Theoretical sources focusing on specifics of internet applications, including common design and development processes, are analyzed in the first part. Activities of every project role and their participation in development process are analyzed in the thesis. Requirements for better efficiency of the whole development process are formed in accordance with results of a survey among internet agencies and project teams' members.

New methodology for effective creation of web applications is based on the acquired knowledge and the survey. Proposed methodology fulfills requirements defined in the first part of the thesis. It is focused on the development process, but it is technologically independent which makes it applicable in various target environments.

The usage of the new methodology is demonstrated on concrete examples. Practical verification is realized on a real project and it is compared to another one realized in common way. Results of the methodology application are evaluated in conclusion.

Keywords

Effectiveness, project team, development process, web application, ETWA, methodology, project phase, internet technologies, WWW, XML collector, transport layer.

Obsah

Úvod

Schopnost jednotlivce orientovat se v informačních zdrojích, efektivně v nich vyhledávat a vyhledané informace interpretovat, zasazovat do kontextu, tvořivě zpracovávat a vytvářet z nich znalosti, patří k základním charakteristikám znalostní společnosti, jejíž rozvoj je přímo ovlivněn rozvojem informačních a komunikačních technologií (ICT). Dnes již není pochyb o tom, že Internet a ICT patří k základnímu vybavení jak jednotlivců, tak podniků. Pro řadu podniků je dnes již otázkou bytí spolehlivě fungující ICT. V některých případech jsou prostředky ICT využívány jako podpůrné nástroje pro efektivní běh podniku, v jiných zajišťují hlavní aktivitu celého podniku a mají přímý vliv na jeho zisk. Pro zajištění efektivního chodu a pro elektronické obchodování se využívají různé internetové aplikace.

Pojem internetové aplikace je možné na základě výkladu různých literárních zdrojů generalizovat na úroveň základních protokolů, na kterých je Internet vybudován a označovat takto veškeré aplikace postavené na protokolu TCP/IP a architektuře klient-server. Do takto široce definované skupiny lze zahrnout i aplikace založené na protokolu Telnet, SSH, apod. Předkládaná práce se zaměřuje na omezenou skupinu internetových aplikací, které se vyznačují tím, že jsou ovládány pomocí internetového prohlížeče. Jejich hlavní výhodou je nezávislost na použitém hardwarovém a programovém vybavení na straně klienta a možnost volného přístupu z libovolného počítače, který je připojen k Internetu a je vybaven standardním prohlížečem webových stránek (tenkým klientem). V některých literárních zdrojích jsou tyto aplikace označovány jako webové aplikace.

Počet webových aplikací neustále roste. Požadavky na webové aplikace vychází z podnikatelské reality. Vývoj těchto aplikací je v mnohých případech prováděn na míru konkrétním požadavkům zadavatele, a to především proto, že software zajišťující požadovanou funkčnost na trhu neexistuje, případně jeho cena není akceptovatelná. Vytvářená aplikace má být specifická a originální. Právě poslední aspekt je typický pro webové aplikace se zaměřením na koncového zákazníka, prostřednictvím kterých je uskutečňován prodej zboží, služeb, či poskytovány služby samotné. Originalita a promyšlenost celého řešení je tedy jedním z faktorů úspěchu.

Úměrně k rostoucí poptávce po internetových aplikacích roste i nabídka poskytovatelů těchto služeb. Vznikají nové internetové agentury[1], přicházejí zahraniční společnosti, zvyšuje se konkurence, o čemž svědčí i rostoucí počet členů profesních organizací, mezi které patří například "Asociace dodavatelů internetových řešení" (Asociace.biz). Rostoucí nabídka vytváří tlak na snižování cen, což musí být promítnuto i do vlastního fungování těchto IT

[1] *Internetovou agenturou* je v předkládané práci rozuměna IT firma malé až střední velikosti, zabývající se tvorbou a poskytováním internetových řešení (převážně na klíč).

firem, projektových týmů, či alokovaných zdrojů. Návrh, ocenění, řízení a tvorba internetových projektů se potýkají s obdobnými problémy, jaké jsou identifikovány i v ostatních oblastech vývoje software.

Na mnoha odborných IT konferencích, jakou je například WebExpo, jsou často diskutovány faktory nízké efektivnosti projektových týmů, vhodná organizace pracovních úkolů v čase, či procesy řízení změn. Existuje celá řada přístupů, jak řídit a realizovat softwarový projekt. Např. Guckenheimer [11] porovnává protikladné přístupy tzv. úkolocentrický (work-down) a hodnotocentrický (value-up), z nichž každý vyžaduje jiný způsob řízení, organizace a práce projektových týmů.

Postupy při tvorbě webových aplikací vychází z metod projektování informačních systémů a projektového řízení. Vzhledem ke specifickým vlastnostem internetových technologií zde existují dílčí odlišnosti. Pro úspěšnou tvorbu webových aplikací je třeba zvolit racionální a efektivní postup, jehož návrhem se předkládaná práce zabývá.

Cíl práce a metodika

Předkládaná disertační práce se zaměřuje na optimalizaci vývojového procesu tvorby internetových, přesněji webových, aplikací. Tvorbu a celkový vývoj těchto aplikací zajišťují jak velcí výrobci klasického software (např. firmy IBM, HP, aj.), ale především malé „IT firmy", které vytvářejí zákaznicky orientované webové aplikace na klíč.

Právě malé „IT firmy" (v této práci označované jako *internetové agentury*) jsou velmi pružné při tvorbě různých internetových řešení. Vzhledem k poměrně významné konkurenci v této oblasti, ale i tlaku na maximální zkracování času dodávek je v rámci internetové agentury důležité kvalitní řízení projektového portfolia a flexibilní alokace lidských zdrojů. Velmi často je řešeno několik projektů současně a alokovaní lidé tak musí střídavě pracovat na více aktivitách. To s sebou přináší mnoho vzájemných vlivů, které je nutné správně řídit a omezovat jejich negativní působení.

Ve většině internetových agentur je pro vývoj využíváno klasického vodopádového modelu, který představuje sice velmi logický postup činností, ale zároveň s sebou přináší mnohá rozčarování a dodatečné náklady na konci projektu.

Cílem předkládané disertační práce je optimalizace standardně využívaného vývojového procesu internetových aplikací formou návrhu vlastní metodiky ETWA. Postupováno je přes identifikaci požadavků a předpokladů, až po vlastní návrh nové metodiky a představení návrhů konkrétních možností jejího použití. Cílem je optimalizovat posloupnost činností při vytváření webových aplikací s důrazem na možnost zkrácení celkové doby realizace, efektivního řízení vývoje, změn a využití lidských zdrojů. Budou prezentovány výsledky praktického použití navržené metodiky, ověření aplikovatelnosti na skutečném projektu. Výsledky budou porovnávány s projektem obdobného rozsahu realizovaného konvenčním postupem. Ve shrnutí bude metodika zhodnocena vzhledem k definovaným požadavkům a míře jejich naplnění.

Dílčí cíle a metodické postupy předkládané disertační práce lze rozdělit ze tří pohledů:

1. Oblast zkoumání

- Analyzovat teoretické zdroje zabývající se zvláštnostmi a specifiky internetových aplikací z pohledu manažerského,
- vybrat a charakterizovat technologie, jichž je v disertační práci využito pro důkazy a ověřování navržených metodických postupů,

- porovnat postupy návrhu a tvorby internetových aplikací, shrnout základní typy webových aplikací,
- vymezit terminologii projektových rolí vystupujících v celém životním cyklu internetové aplikace.

2. Návrh vlastní metodiky ETWA

- Analyzovat činnosti a identifikovat vzájemné vztahy jednotlivých projektových rolí,
- na základě spolupráce s internetovými agenturami, průzkumu mezi členy projektových týmů, provést vyhodnocení překážek v efektivnosti projektových týmů,
- definovat předpoklad cílového stavu, kterého by mělo být aplikací metodiky dosaženo, ve formě požadavků,
- navrhnout a popsat vlastní metodiku efektivní tvorby webových aplikací, včetně ověření její použitelnosti pomocí vybraných technologií,
- zhodnotit navrženou metodiku a identifikovat potřebné změny v jednotlivých fázích návrhu a vývoje internetových aplikací.

3. Praktická aplikace metodiky a závěry

- Vybrat a popsat dva reálné internetové projekty obdobné velikosti,
- ověřit aplikaci metodiky na reálném projektu,
- porovnat výsledky vůči standardně realizovanému projektu,
- formulovat závěry a doporučení.

Pro rešeršní část práce je využito knižních i internetových literárních zdrojů, uvedených v seznamu literatury. Návrh a ověření metodiky ETWA čerpají ze zkušeností tří internetových agentur, se kterými autor práce při návrhu metodiky spolupracoval. Navržená metodika je navrhována na základě teoretických poznatků a praktických zkušeností a vychází tak z reálných podkladů a situací.

Kapitola 1

Oblast zkoumání

„Co chceme, tomu také ochotně věříme, a doufáme, že si i ostatní myslí, co si myslíme sami.“
Gaius Julius Ceasar

Tato práce se věnuje návrhu, řízení a optimalizaci vývojového procesu *webových aplikací*. Protože terminologie v oblasti internetové tvorby není ustálena, a čtenář se může setkat s různým výkladem stejného termínu, bude nejprve vymezena cílová oblast této práce.

Webovou aplikací se rozumí systém pracující na základech protokolů TCP/IP a HTTP. V odborné literatuře je využíváno také označení *internetová aplikace* nebo *internetový projekt*. Všechna tato označení jsou v předkládané disertační práci chápána identicky a rozumí se jimi aplikace využívající vlastností standardního internetového prohlížeče, který vůči aplikaci vystupuje jako tenký klient a zajišťuje obsluhu celého uživatelského rozhraní a nabízí standardní služby, kterými jsou především:

- zobrazování a operace s uživatelským rozhraním dle definice značkovacího jazyka XHTML,
- formátování uživatelského výstupu na základě stylové definice, kterou taktéž podporuje vykreslovací jádro (pro stylopis využívá CSS, nebo XSL),
- obsahuje parser HTML DOM[1], resp. XML a umožňuje procházení struktury elementů,
- podporuje skriptovací jazyk JavaScript, který umožňuje spouštět skripty v kontextu prohlížeče,
- transformuje události (vstupy) provedené v prohlížeči na unikátní formát URL adresy, kterou prostřednictvím HTTP odesílá serveru k dalšímu zpracování,
- podporuje lokální uložení uživatelských dat a jejich sdílení na server (např. formou cookies)

Jak vyplývá z výše zmíněného, jsou v práci uvažovány pouze webové aplikace ovládané přes internetový prohlížeč. Obecně se jedná o aplikace nazývané jako World Wide Web, což

[1] DOM – Document Object Model. Představuje objektově orientovanou reprezentaci XML nebo HTML dokumentu.

je dodnes používané označení pro webové stránky a aplikace, reprezentované třemi „w" na začátku adresy webové stránky nebo aplikace (např. `www.wikipedia.org`) a z toho plynoucí obecné označení *web*. Díky rapidnímu vývoji technologií se od dob vzniku myšlenky World Wide Webu, v roce 1989, způsob využití webu podstatně změnil, než jak ho původně Tim Berners-Lee [2] zamýšlel.

Můžeme se stále setkávat s označením *statické* a *dynamické* webové stránky. *Statickým* bývá označován typ stránek, který původně navrhl Berners-Lee, tedy sada hypertextových souborů, vzájemně propojených odkazy. Zde je nutné poznamenat, že první definice hypertextu přichází již v roce 1965 od Teda Nelsona. Myšlenka *odkazů* uvnitř vlastního textu, umožnila významové propojení značného množství hypertextových dokumentů. Důležité je právě hledisko *souborů*, kdy by se s určitým zjednodušením dala URL adresa definovat jako označení umístění (cesta) konkrétního HTML *souboru*, na určeném serveru, jeho adresáři, v rámci sítě Internet. Nedá se říci, že by takovéto stránky v současné době neexistovaly, ale zcela jistě tvoří minoritní část obsahu současného webu. Trendem jsou právě stránky *dynamické*, označované jako portály, webové aplikace, skriptované stránky, generované stránky, apod. Na serveru již nejsou umístěny dokumenty, které jsou na základě URL požadavku zobrazeny v prohlížeči, ale je zde programový kód, který HTML stránky generuje na základě požadavků uživatele. Požadavky jsou formovány právě prostřednictvím URL, která v současné době spíše označuje akci, která se bude provádět, její parametry a co je očekáváno jako výsledek (např. `http://www.priklad.cz/clanek/prehled-vysokych-skol/`). Změna orientace na dynamické technologie je i v případech relativně jednoduchých prezentací, které nevyžadují výrazně složitou interakci s uživatelem, dána snadností údržby a změn na takto vytvořených stránkách. Dynamické technologie umožňují skládání stránky z opakujících se elementů, obsluhu jednoduchých formulářů, efektivní SEO optimalizaci, apod.

Změna orientace z konvenčního na webové ovládání je patrné u mnoha výrobců software. Výhodou je zejména centralizovaná správa jedné verze aplikace, možnost její rychlé opravy nebo úpravy na jednom místě pro všechny uživatele, či instance najednou.

Tato práce se orientuje především na komplexnější aplikace, nikoliv na tvorbu webových prezentací firem, kterými jsou rozuměny elektronické obchody, elektronické služby, intranetové, nebo informační systémy. Vývoj tohoto typu aplikace vyžaduje účast většího počtu specialistů, obdobně jako vývoj klasického softwarového nástroje.

1.1 Problémy řízení webových projektů

V oblasti webového vývoje je velice často kladen důraz na kvalitu finálního produktu z pohledu uživatelského rozhraní a jeho úspěšnost na internetu. Ovšem toto kritériím je spíše hodnocením obchodního přínosu projektu a celého záměru, nikoliv měřítko úspěšnosti realizace webového projektu. U tohoto typu projektů je možné shrnout charakteristiky úspěšné realizace a jejího řízení do následujících bodů:

- projekt je dokončen dle časového harmonogramu v dohodnutém termínu,
- jakékoliv změny jsou evidovány a odsouhlaseny vzájemně se zadavatelem,

- jestliže dochází ke změně rozsahu projektu během realizace, jsou pro tyto změny připraveny objednávky a jsou účtovány separátně.

Výše uvedené body představují ideální průběh projektu, protože během realizace dochází k odchylkám jak na straně zadavatele, tak na straně realizátora. Vedením projektu na straně realizátora je pověřen projektový manažer. V následujících oddílech jsou shrnuty nejčastější problémy řízení webových projektů.

1.1.1 Vzájemná komunikace

Existují různé typy zadavatelů. Někteří se rádi zapojují do mikro-managementu projektu a zajímají se o každý detail přípravy, na druhé straně jsou klienti, kteří realizaci pozorují spíše z povzdálí a nechávají vše na realizátorovi. Ani jeden z těchto extrémů není pro projekt užitečný. Existuje množství metod, jak donutit zadavatele, aby efektivně komunikoval s projektovým manažerem. Na počátku projektu by měla být odsouhlasena a připravena forma komunikace mezi zadavatelem a projektovým manažerem na straně realizátora. K tomuto účelu je možné využít např. webových nástrojů, což je velice příznačné při tvorbě webových projektů. Příkladem dobře fungujícího webového nástroje pro tento účel je oblíbený Basecamp (`http://www.basecamp.org`), který pro zadavatele představuje pohodlný nástroj pro komunikaci s projektovým manažerem nebo dokonce s klíčovými účastníky projektu. Má tak vždy přehled o termínech, aktuálním stavu projektu, naplňování schválených cílů, apod. Pro klienty, kteří nepreferují tento způsob komunikace, případně nejsou jakýmkoliv způsobem připraveni komunikovat prostřednictvím elektronických nástrojů, je třeba dohodnout *pravidelné osobní schůzky*. Ty musí být dobře připraveny a jejich předmětem je zodpovězení klíčových otázek v projektu, nebo řešení problémů, které se aktuálně v projektu vyskytly. Efektivní komunikací je možné docílit plynulého průběhu projektu a eliminovat tak neočekávaná překvapení.

1.1.2 Spolupráce se zadavatelem

Největší odpovědností projektového manažera je komunikace uvnitř vývojového týmu, který je pro účely projektu nominován. Je odpovědný za vytvoření realistického plánu, který je postupně naplňován s cílem dokončit práce ve stanoveném čase. Aby bylo možné časový plán korigovat a včas odhalit odchylky, jsou do plánu zaneseny milníky, tedy termíny, ve kterých mají být dokončeny určité části dodávky. Milníky jsou tedy ve většině případů svázány s konkrétní částí dodávky. Některé milníky, jako je například dodání obsahu, definice algoritmů výpočtu, specifikace vstupních hodnot, jsou zodpovědností zadavatele.

Webové studio ve většině případů pracuje (tedy i členové jednotlivých projektových týmů) na více projektech najednou. Čas jednotlivých týmů, či jednotlivců, musí být naplánován na konkrétní úkoly (projekty) v konkrétním časovém období tak, aby všechny dílčí milníky mohly být dodrženy a zdroje byly efektivně alokovány. Jestliže se prodlouží milník některého projektu kvůli zdržení na straně zadavatele, promítají se důsledky na plán týmu, který měl pracovat na navazujících částech. Proto tedy nelze zdržení na straně zadavatele promítnout pouze do prodloužení doby trvání projektu o délku tohoto zpoždění, ale je třeba dopad zhodnotit na úrovni organizace práce týmů a souvislostí na úrovni ostatních projektů.

1.1.3 Zadavatel změní své požadavky

Změna rozsahu (scope) je jedním z nejhůře plánovatelných aspektů vývoje nejen webových aplikací. Dochází k němu v případě, kdy požadovaná funkčnost splní požadavky definované specifikací a projektovým plánem, ale zadavatelova představa je ve skutečnosti jiná. Změny v zadání jsou prováděny velice často během probíhajícího vývojového procesu. V této souvislosti je na místě zmínit používaný anglický akronym *IKIWISI* z anglického výrazu *I'll Know It When I See It*. Je často používán v souvislosti s řízením a přípravou webových projektů a popisuje chování zadavatele, který nedokáže odsouhlasit zadání připravené analytiky do té doby, než uvidí funkční aplikaci nebo prototyp. Případně opačně, kdy zadavatel zadání odsouhlasí, ale v okamžiku prezentace výsledku zjišťuje, co měl být skutečný záměr.

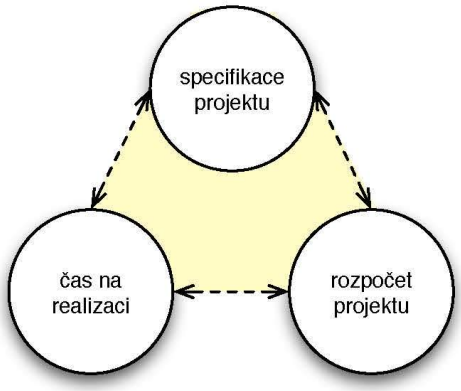
Pro úspěšné řízení projektu je nutné ke každé změně vést evidenci a podle její důležitosti jednotlivě nebo skupinově připravit dokument změnového požadavku (Change Order, Change Request), který je následně přílohou k původní specifikaci. Samozřejmostí každé změny musí být její ohodnocení a stanovení finančních a časových důsledků.

Projekt jako celek je možné vymezit na základě tzv. *projektového trojimperativu* (viz obr. 1.1), který tvoří tři základní vzájemně propojené parametry projektu: specifikace projektu, čas na realizaci, rozpočet projektu [41]. Tyto tři parametry jednoznačně definují projekt z hlediska jeho výsledku v rovině kvalitativní – rozsah projektu (specifikace produktu, popis díla); v rovině časové – harmonogram (termíny); v rovině finanční – náklady (náklady na práci a potřebné zdroje). Podstatou trojimperativu je vzájemná závislost jednotlivých parametrů, kterou je možné pozorovat ve dvou rovinách. Jednak každý projekt má určité minimální hodnoty všech parametrů, a tím stanovenou minimální „plochu“ trojimperativu. Určité kombinace parametrů rozsahu, harmonogramu a nákladů jsou invalidní a na jejich základě není možné projekt realizovat. Dalším projevem vzájemné závislosti je chování trojimperativu při změně hodnoty kteréhokoliv z jeho parametrů. Změna jednoho parametru automaticky vyvolá změny ostatních dvou. Například po snížení rozpočtu projektu by měly následovat zásahy do rozsahu (ve smyslu zjednodušení), což povede ke změně harmonogramu projektu. Je to právě vzájemná závislost parametrů trojimperativu, která je často při dojednání smluvních podmínek a změnových řízení bagatelizována nebo opomíjena. Taková situace může vést ke snížení hodnoty projektu ať ze strany zákazníka či dodavatele [41]. Časté změny v projektu mohou být eliminovány například [22]:

- přípravou projektového plánu založeného na studiu požadavků klienta před zahájením etapy vývoje,
- spoluprací a tvorbou návrhu a plánu aplikace spolu s klientem, z něhož bude sestaven návrh struktury a obsahu aplikace, navigace a ovládání

1.1.4 Projektové milníky nejsou dodrženy

Jestliže na projektu pracuje zkušený tým a nevyskytnou se žádná zpoždění na straně zadavatele, a i přesto se nedaří naplnit plánované cíle, je třeba hledat problém ve schváleném plánu. Projektový manažer musí při sestavování plánů respektovat rozvržení práce jednotlivých členů týmu. Rozvržení času musí být konzultováno buď s ostatními projektovými



Obrázek 1.1: Projektový trojimperativ

manažery, kteří využívají stejný zdroj, aby byla alokace daného člověka naplánována realisticky. Rozvržení práce na projektech neslouží pouze pro potřeby projektového manažera a zadavatele, ale také pro členy projektového týmu, kterým určuje rozdělení práce na jednotlivé dny. Členové projektového týmu zpětně informují o množství odvedené práce na dané úloze a procentním splnění úkolu.

1.2 Technologie internetových aplikací

V tomto oddíle jsou představeny technologie využité při návrhu vlastní metodiky. Tyto tvoří buď přímou součást návrhu, nebo jsou použity pro účely důkazů a příkladů konkrétní implementace. Je proto důležité, aby byl čtenář předem seznámen s bližšími souvislostmi. Naopak se předpokládá znalost základních jazyků, protokolů a technologií, jako jsou HMTL, CSS, JavaScript, HTTP, URL, MVC, apod. Jejich představení je nad rámec této práce.

1.2.1 XML (eXtensible Markup Language)

XML je podmnožinou jazyka SGML[2]. Podle Koska [23] jde o zjednodušenou verzi, která obsahuje ze SGML to nejdůležitější. Je v něm možné definovat vlastní značky (tagy) a pak pomocí nich vytvářeny dokumenty a zpřístupňované ostatním pomocí webu. Pro pochopení přínosu XML Kosek [23] dále uvádí, že SGML je normou ISO (číslo 8879 z roku 1996). Nutno podotknout, že normou velice obsáhlou. V praxi to znamená, že se většinou nevyužívají zdaleka všechny možnosti SGML. Programy, které mají umožnit práci se SGML dokumenty, však musí odpovídat normě a proto jsou poměrně složité a jejich vývoj je nákladný a zdlouhavý. Právě tento problém řeší XML - obsahuje vše potřebné a přitom je dostatečně jednoduchý, aby šly poměrně snadno psát aplikace pracující nad XML dokumenty. Díky tomu, že XML je podmnožinou SGML, lze pro práci s XML dokumenty využít všechny stávající SGML aplikace.

[2]SGML = **S**tandard **G**eneralized **M**arkup **L**anguage. SGML je metajazyk, který umožňuje standardizovaným způsobem definovat další značkovací jazyky.

1.2.1.1 Příčiny vzniku XML

Ačkoliv se vznik XML mohl podle Koska [23] v době svého vzniku zdát jako poněkud neočekávaný, pro zasvěceného to bylo jen logické vyústění překotného vývoje na poli informačních a komunikačních technologií. Obrovské množství informací, kterými lidstvo disponuje, přináší mnoho problémů při jejich zpracování. Jedním z těchto problémů je sdílení informací po celém světě. Tento problém se poměrně úspěšně podařilo vyřešit pomocí Internetu a jeho služby WWW. Informace uložené na webu jsou přístupné všem a vůbec nezávisí na tom, jaký používají počítač a operační systém.

V dokumentech založených na XML můžeme snadno vyhledávat odpovědi na dotazy typu: „Které básně napsal pan X?“. Při použití HTML můžeme klást pouze dotaz typu: „Které stránky obsahují slovo X?“. Výhody prvního způsobu dotazování v dnešní, informacemi přehlcené, době netřeba zdůrazňovat.

Samozřejmě, že předchozí výhody je možné využívat pouze v případech, kdy prohledávané dokumenty budou vyhovovat stejnému DTD[3] (budou používat stejnou množinu značek). Již dnes existují DTD, které používají určité skupiny uživatelů a v budoucnosti patrně vzniknou ještě další[4]. Kosek [23] již v roce 1999 očekával, že své DTD tak budou používat matematici, historici, chemici, elektrotechnici, apod. V dnešní době jsou již DTD, ne-li pokročilejší definice XML dokumentů, předmětem standardizace.

Jazyk XML byl vytvořen proto, aby uspokojil stále vzrůstající nároky na kvalitu a kvantitu zpracovávání informací. Díky tomu, že v XML lze definovat téměř jakkoliv složitou strukturu, je možné vytvořit DTD, která budou používaná pro výměnu dat např. mezi lékaři a zdravotními pojišťovnami, mezi výrobci elektrotechnických součástek atd. XML poslouží jako univerzální, na platformě nezávislý, formát pro výměnu informací.

Vývoj XML a příbuzných technologií zajišťuje konsorcium W3C. To tvoří *pracovní skupiny*, které se skládají ze zástupců komerčních organizací, průmyslu, i akademických odborníků. Pracovní skupiny vytváří tzv. *doporučení*, která jsou *de facto* standardy pro webové technologie. Při tvorbě doporučení pro XML, byly cílem návrhu definovány následující body [1]:

- XML by mělo být ihned použitelné na Internetu,
- XML by měla podporovat široká škála aplikací,
- XML by mělo být kompatibilní se SGML,
- mělo by být jednoduché psát programy pro zpracování XML dokumentů,
- množství volitelných rozšíření XML by mělo být drženo na absolutním minimu, ideálně na nule,
- XML dokumenty by měly být čitelné člověkem a přiměřeně srozumitelné,
- návrh XML by měl být připraven rychle,

[3] DTD **D**ocument **T**ype **D**efinition

[4] Příkladem takového DTD může být například DTD NITF (News Industry Text Format) <`http://www.nitf.org`>, který pro strukturalizaci uložených textů využívá například projekt ČZU v Praze - Agrární www portál Agris <`http://www.agris.cz`>.

- ▷ návrh XML by měl být formální a stručný,
- ▷ XML dokumenty by měly být lehce vytvořitelné,
- ▷ stručnost XML zápisu má minimální důležitost.

Není cílem rozebírat původní body návrhu doporučení, ale již dnes si můžeme odpovědět, jak byl původní návrh daleko od současné reality.

1.2.1.2 Vlastnosti XML

Výhody jazyka XML přehledně shrnují Harold [12] a Nghiem [36], včetně jejich krátkého vysvětlení.

Jednoduchost Informace uložené ve formátu XML je možné lehce číst, porozumět jim a navíc mohou být jednodušeji zpracovávány počítači.

Otevřenost XML představuje standard dle konsorcia W3C[5], podporovaný vedoucími firmami na softwarovém trhu.

Rozšiřitelnost U XML není nadefinována pevná sada značek (tagů). Mohou být vytvářeny nové značky, a to na základě zjištěných potřeb subjektů, aplikací, apod.

Vlastnost sebepopisnosti U tradičních databázových systémů vyžadují datové záznamy pevné schéma vytvořené administrátorem databáze. Dokumenty XML mohou být ukládány i bez takovéto definice, protože mohou obsahovat metadata ve formě značek (tagů) a jejich atributů. XML také poskytuje prostředky pro identifikaci autora a verzování na úrovni jednotlivých elementů. Jakákoliv XML značka (tag) může obsahovat neomezené množství atributů, jako např. autora nebo verzi.

Hierarchie Jakýkoliv element může v XML obsahovat neomezené množství podřízených elementů, pro které platí identické pravidlo. Tímto způsobem je možné naprosto neomezeně modelovat objekty reálného světa a i průběžně je doplňovat o podrobnější rozpad informací, což je těžko představitelné u klasických databází.

Modularita Možnost odkazování XML dokumentů na jiné dokumenty podporuje modulární návrh systémů a také znovupoužití existujících částí.

Obsahuje strojově čitelné kontextové informace Značky, jejich atributy a struktura elementů poskytuje kontextovou informaci, která může být použita k interpretaci smyslu obsahu, otevírá nové možnosti pro velice efektivní vyhledávání, inteligentní získávání informací (data mining), apod. Toto je hlavní výhodou oproti HTML nebo prostému textu, kde je vyhodnocení kontextových informací složité, či zcela nemožné.

[5] World Wide Web Consortium <`http://www.w3.org`>

Obsah oddělený od prezentace XML značky narozdíl od značek HTML popisují svůj obsah, nikoliv jeho prezentaci. Mottem HTML je: „Vím jak to vypadá“, naproti tomu motto XML by se dalo formulovat jako „Vím, co to znamená a vy řekněte, jak to má vypadat.“. Vzhled a prezentace XML dokumentů může být zajištěna přes pravidla stylů XSL, která umožňují změnit vzhled dokumentu bez nutnosti zásahu do jeho obsahu. Tímto způsobem je možné realizovat také různé pohledy nebo prezentace stejného obsahu.

Podpora vícejazyčných dokumentů a Unicode Představuje velice důležitou vlastnost pro přizpůsobování aplikací pro různé jazyky a národnosti. Kosek [23] poukazuje na skutečnost, že XML již od začátku řeší podporu různých jazyků. Jako standardní znaková sada je používáno ISO 10646, což je 32bitové kódování, které obsahuje všechny dnes používané znaky. V XML je i podpora pro text psaný zprava doleva či shora dolů. V současné době je již zcela nepředstavitelné, aby jazyk pro uložení dat nepodporoval specifické znaky a diakritiku národních abeced pro uložení dat jako jsou jméno, adresa, apod.

Podpora porovnávacích a agregačních funkcí nad daty Stromová struktura XML dokumentu umožňuje efektivní porovnávání a agregaci dokumentů element po elementu.

Umožňuje vkládání různých datových typů XML dokument může obsahovat jakýkoliv datový typ - od prostého textu, přes multimediální data (obraz, zvuk, video) až po aktivní komponenty (Java applety, ActiveX).

Umožňuje vkládání existujících dat Mapování existujících datových struktur do XML, jako například souborových systémů nebo relačních databází je velmi jednoduché. XML podporuje různé datové formáty a pokrývá tak všechny existující databázové struktury.

Poskytuje „jednoserverový pohled“ na distribuovaná data Dokumenty XML mohou obsahovat tzv. „zahnízděné“, resp. vložené/vnitřní elementy, které jsou distribuovány přes více vzdálených serverů. XML je v současné době nejsofistikovanější nástroj pro distribuovaná data - World Wide Web může být chápán jako ohromná XML databáze.

Rychlé přizpůsobení softwarového průmyslu Velké korporace, jako je IBM, HP, Sun, Microsoft, AOL, SAP a mnoho dalších, již nativně podporují XML. Microsoft začal používat XML jako výměnný formát pro svoji produktovou linii Office[6]. Všechny tři vedoucí internetové prohlížeče (Microsoft Internet Explorer, Mozilla Firefox i Opera) v současné době podporují zobrazování a práci s XML formátem. SAP podporuje XML prostřednictvím SAP Business Connector v kombinaci s R/3, na XML jsou postavena standardní aplikační rozhraní (API, WS), apod.

[6] Pozn.: XML jako svůj nativní formát v současné době využívá například konkurenční OpenOffice <`http://www.openoffice.org`>.

```
<?xml version="1.0" encoding="windows-1250"?>
<osoby>
   <zamestnanec>
      <jmeno>
         <krestni>Josef</krestni>
         <prijmeni>Novák</prijmeni>
      </jmeno>
      <pracoviste>Výroba 007</pracoviste>
      <adresa>
         <ulice>Hradecká</ulice>
         <cp>603</cp>
         <mesto>Lovosice</mesto>
         <psc>410 02</psc>
      </adresa>
      <popis>Výrobní vedoucí.</popis>
      <fotografie soubor="novak.jpg" />
   </zamestnanec>
   <zamestnanec>
      <jmeno>
         <krestni>Jan</krestni>
         <prijmeni>Sova</prijmeni>
      </jmeno>
      <pracoviste>Expedice</pracoviste>
      <fotografie soubor="sova.jpg" />
   </zamestnanec>
</osoby>
```

Obrázek 1.2: Ukázka XML dokumentu

1.2.1.3 Princip XML

Princip XML, jakožto značkovacího jazyka, spočívá podle Talicha [45] zjednodušeně řečeno v tom, že značky obepínající v textovém XML dokumentu určité části textu, označují jejich věcný význam. Které značky a v jaké struktuře je možné použít, si autor stanoví sám v Definici Typu Dokumentu - DTD (Document Type Definition) a způsob, jak se má dokument zobrazit, si určí pomocí stylů výsledného vzhledu XSL (eXtensible Stylesheet Language). Odlišně od HTML neurčují značky způsob prezentace označené části textu, ale její věcný význam. Například element `<b>284 890 515</b>` označuje v HTML textu jakési číslo, které má být zobrazeno tučně, více o něm není známo. Ovšem element `<telefon>284 890 515</telefon>` v XML dokumentu již označuje telefonní číslo (ne tedy obecně jakési číslo) a způsob jeho prezentace může být nezávisle na tomto dokumentu stanoven v externím stylovém souboru (například tučné vysázení). Přitom skutečnost, že tato konkrétní značka vůbec existuje, že je ji možno použít právě na tomto místě XML dokumentu, a že může obsahovat číslo, může být opět nezávisle na tomto dokumentu stanoveno např. v externím souboru DTD. Toto je další důležitou vlastností dokumentů XML - kontrola dokumentu podle stanovených pravidel, přičemž jedna pravidla (např. jeden DTD soubor) mohou být současně použita pro neomezený počet XML dokumentů, které je mají splňovat.

Podle ukázkového dokumentu na obrázku 1.2 můžeme demonstrovat základní části XML dokumentu a uvést jejich oficiální označení [1]:

- *element* je jakákoliv část dokumentu uzavřená do *značek* stejného názvu (v ukázce jsou např. elementy `zamestnanec`, `jmeno` nebo `ulice`. Zkrácené formy uzavření elementu si můžeme všimnout u elementu `fotografie`),
- *atribut* rozšiřuje a popisuje vlastnosti elementu; skládá se z názvu atributu a jeho hodnoty a je součástí uvozující značky elementu (v ukázce má např. element `fotografie` atribut `soubor` s různými hodnotami v závislosti na osobě),
- *zahnízdění* vyjadřuje vztah zahnízděných elementů k jejich nadřazenému elementu - vyjadřuje kontext (v ukázce jsou elementy `krestni` a `prijmeni` zahnízděny do elementu `jmeno`, `zamestnanec` je zahnízděn do elementu `osoby`).

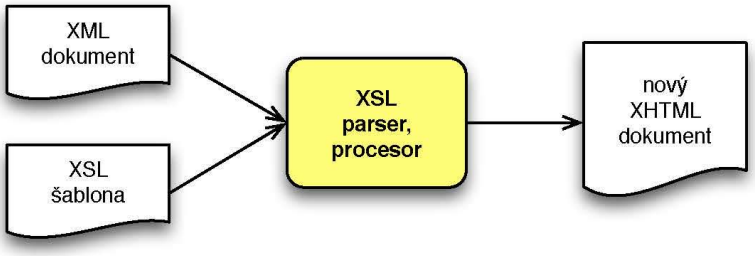
Vzorový XML dokument na obrázku 1.2 demonstruje několik pravidel XML:

- pro jakoukoli část dat v souboru lze vytvořit element, kterým je část označena,
- elementy do sebe lze libovolně vnořovat,
- musí se zachovat správné zahnízdění elementů, aby byla zachována logika vnoření, tj. elementy se nesmí překrývat. HTML podporovalo, resp. analyzátory tohoto jazyka, zápis typu `<b><i>toto je tucna kurziva</b></i>`, naproti tomu pravidla XML takovéto konstrukce neumožňují. Chyba tkví v tom, že element `<i>` je zahnízděn do elementu `<b>`, a proto by měla značka `</i>` být uzavřena před uzavřením značky `</b>`. Korektní zápis podle pravidel XML by měl vypadat následovně `<b><i>toto je tucna kurziva</i></b>`,
- veškeré argumenty (hodnoty atributů) musí být uzavřeny do uvozovek,
- veškeré elementy (značky) musí být uzavřeny (buď prostřednictvím koncového elementu, nebo pomocí zkrácené formy - viz element `fotografie` v ukázce na obr. 1.2),
- speciální znaky musí být reprezentovány ve formě entit, aby nemohlo dojít k jejich špatné interpretaci (zejména znaky „<“, „>“, apod.),
- aby mohlo být s XML dokumentem pracováno musí být *validní*, tj. musí splňovat výše uvedená pravidla, včetně dalších specifikovaných konsorciem W3C[7].

1.2.1.4 Transformace prostřednictvím XSL procesorů

Transformace XML dokumentů do výsledných formátů je možná například pomocí jazyka XSL[8]. Aby bylo možné transformaci dat z XML provést je podle Batese [1] nutné, aby na zpracovávající straně byl tzv. XSL procesor, jehož funkcí je pomocí XSL a XML souborů (vstupů) vytvořit výstupní soubor např. ve formátu HTML. XSL procesor tak pomocí XSL formátovacího souboru transformuje XML data na jiný formát (přesněji řečeno jazyk podle specifikace XSLT). Schéma zpracování souborů pomocí XSL procesoru lze znázornit způsobem uvedeným na obrázku 1.3.

[7] http://www.w3.org/XML/
[8] XSL = e**X**tensible **S**tylesheet **L**anguage



Obrázek 1.3: Schéma zpracování XML dokumentu pomocí XSL šablony (stylesheetu) [1].

XSL také nabízí možnosti pro zpracování vysoce neuspořádaných a rekurzivních dat, jako jsou například klasické textové dokumenty. V tomto případě jsou definovány fragmenty šablon a XSL procesor tyto fragmenty slučuje a vytváří z nich výsledný strom založený na datech ze zdrojového souboru. Každá šablona deklaruje typ, kontext, a pro která zdrojová data má být šablona aplikována, což XSL procesoru dovoluje vyhledávat shodu mezi zdrojovými datovými uzly a fragmenty šablon podle několika kritérií. V jednom XSL souboru mohou být také kombinovány oba přístupy: jak šablonami tak i daty řízená transformace, aby bylo možné dosáhnout široké variability výsledného kódu, který by pak mohl být použitelný na mnoha cílových platformách a zařízeních. XSL soubor aplikuje své šablony na elementy obsažené v XML souboru, pokud se shoduje uvedená specifikace elementu se vzorem tzv. patternem, který je definován v šabloně (template). Jak dále uvádí Bates [1] je vysoce výhodné využívat transformace na základě XSLT, protože ta je nezávislá na aktuálně používaném programovacím jazyce. V budoucnu je tak možné přejít např. z Java na .NET framework a pokud nebudou XSLT transformace doplňovány o zvláštní funkce, je možné je využít i v novém prostředí s minimálním úsilím.

1.2.2 Skriptovací jazyk PHP

PHP je stále velmi populární programovací jazyk [46] s otevřeným kódem (open-source), který je nejčastěji využíván pro tvorbu serverových aplikací a dynamického obsahu internetových stránek, a v poslední době také i pro vývoj ostatního software. Původně zkratka PHP znamenala „Personal Home Page“. Dnes se používá oficiální akronym „PHP Hypertext Preprocessor“.

Jednoduchost a podobnost PHP s běžnými strukturovanými programovacími jazyky (jmenovitě C a Perl) a částečně i objektově orientovanými jazyky (PHP 5 s jazykem Java) umožňuje zkušeným programátorům začít v něm vyvíjet komplexní aplikace s minimální nutností se učit něco nového. Pro tyto uživatele je také výhodné, že mohou začít vyvíjet internetové aplikace bez toho, aniž by se museli naučit celou novou sadu funkcí, příkazů a postupů.

Nejatraktivnější je na PHP skutečnost, že se nejedná pouze o skriptovací programovací jazyk. Díky modulárnímu designu umožňuje vytvářet aplikace s uživatelským rozhraním (s využitím PHP-GTK) a může být použito také přímo z příkazové řádky jako například Perl nebo Python.

PHP umožňuje kooperaci s množstvím relačních databázových systémů (např. MySQL, Oracle, IBM DB2, Microsoft SQL Server, PostgreSQL a SQLite) za použití jednoduché syntaxe. PHP je možné provozovat na hlavních typech operačních systémů zahrnujíce Unix, Linux, Windows a MacOS X, spolupracuje s mnoha webovými servery. Architektura postavená na operačním systému Linux, webovém serveru Apache, databázovém serveru MySQL a PHP (označovaná jako „LAMP“[9]) se stala velmi populární ve webovém průmyslu pro vývoj finančně nenáročných, spolehlivých, rozšiřitelných a bezpečných internetových aplikací.

Vývoje PHP se účastní mnoho spolupracujících subjektů. Je šířeno pod licencí „PHP license“, která vychází z licence BSD. PHP je od verze 4 provozováno na jádru označovaném jako Zend Engine.

1.2.2.1 Historie

Původní PHP bylo navrženo jako malá sada skriptů v jazyce Perl, provozovaná na přepsané sadě binárních částí CGI v jazyce C, vytvořené dánsko-kanadským programátorem Rasmusem Lerdorfem v roce 1994. Tento software byl vytvořen za účelem zobrazování provozu na domovské stránce autora. *Personal Home Page Tools* byly poprvé veřejně představeny 8. června 1995, poté co je Lerdorf zkombinoval s interpretem formulářů PHP/FI. Dva izraelští programátoři, Zeev Suraski a Andi Gutmans z Technionu - Israelského institutu pro technologie, přepsali v roce 1997 původní parser a vytvořili základ pro PHP verze 3. Veřejné testování PHP 3 začalo v červnu 1998. Následoval další přepis jádra, který vyústil ve vytvoření Zend Engine v roce 1999. V Izraeli byla také založena společnost Zend Technology (`http://www.zend.com`), která dohlížela na rozvoj PHP. V květnu roku 2000 bylo dokončeno PHP 4, běžící na jádru Zend Engine 1.0. Jádro Zend Engine II bylo použito do další verze PHP 5, které bylo dokončeno 13. července 2004 a obsahovalo mnoho očekávaných vylepšení. Aktuálně poslední vydanou verzí je PHP 5.2.8.

1.2.2.2 Výhody

Jedním z hlavních důvodů, který dopomohl PHP stát se velmi oblíbeným, je poměrně volný, dynamicky tvořený jazyk. Pravidla jazyka nejsou striktní k proměnným, které nemusí být deklarovány a mohou obsahovat jakýkoliv typ objektu. Pole jsou heterogenní ve smyslu, že jedno pole může obsahovat objekty více jak jednoho typu.

Na základě průzkumu společnosti Netcraft [33] provedené v dubnu 2002 bylo PHP nejnasazovanějším serverovým skriptovacím jazykem, běžícím na přibližně 9 milionech z celkových 37 milionů domén zahrnutých do průzkumu. Toto potvrzují také vlastní ukazatele PHP, které potvrzují 5% měsíční nárůst (měřeno podle počtu domén), který se zastavil na konci roku 2005 a od té doby víceméně stagnuje. V září 2005 bylo PHP provozováno na 22 167 075 doménách, v roce 2008 na cca 21 000 000, přičemž se vychází opět z průzkumu Netcraft [33, 35].

Díky popularitě PHP na webu vznikla nová skupina programátorů, kteří jsou navyklí pouze na PHP. To napomohlo vývoji rozhraní PHP pro příkazovou řádku, stejně tak jako knihoven grafického rozhraní (např. GTK+) a textového módu (Ncurses and Newt). Toto

[9] písmeno „P“ v označení *LAMP* může také v některých případech označovat Perl nebo Python

byl důležitý krok pro PHP, který jej posunul z jazyka určeného pouze pro CGI do role programovacího jazyka pro obecné účely. U klasických počítačů je oblíbené u některých programátorů zejména jako prostředí pro rychlý vývoj prototypových řešení.

1.2.2.3 Nevýhody

Množství nevýhod jazyka PHP vychází především z udržování jeho zpětné kompatibility, podpory začínajících programátorů, velké modularity a rozšiřitelnosti o vlastní komponenty. Kritické body jazyka je možné zařadit do následujících kategorií:

Syntaxe

- PHP nevyžaduje deklaraci proměnných a i s neinicializovanými proměnnými je možné pracovat. Nedeklarovaná proměnná vyvolá pouze chybu typu `E_NOTICE`, jejíž zobrazování je standardně zakázáno. Toto vede k bezpečnostním rizikům v souvislosti s registrováním globálních proměnných (viz. dále),
- některé vestavěné funkce nejsou konzistentní v předávání parametrů, zejména v jejich pořadí (např. některé funkce pro vyhledávání v polích vyžadují jako první parametr pole a některé hledaný element).

Vestavěné funkce

- jména vestavěných funkcí nemají žádnou standardizovanou formu - některé funkce používají podtržítka, jiné ne (např. `strip_tags` vs. `stripslashes`), některé obsahují první sloveso a pak podstatné jméno, jiné naopak,
- výstup některých funkcí je nekonzistentní - v dokumentaci jsou výstupy popsány jako `true` nebo `false`, ale funkce může vracet také nebitovou hodnotu, reprezentující například `false` (např. `0`, nebo „“),
- množství obsažených funkcí je velké, přičemž některé vykonávají stejné akce, maximálně s malou odchylkou,
- více jak 3 000 funkcí sdílí stejný globální jmenný prostor - množství funkcí není vestavěno, ale začnou být dostupné až po připojení požadované knihovny - k odlišení je používán prefix u jména funkce, reprezentující jméno knihovny,
- vlastnost „magic quotes“ přidává zpětná lomítka do uživatelem zadaných textů. Tato vlastnost byla představena kvůli eliminaci bezpečnostních rizik (SQL injection) u začínajících programátorů, ale je kritizována z důvodu špatného zobrazování textů a problémy s migrací aplikací mezi různě nastavenými servery, na druhou stranu systém obsahuje funkci pro kontrolu aktuálního nastavení této vlastnosti.

Bezpečnost

- jestliže je parametr `register_globals` v konfiguračním souboru PHP povolen, mapují se uživatelem zaslané proměnné metodami POST, GET a ze SESSION automaticky na systémové proměnné. Tím je možné u nedeklarovaných proměnných změnit chování aplikace přes uživatelský vstup,
- ostatní programovací jazyky jako např. ASP.NET obsahují oproti PHP funkcionalitu zamezující cross-site scriptingu,
- ve většině případů běží PHP na Unixu a Linuxu pod uživatelem `nobody`, což poměrně zhoršuje možnosti bezpečnostní politiky u hostingových serverů.

Různé

- chybová hlášení jsou matoucí, což není kritizováno pouze u PHP,
- mnoho nastavení uvnitř konfiguračního souboru `php.ini` předurčuje, že daný skript bude provozuschopný pouze na daném serveru, ale po přenesení na jiný server být funkční nemusí (`register_globals`, `magic_quotes`, apod.),
- některá rozšíření používají knihovny, které nejsou „threadsafe“ (spolehlivé při vláknovém zpracování), což může v kombinaci např. se serverem Apache 2 Multithreaded MPM způsobit řadu problémů.

1.2.2.4 Objektově orientované programování

Až do verze 3 PHP nepodporovalo objektově orientované techniky. Stejné vlastnosti byly přeneseny i do verze 4, kde bylo u objektů přidáno předávání pomocí odkazu a navrácení výsledků odkazem, ale implementace stále postrádala výkonné a užitečné vlastnosti, kterými disponují ostatní objektově orientované jazyky jako Java, Smalltalk, nebo jazyky .NET a rozhodně postrádala některé klíčové vlastnosti OOP.

Ve verzi 5, která byla představena v červenci 2004, byl objektový přístup zlepšen a nyní je více robustní a obsahuje všechny klíčové vlastnosti OOP. Následuje výčet hlavních změn v oblasti OOP u PHP 5 [3]:

Nový objektový model Manipulace s objekty byla kompletně přepsána, aby byl zajištěn lepší výkon a více možností. V původních verzích bylo s objekty manipulováno jako se základními typy dat (např. integer, string, apod.) a celý objekt byl, v případě přiřazení nebo předání, zkopírován. V novém přístupu je s objekty manipulováno na základě „handlerů“, které je možné označit jako identifikátor objektu.

Typy Private a Protected PHP 5 představilo členské proměnné a metody typu Private a Protected, což umožňuje definovat dostupnost třídních proměnných a metod.

Abstraktní třídy a metody Abstraktní metody deklarují pouze označení metody, ale neposkytují její implementaci. Třída, která obsahuje abstraktní metody musí být také deklarována jako abstraktní. Abstraktní deklarace samozřejmě nemá smysl pro vlastnosti.

Rozhraní (Interfaces) Pomocí abstraktních metod lze takto deklarovat metody třídy, tedy jako metody bez konkrétní implementace. Tímto způsobem může jeden vývojář stanovit rozhraní pro další odvozené třídy, jež jsou nuceny metodu implementovat, jinak ji v odvozených třídách, respektive instancovaných objektech, nelze použít.

Klonování objektů Jak již bylo zmíněno výše, PHP 5 odbouralo kopírování objektů a ty jsou místo toho předávány odkazem. Někdy však může nastat situace, kdy je nutné vytvořit skutečnou kopii objektu. Pro tento účel se používá již zmíněná metoda klonování. Standardně tato metoda vytváří přesnou kopii objektu včetně všech jeho vlastností.

Konstruktory a destruktory V PHP 3 byla konstruktorem metoda, jež má stejné jméno jako třída. Zde mohly vznikat problémy při dědičnosti v případě stejných názvů metod. V PHP 5 jako konstruktor slouží metoda jménem `__construct()` v příslušné třídě. Tím odpadají veškeré problémy, neboť například konstruktor předka lze volat pomocí `parent::__construct()` bez ohledu na jméno předka. Z důvodů zpětné kompatibility bude interpret v případě, že metodu `__construct()` nenajde, hledat také metodu se stejným názvem jako příslušná třída, takže stávající kód není při migraci na novou verzi nezbytně třeba měnit.

Díky implementaci odkazů na objekty je možné také definovat destruktor objektu, a to jako metodu `__destruct()`. V této metodě je možné uvést kód automaticky vykonávaný při zániku objektu - uvolnění zdrojů, zápis dat do logů a podobně.

V současné verzi PHP 4 destruktory používat nelze, neboť se nedá dost dobře poznat, kdy vlastně objekt skutečně zanikl. Místo destruktorů lze v omezené míře využít funkci `register_shutdown_function()`, která může základní „úklid“ provést, ale jde o značně neobjektové řešení.

Výjimky Ve verzi 4 neexistovalo žádné rozhraní pro řízení výjimek. Tento model představilo PHP 5 a je implementován obdobně jako u ostatních programovacích jazyků. V případě výskytu výjimky je předána výjimka části kódu, kde je možné ji programově ošetřit.

1.3 Používané způsoby tvorby internetových aplikací

Stejně rychle jako se rozvíjí jednotlivá odvětví na Internetu se rozvíjí i způsoby tvorby aplikací a webů. Protože webové aplikace jsou ve své podstatě počítačovými programy, odvíjí se způsob jejich tvorby webu nejčastěji dle zvoleného způsobu programování a možností zvoleného programovacího jazyka (např. jazyky, které podporují objektově orientované programování (OOP) nebo programování strukturované). Historicky první používanou skupinou jazyků byly tzv. *skriptovací jazyky*. Mezi ně řadíme velmi oblíbený jazyk PHP, Perl, ASP (resp. VisualBasic), nebo Python. Skriptovací jazyky jsou jednodušší, názornější a intuitivnější. To je z velké části dáno faktem, že se jedná o jazyky dynamické a slabě typované (obecně řečeno, protože např. Python je typově silný jazyk). Vzhledem k popsaným vlastnostem jsou vhodné pro začátečníky. Programový kód je vždy uložen v čitelné podobě a kompiluje jej až preprocesor, a to pokaždé, když přijde žádost o vygenerování stránky, resp. spuštění programu.

Existují různé způsoby jak se programový kód integruje do webové stránky, případně jak kompletně generuje výstup ve formě srozumitelné webovému prohlížeči. Bylo by možné shrnout tyto postupy do následujících kategorií:

1. Programový kód je *vkládán formou elementů* do existující webové stránky,
2. Program je přímo spouštěn a *generuje kompletně celé stránky* včetně okolního kódu,
3. Program je udržován odděleně a do stránky jsou výstupy generovány *prostřednictvím šablony.*

1.3.1 Skládání formou elementů skriptu

Programový kód je do stránky vkládán pomocí tzv. *Server Side Includes (SSI)*, tedy vložením kódu programu přímo do připraveného místa ve stránce. Každý skriptovací jazyk má potom definován způsob, jakým je element (blok) odlišen od zbytku stránky. Zpravidla se používá notifikace HTML elementu, tj. otevírací závorky „<" a za ní následuje specifický kód, který určuje, že za ním následující kód má být zpracován konkrétním preprocesorem (jazykem). Blok kódu je poté ukončen zrcadlovou značkou - tj. specifickým kódem a uzavírací závorkou „>" (viz příklad 1.3.1).

Jak je vidět na příkladu 1.3.1 je použito u standardního SSI uvozovací sekvence znaků `<!--#` a ukončovací `-->`. Jedná se o standardní značku HTML pro ohraničení *komentáře.* Za uvozující značku je pouze navíc vložen symbol dvojitého křížku `#`. PHP využívá pro otevření bloku programového kódu `<?php` a pro ukončení již pouze `?>`. V příkladu SSI jsou vkládány elementy kódu přímo do připravených kontejnerů, kde budou zobrazeny výsledky přímo, bez jakékoliv další úpravy. Proto je např. kolem výpisu adresáře vložen element `<pre>`, aby byl adresářový výpis ve stránce dobře čitelný a byly zachovány odsazením vytvořené sloupce výpisu souborů (výstup příkazu `ls -l`).

Naopak u příkladu PHP je vidět, že je kód vložen také přímo do stránky, ale mimo to ještě i vlastní programový kód generuje některé HTML značky (v tomto případě odkaz na úvodní stránku). Přitom je potřeba brát v úvahu, že se tu takto potkávají dva programovací jazyky: HTML a PHP. A protože každý má vlastní pravidla syntaxe, musí být ve výstupu PHP použito v elementu odkazu ošetření uvozovek zpětným lomítkem. Programátoři musí mít toto prolínání dvou prostředí na paměti a ošetřovat výstupy programu tak, aby odpovídaly nejen konvencím PHP, ale i HTML. Část HTML kódu, který řídí zobrazování obsahu, je tak obsažen uvnitř kódu programovacího jazyka. Ačkoliv zmíněné propojení bylo demonstrováno na jazyku PHP, je toto problémem u všech výše zmíněných skriptovacích jazyků.

U obou jazyků v příkladu 1.3.1 je do výstupu vkládán externí soubor `header.html`. Především tato vlastnost, která byla prvně prezentována právě ve specifikaci SSI, byla jednou z příčin, proč se začaly skriptovací jazyky používat i pro relativně jednoduché webové stránky a aplikace. Umožňuje statickou HTML stránku rozdělit na samostatné části a z nich pak skládat výsledné stránky s různým obsahem. Extrakcí stejných částí do samostatných souborů se výrazně zjednodušuje správa, protože změnu ve společné části všech stránek stačí provést pouze v jednom sdíleném souboru. Kromě této nesporné výhody se zdrojový kód

Příklad 1.3.1 Vkládání kódu formou elementů.

SSI:

```
<html><body>
<!--#include virtual="header.html"-->
<h1>Vaše soubory</h1>
<p>Vaše IP adresa je <!--#echo var="REMOTE_ADDR" --> </p>
<h2>Obsah vašeho domácího adresáře</h2>
<pre> <!--#exec cmd="ls -l"--> </pre>
</body></html>
```

PHP:

```
<html><body>
<?php include "header.html"; ?>
<h1>Ověření uživatele</h1>
<?php $adr = $_SERVER["REMOTE_ADDR"]; ?>
<p>IP adresa vašeho připojení je <?php echo $adr; ?> </p>
<h2>Ověření přístupu na tuto stránku:</h2>
<?php
if ($adr == "193.84.34.12") {
    echo "Přístup je povolen, pokračujte prosím
         na <a href=\"uvod.php\">úvodní stránku</a>";
} else {
    echo "Bohužel z této adresy nemáte povolen přístup
          na tento web.";
}
?>
</body></html>
```

Příklad 1.3.2 Generování kompletního výstupu (Perl).

```
#!/usr/local/bin/perl
foreach $item(@pairs) {
  ($key,$content)=split(/=/,$item,2);
  $fields{$key}=$content;
}
print "Content-type: text/html\n\n";
print "<html><body>\n";
print "<h1>Dokončení registrace</h1>\n";
print "<p>\nJméno uživatele: $fields{fname} $fields{lname}<br />\n";
print "E-mailová adresa: $fields{email}<br />\n";
print "<a href=\"homepage.cgi?id=$fields{id}\">Vaše homepage</a>.\n";
print "</p></body></html>";
```

rozpadne do množství malých souborů a může tak být problematické takovýto kód na úrovni HTML ladit a opravovat, neboť logika rozdělení vychází právě z logiky programového kódu.

1.3.2 Generování kompletního výstupu

Především pro programátory přecházející z tradičního programování klasických aplikací bylo v roce 1995 představeno *Common Gateway Interface* (CGI). CGI je rozhraní pro propojení externích aplikací s webovým serverem. To umožňuje serveru delegovat požadavek od klienta na externí aplikaci, která dle požadavku vrátí výstup. Určená aplikace zpracuje požadovaný skript a webovému serveru navrátí statickou stránku, která je následně poslána klientovi jako výstup jeho požadavku. Spouštění je obdobné jako u klasických programů v prostředí shellu operačního systému. Z bezpečnostních důvodů je pro CGI skripty na webserveru vymezen většinou zvláštní adresář `/cgi-bin/`, ve kterém se všechny takto spouštěné skripty nachází. Každý HTTP požadavek se spustí v samostatném procesu v jeho vlastním adresovacím prostoru, což je nevýhodné zvláště při spuštění mnoha uživateli. Proces zpracuje požadované vstupy a formou standardního výstupu na ně „odpoví" webovému serveru. Tento způsob zpracování přinášel značná bezpečnostní a výkonnostní rizika, a proto vznikly interprety integrované přímo do webserveru, umožňující běh přímo pod procesem webového serveru (mod_perl) nebo ovládané externím aplikačním serverem (FastCGI).

Pro CGI skripty je možné využívat poměrně velké množství skriptovacích programovacích jazyků, které na daném systému (serveru) existují. Mezi nejčastěji používané patří C, Perl, Basic nebo i Java.

Výstup těchto programů je kompletně v rukou programátora, který do výstupů musí kromě generovaných dat také vložit syntaxi HTML. Jak je demonstrováno na příkladu 1.3.2, je výstup třeba vždy ošetřit na znaky jako jsou např. uvozovky, které se používají jak v samotném kódu programu, tak velice často také v generovaném HTML. Pro účely generování HTML existují knihovny funkcí, kdy příslušná funkce obalí zadanou hodnotu požadovaným HTML kódem a případně zajistí ošetření sporných znaků. Tento problém se vyskytoval i v příkladu 1.3.1, nicméně zde je většina HTML kódu umístěna mimo prováděný program a výstupy programu obsahují jen menší množství HTML kódu. Generování kompletního vý-

Šablonovací nástroj	Určený pro jazyky	Platforma
StringTemplate	Java (nativní), Python, C#	multiplatformní
ASP.NET (Microsoft)	C#, VB.NET	Microsoft Windows
ASP.NET (Mono)	C#	multiplatformní
Apache Velocity	Java	multiplatformní
Beilpuz	PHP 5	multiplatformní
CheetahTemplate	Python	multiplatformní
Chip Template Engine	PHP, Perl	multiplatformní
CTPP	C, C++, Perl, PHP, Python	multiplatformní
Dylan Server Pages	Dylan language	nezjištěno
eRuby	Ruby	multiplatformní
Evoque Templating	Python	multiplatformní
FreeMarker	Java	multiplatformní
Genshi (templating language)	Python	multiplatformní
Haml	Ruby, PHP (WIP)	multiplatformní
JSP Weaver	Java	multiplatformní
Jasper framework	Perl, PHP, C#, Java	multiplatformní
JavaServer Pages	Java	multiplatformní
Jinja (Template engine)	Python	multiplatformní
Kid (templating language)	Python	multiplatformní
Open Power Template	PHP 5	multiplatformní
Outline	PHP 5	multiplatformní
PHAML	PHP	multiplatformní
Smarty	PHP	multiplatformní
TinyButStrong	PHP	multiplatformní
Vemplator	PHP	multiplatformní
VlibTemplate	PHP	multiplatformní
Zeihro TemplateEngine	PHP	multiplatformní
WebMacro	Java	multiplatformní

Tabulka 1.1: Přehled některých šablonovacích nástrojů.

stupu prostřednictvím programu tak na programátora klade další nároky a přenáší se na něj také odpovědnost za vzhled stránek.

1.3.3 Generování prostřednictvím šablon

Vznik šablonovacích systémů byl dán především požadavky na rychlou produkci webových stránek, kdy je možné kombinovat různé typy vzhledů nad jedním programovým kódem. Programovým kódem může být jádro CMS systému[10], který generuje stejnou sadu dat. Je tak při zachování stejných dat možné vytvářet různě vypadající a strukturované HTML dokumenty, které zobrazují stejné informace. Kromě snadné integrace s daty umožňuje oddělení aplikační od prezentační vrstvy vytvářet prototypové stránky nebo elementy stránek nezávisle na vývoji programu. Zároveň umožňuje poměrně jednoduše měnit vzhled existujícího webu, mnohdy i bez účasti aplikačního programátora.

[10]CMS - *Content Management System* je aplikace umožňující vytvářet nebo editovat současný obsah např. webových stránek. Obsah v nich vytvořený poté umožňuje generovat do obsahu webových stránek, které si prohlíží návštěvníci. Umožňuje tak rychle aktualizovat určené části webu.

Příklad 1.3.3 Šablona HTML výstupu ve Smarty (index.tpl).

```
<html>
<head><title>{$title_text}</title></head>
<body>
<p>{$body_text}</p>
</body>
</html>
```

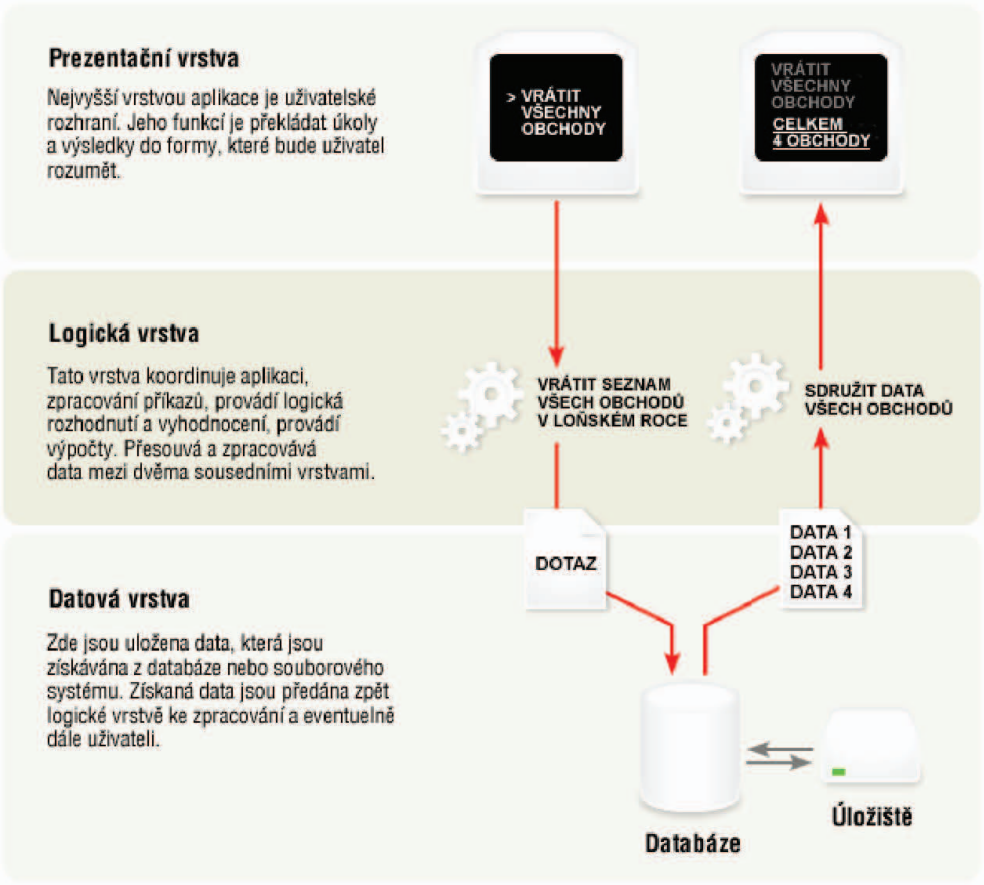
Příklad 1.3.4 Program plnící šablonu Smarty.

```
define('SMARTY_DIR', 'smarty-2.6.9/' );
require_once(SMARTY_DIR . 'Smarty.class.php');

$smarty = new Smarty();
$smarty->template_dir = './templates/';
$smarty->compile_dir = './templates/compile/';
$smarty->cache_dir = './templates/cache/';
$smarty->caching = false;
$smarty->error_reporting = E_ALL;
$smarty->debugging = true;

$smarty->assign('title_text', 'Textový titulek testovací stránky');
$smarty->assign('body_text', 'Tento kód byl dodán do šablony stránky.');

$smarty->display('index.tpl');
```



Obrázek 1.4: Třívrstvá architektura.

1.4 Charakteristika webových aplikací

S rozvojem Internetu a využíváním třívrstvé architektury (viz. schéma na obr. 1.4) se při návrhu a tvorbě aplikací a informačních systémů ve stále vetší míře prosazují internetové, resp. webové technologie. Hlavní výhodou webových aplikací je nezávislost na použitém hardwarovém a programovém vybavení na straně klienta, možnost volného přístupu z libovolného počítače s připojením do Internetu, rychlá a bezproblémová aktualizace libovolné programové komponenty a také možnost využívat výhod nejmodernějších postupů a technologií [5]. Služba WWW je tak využívána jako prostředek uživatelského rozhraní k aplikacím. Aplikace, které tohoto rozhraní využívají, se označují jako webové aplikace [10].

Webové aplikace, resp. jejich uživatelské rozhraní je doručeno uživatelům z webového serveru přes síť World Wide Web. Populární jsou zejména kvůli všudypřítomnosti webového prohlížeče jako klienta, označovaného jako *tenký klient* [5]. Klíčovými důvody pro rozšířenost webových aplikací je schopnost aktualizace a údržby bez nutnosti distribuovat a instalovat software u stovek potenciálních klientů. Spektrum webových aplikací je velmi široké - používají se například pro implementaci rozhraní poštovních klientů, e-komerci, on-line aukce, diskusní fóra, weblogy[11], portály, apod.

[11] *Weblog* je v podstatě publikační systém, obsahující periodicky řazené příspěvky na nejrůznější témata.

U dříve používaných aplikací typu klient-server měla aplikace svůj vlastní klientský program, který byl nainstalován u jednotlivých klientů. Aktualizace serverové části aplikace obvykle vyžadovala také aktualizaci klientů, což bylo časově i finančně náročné.

Naproti tomu webové aplikace dynamicky generují série webových stránek ve standardním formátu, který podporují všechny dnešní internetové prohlížeče (typicky ve formátu HTML/XHTML). Skripty pro stranu klienta (prohlížeče) jsou obvykle napsány v jazyce JavaScript a jsou distribuovány s webovou stránkou a většinou pouze rozšiřují uživatelské rozhraní o dynamické elementy. Uživatelské vstupy jsou aplikaci běžící na straně serveru zprostředkovány sadou standardizovaných formulářů a parametrů URL adres. Internetový prohlížeč vystupuje jako univerzální klient pro jakoukoliv webovou aplikaci [5].

Podstatnou výhodou při tvorbě webových aplikací běžících v internetovém prohlížeči je schopnost pracovat nezávisle na operačním systému nebo jeho verzi. Na druhé straně nekonzistentní implementace HTML, CSS, DOM a ostatních specifik internetových prohlížečů mohou působit problémy při tvorbě a údržbě systémů. Uživatelé mohou ve svých prohlížečích upravovat některá nastavení vzhledu (velikost a typ fontů, barvy, zákaz klientských skriptů, apod.), která mohou být překážkou udržení konzistentního vzhledu webové aplikace.

Jiným přístupem je používání Macromedia Flash nebo Java appletů k tvorbě částí nebo celého uživatelského rozhraní. Většina internetových prohlížečů tyto technologie podporuje (nejčastěji prostřednictvím zásuvných modulů - *plug-ins*) a distribuce Flashově- nebo Javově-založených aplikací je obdobná jako u klasických webových aplikací. Navíc mohou mít tyto aplikace přístup k nastavením prohlížeče a dalším prostředkům, ke kterým běžné webové aplikace přístup nemají. Díky podobnosti jejich architektury s tradičními klient-server aplikacemi jsou jakýmsi „tlustým“ klientem pro tyto aplikace a používá se pro ně v poslední době termín „rich internet application“ - tedy „bohaté internetové aplikace“.

1.5 Specifika webových aplikací

Šorm [44] popisuje specifika webových aplikací z hlediska technologického a uvádí především *nelinearitu, bezstavovost, zajištění autorizace a autentizace, šablonování,* nebo *modularitu.* Tyto aspekty odlišují vývoj aplikací určených pro provoz na internetu od klasického přístupu a proto jsou následující oddíly věnovány bližšímu popisu uvedených specifik.

1.5.1 Nelinearita

Při tvorbě webových aplikací je výrazným problémem nelinearita jejich používání. Uživatel může začít aplikaci používat v neočekávaném bodě (např. od konce sekvence požadavků), může pozměnit posloupnost kroků (návrat pomocí vestavěné funkce *Zpět* v prohlížeči), případně některý krok zopakovat (znovunačtení stránky tlačítkem *Obnovit*). Většinu těchto případů lze řešit pomocí metody serializace požadavků - jednotné identifikace požadavků, jejich posloupnosti a jednoznačnosti. Každý požadavek je označen jednoznačným identifikátorem, který je distribuován do všech ovládacích prvků stránky (formuláře, odkazy). Při

Díky webovým aplikacím byla jejich správa a údržba zautomatizována a *weblogy* se tak staly dostupné i pro obyčejné uživatele Internetu.

provedení akce uživatelem je použití tohoto identifikátoru zaznamenáno do databáze. Při opětovném pokusu o provedení stejné operace systém detekuje duplicitní použití identifikátoru a provede obsluhu ošetření požadované operace. Tímto jednoduchých mechanismem je uživatel nucen pohybovat se v systému lineárně. Šorm [43] nelinearitu vyvozuje z bezstavovosti protokolu HTTP, kdy jednotlivé požadavky a prováděné operace nemají mezi sebou žádnou návaznost, podobně jako nemají pořadí ani vstupní bod a jsou v podstatě izolovanými přístupy k informačnímu systému.

1.5.2 Bezstavé prostředí

Specifikem prostředí WWW oproti klasické architektuře klient-server je tzv. bezstavové programování. Ve skutečnosti ovšem bezstavovost programování vyplývá z bezstavového HyperText Transfer Protocol (HTTP), který je založen na principu dotazů a odpovědí bez jakéhokoliv uchovávání historie komunikace klienta se serverem. Jednotlivé operace jsou podle Eernisse [8] v podstatě izolovanými přístupy do systému, nemají na sebe žádnou návaznost. Zajištění požadované posloupnosti jednotlivých kroků vyžaduje transport řídících dat mezi těmito kroky. To lze řešit předáváním parametrů mezi požadavky nebo pomocí skladu parametrů relace. Přenos parametru je zajišťován postupným sběrem zadaných údajů a jejich předávání pomocí skrytých prvků jednotlivých formulářů. Nevýhodou je přenos vyššího množství údajů a také vyšší náročnost na aplikačního programátora, který musí zajistit předávání všech relevantních hodnot v příslušné části aplikace. Výhodou je jednoduchost a šetrnost v nakládání s datovými zdroji. Druhou možností je sklad parametrů relace. Shromážděné požadavky jsou ukládány do speciální datové struktury pod jednoznačným identifikátorem, který je mezi požadavky předáván jako běžný parametr. Pomocí něj je možné získat všechna požadovaná data z datového úložiště. Problém vzniká, pokud uživatel přeruší posloupnost kroků a data začínají stárnout. Je tedy vhodné periodicky odstraňovat zastaralé záznamy. Výše popsaný způsob se využívá u tzv. *sessions*.

Problém bezstavovosti internetového protokolu HTTP se snaží odstranit například platforma ASP.NET. Popisované přístupy a metody je vhodné použít u dynamických jazyků, které nepodporují tzv. *viewstate* a jemu podobné přístupy, nebo u rozdílných skriptů. Drlík [6] definuje viewstate jako nástroj umožňující uchování stavu stránky a serverových objektů na ní umístěných mezi jejím opakovaným zpracováním. Narozdíl od jiných metod uchování stavu v ASP.NET aplikacích, které je možno použít v rozsahu platnosti uživatele aplikace (Session) nebo celé aplikace (Application, Cache...) je tedy jeho platnost omezena pouze po dobu tohoto opakovaného zpracování jedné stránky. ViewState je využíván zejména pro uchování hodnot vlastností ovládacích prvků umístěných na stránce, ale lze ji samozřejmě také použít v kódu stránky. Toto lze například využít ve stránce s detailem entity načtené z databáze pro uložení hodnot, které identifikují entitu a přitom se nezobrazují ve formuláři (primární klíč a podobně). ViewState se z principu nehodí a nedoporučuje pro ukládání jakýchkoli citlivých dat. Pokud aplikace takováto data do ViewState ukládá, je nutné zvýšit jejich zabezpečení buďto nastavením šifrování ViewState, nebo jeho ukládáním do jiných úložišť než je ve výchozí implementaci předvolené skryté pole v generované stránce. Tímto způsobem je programově ošetřována bezstavovost internetového protokolu

na platformě Microsoft .NET, i když bezstavovost zcela neodbourává.

1.5.3 Autentizace a autorizace

Mnohé webové aplikace nejsou zcela otevřenými systémy a proto je nutné zajistit do nich pouze omezený přístup. Standardní metodou je uživatelské jméno a heslo. Zjištění a prokázání totožnosti jsou často řešené otázky i u běžných informačních systémů. V případě nestavového a nelineárního prostředí WWW však představují podstatně vetší problém. Vše ještě komplikují omezené možnosti klientské aplikace. Řešení mohou být vystavěna buď na bázi tiketování, nebo autentizací závislou na použité platformě. Tiketování představuje obecnější řešení - po úspěšné autorizaci je uživateli vygenerován tiket (jednoznačný identifikátor) s omezenou platností, který je dále předáván mezi požadavky. Pokud platnost tiketu vyprší nebo jej aplikace nepředá, je uživatel ze systému odhlášen. To umožňuje jednoduše sledovat dobu přihlášení uživatele, využití zdrojů nebo bezpečné odhlášení. Nevýhodné je zejména neustálé generování tiketu. Autentizace závislá na platformě může být realizována například pomocí databázového autentizačního modulu pro server Apache[12], který zajišťuje transparentní autentizační proces podle daných pravidel. Některé klientské aplikace však nedokáží tento proces korektně řídit a zejména bezpečné odhlášení ze systému lze provést pouze ukončením klienta (internetového prohlížeče).

V souvislosti se zabezpečením přístupu k webovým aplikacím a na základě studie skupiny OWASP[13] definuje Hrdina [15] deset nejzávažnějších zranitelností webových aplikací, které by si měl každý bezpečnostní manažer prostudovat (dokument v původním znění je publikován na stránkách `http://www.owasp.org/documentation/topten`). Seznam popisuje zranitelnosti, či bezpečnostní nedostatky, ke kterým se váže nejvíce úspěšných útoků po Internetu a podrobněji jsou uvedeny v oddílu 1.7.2.

1.5.4 Šablonování

Při provozu velkých webových aplikací dochází k častým změnám v požadavcích na evidenci informací o různých objektech. Zajímavým řešením této problematiky se jeví metoda šablonování. Ta umožňuje definici šablon objektu, které popisují jeho jednotlivé atributy. Využitím šablon lze dosáhnout maximální parametrizace systému. Praxe však ukázala, že necitlivé používání šablon má poměrně nepříznivý vliv na databázový výkon při vyhledávání nebo třídění podle šablonovaných údajů. Pro správnou funkčnost je tedy důležité najít vhodný kompromis mezi běžným relačním přístupem a šablonováním. Tuto problematiku ještě více rozvádí Šorm [43, 44]. Využití šablonování umožňuje vytvářet aplikace řízené datovou vrstvou, kdy standardizované aplikace (standardizaci vzhledu zajistí prezentační vrstva, standardizaci chování pak objektový model aplikační vrstvy) zjišťují definici dat až po svém vytvoření a tím ovlivňují náplň své práce. Uživatelé získávají nástroj, kterým mohou systém rozšiřovat a vylepšovat bez nutnosti konzultovat vývojový tým.

[12] modul *mod_auth* serveru Apache
[13] OWASP = **O**pen **W**eb **A**pplication **S**ecurity **P**roject

1.5.5 Modularita

Dosažení maximální efektivnosti při vývoji rozsáhlejších webových aplikací (např. informačních systémů) je možné pouze při důsledném uplatnění modulárního přístupu při návrhu systému. Lze tak připravovat aplikace pomocí prototypování, libovolně měnit počet vývojářů pracujících na konkrétním modulu, průběžně uvolňovat výsledky, připravovat aplikace pomocí prototypů, opravovat a modernizovat jednotlivé části systému podle požadavků uživatelů bez nutnosti zásahu do dalších komponent systému [43].

1.6 Typy webových aplikací

Dnes se již internetové stránky neomezují jen na prosté sdělování informací. Výkonné servery a nové technologie umožňují vytvořit interaktivní aplikace, které mohou být provázané i s jinými, existujícími systémy (internetovými nebo klasickými). V obchodní sféře jsou nejběžnější následující typy webových aplikací.

Dynamické internetové stránky Jelikož internetová prezentace již v současné nezobrazuje pouze statické texty, ale obsahuje mnoho dalších prvků, které mají za cíl přitáhnout návštěvníky, a jejichž funkcionalitu je nutné skutečně naprogramovat (kalkulátory, vyhledávače, překladače, hlasování, diskuse, apod.), stávají se z nich webové aplikace vytvořené na klíč.

E-shop Elektronický obchod, kde si může návštěvník pohodlně vybrat zboží a jednoduše objednávku odeslat. Aplikace pro elektronické obchodování v sobě obsahují velké množství dílčích funkčních celků (tříděné a omezené výpisy, práce s obrázky, princip nákupního košíku, zabezpečená komunikace, e-mailové služby, porovnávání, apod.).

Redakční systém Redakčním systémem, nebo také CMS[14], se rozumí jakákoli aplikace, která umožňuje oprávněnému uživateli jednoduše měnit obsah stránek internetové prezentace. Většina redakčních systémů poskytuje moduly pro editaci textů, přidělování práv uživatelům, správa rubrik, manipulace s fotografiemi a obrázky, služby pro správu samostatných souborů, apod.

Informační systémy Typické internetové IS pokrývají specifickou oblast podniku (například CRM[15], ERP[16], apod.). Většinou jde o oblasti, kde klíčovým faktorem bývá snadná dostupnost uložených dat, tedy např. právě systém CRM. Výjimkou však nejsou ani rozsáhlé systémy integrující řízení výroby, fakturaci, účetnictví a řízení dodavatelsko-odběratelských vztahů.

[14] *CMS* Content Management System - Systém pro správu obsahu
[15] *CRM* = Customer Relationship Management - Systém řízení vztahů se zákazníky
[16] *ERP* = Enterprise Resource Planning - Systém automatizace procesů souvisejících s produkčními činnostmi

Bankovní aplikace Už jen málokterá firma by dokázala existovat bez přímého přístupu do banky. Provádění finančních transakcí on-line bezpochyby ušetří spoustu času. Jedná se o provádění bankovních operací, příjem či odesílání plateb nebo obchodování s akciemi na webu prostřednictvím počítače. Nedávno se do této oblasti přiřadilo využívání mobilních telefonů, zařízení PDA či jiných bezdrátových zařízení. Při přenosu dat týkajících se finančních prostředků a investic na webu je však důležité získat co nejvíce informací o zabezpečení takových transakcí [31].

Bylo by poměrně složité vyjmenovat typy aplikací, které lze provozovat prostřednictvím Internetu. Technologie jsou stále dokonalejší a proto je možné na Internetu provozovat už i ty aplikace u nichž se zdálo, že jsou do prostředí Internetu nepřenosné (mapové a GIS systémy, kreslící software, kancelářské balíky, apod.). Určitým „hnacím motorem" tohoto rozvoje jsou internetové portály, které do sebe integrují služby nejrůznějšího druhu - zpravodajství, rezervace letenek, předpověď počasí, slovníky, mapové systémy až po e-mailové klienty, které se vyrovnají téměř klientům klasickým. Protože na portály lze nahlížet z různých pohledů budou jim věnovány následující oddíly.

1.6.1 Veřejný internetový portál

Portál je množina technologií a aplikací, tvořící univerzální rozhraní, jehož prostřednictvím je každému, kdo se dotýkají činnosti organizace (zákazník, dodavatel, zaměstnanec, apod.), umožněno účastnit se procesů organizace, přistupovat ke všem relevantním informacím, komunikovat s ostatními participujícími lidmi a realizovat adekvátní aktivity spojené s podnikovými procesy [10].

Historicky první internetové portály vznikaly jako určité rozcestníky, směřující uživatele na další zdroje informací. Dalším vývojem se však jejich účel podstatně změnil. Dnešní portál se naopak snaží udržet si uživatele co nejdéle a poskytnout mu vše, co by mohl potřebovat. Služby internetových portálů zahrnují zpravodajství, vyhledávání, emailové služby a nejrůznější informační služby. Aby výsledný přísun informací uživatele nezahltil, má příjemce možnost specifikovat, které části hodlá sledovat, a které služby ho naopak nezajímají. Fakticky si tak uživatel sám přizpůsobuje nabídku portálu, včetně povahy, rozsahu a podoby předkládaných informací. Veřejné internetové portály však primárně nabízí pouze to, co se vyplatí jejich provozovateli, což nemusí nutně odpovídat požadavkům a potřebám všech uživatelů. To samozřejmě souvisí i s použitým obchodním modelem a neochotou uživatelů za cokoliv platit.

1.6.2 Firemní portál

Odlišná je však situace ve firmách. Zde existují poměrně přesně definované skupiny uživatelů s přesně vymezenými potřebami. Vhodně navržený portál jim nabízí informace a služby, které potřebují ke své činnosti. Rozdíl mezi veřejným a firemním portálem existuje i po stránce ekonomické - zavedení firemního portálu může organizaci značně uspořit náklady, které vynakládá na podporu svých zaměstnanců. Dalo by se říci, že vytváření podnikových portálů se stalo trendem poslední doby. Svědčí o tom i nemalý zájem softwarových firem o tuto oblast. Většina velkých softwarových společností nabízí vlastní řešení, které pokrývá kom-

pletní problematiku tvorby a provozu informačního portálu organizace. Mezi neznámější patří firmy Oracle, IBM, Microsoft, Sybase či Novell. Podnikový portál můžeme podle Manhartsbergera [30] definovat jako víceúrovňový technologický systém, který integruje procesy, aplikace i data, a vytváří jedno celistvé prostředí, jež umožňuje společnostem poskytnout jednotný komunikační kanál všem, kdo se chtějí zúčastnit jeho aktivit. Výhod portálu by měl mít možnost využívat každý, kdo má zájem zúčastňovat se aktivit organizace. Portál by tedy neměl být určen pouze pro zaměstnance, ale jeho služeb by měli využívat i obchodní partneři firmy. Portál by v takovém ideálním případě tvořil prostředníka mezi jednotlivými stranami, pomocí něhož si mohou vyměňovat informace.

Podle Drakose, citovaného v [10], prošly portály v průběhu svého vývoje čtyřmi zásadními fázemi svého vývoje:

1. Portál jako *vstupní bod.* V první fázi svého vývoje byl portál chápán pouze jako vstupní bod („rozcestník"), poskytující jak různé informace, tak i prostředky pro jednoduché vyhledávání v nich. Přínosem tohoto řešení je vybudování centralizovaného místa přístupu k informacím.

2. Portál jako *integrátor obsahu.* Ve druhé fázi svého vývoje se portálové řešení stává centrálním úložištěm podnikových dat s tím, že nabízí pokročilé mechanismy vyhledávání a zpracování dat (klasifikace informací, jejich kategorizace a indexace). Zároveň nabízí funkce customizace a personalizace řešení pro každého uživatele, včetně definování preferenčních pravidel, která zajišťují selektivní přístup k datům.

3. Portál jako *integrátor „pracovní plochy".* Ve třetí fázi vývoje dochází k situaci, kdy se portál svým univerzálním rozhraním stává jakousi novou „pracovní plochou" (v analogii k ploše operačního systému), jejímž prostřednictvím je uživateli zajištěn přístup ke všem relevantním datům a aplikacím.

4. Portál jako *integrátor trhu.* Čtvrtá fáze reaguje na potřeby mezipodnikové integrace, kdy uživatelům již pro realizaci procesů nepostačují informace a aplikace na úrovni podniku, ale chtějí mít zajištěn přístup i k datům a aplikacím kooperujících subjektů. Typicky dochází k integraci s jinými portály, případně elektronickými tržišti.

Podnikatelské sféry v nejrůznějších odvětvích vytváří specializované portály pro rychlejší a efektivnější B2B[17] a B2C[18] komunikaci. Největší výhoda je spatřována především v nástrojích, které umožňují doručení obsahu (informací), časově a věcně platných v daný okamžik, širokému spektru uživatelů napříč různým geografickým vzdálenostem a hranicím.

Portál může také sdílením poznatků a informací přispívat ke zvyšování produktivity práce a poskytovat ucelený a souhrnný pohled na firmu jako celek pro zákazníky, dodavatele, investory, partnery i zaměstnance. To může v některých firmách ušetřit náklady na distribuci podobných materiálů.

Obecně portály umožňují společnostem kombinovat možnosti dynamického a okamžitě přístupného obsahu se službami a aplikacemi ve formátu, který je personalizován pro každé-

[17] B2B bussiness-to-bussiness
[18] B2C = bussiness-to-customer

ho uživatele. Podle Homana |14| by měl dobrý portál poskytovat určité množství klíčových schopností, zahrnujíce:

- *intuitivní a přizpůsobitelné webové rozhraní* - portály by měly nabízet rozhraní se srozumitelnou navigací, která může být designována v souladu s existujícími aplikacemi dané firmy. Na pozadí by toto rozhraní mělo sloužit jako brána k bezpečnosti a kontrole přístupů pro systémového administrátora a pro služby jakými je například vyhledávání. Portály se mohou značně lišit v možnostech přizpůsobitelnosti a flexibility,
- *personalizovaná prezentace obsahu* - na základě uživatelského profilu je možné poskytnout přizpůsobený obsah a umožnit přístup jen do těch rubrik, do kterých má uděleno právo. Jiný pohled a jiné informace budou poskytovány zaměstnanci firmy a jiné obchodnímu partnerovi,
- *bezpečnost* - zvláště pokud jsou na portálu prezentována data, která mají být poskytnuta jen některým uživatelům, je velice důležité dbát na zabezpečení portálu. Kromě zabezpečeného přístupu, např. k důvěrným firemním informacím, by měl být také odlišen přístup do administrační části portálu, určené pro správu vlastních informací,
- *aplikační integrace* - do portálů se čím dál častěji zařazují nejrůznější aplikace. Mohou to být například bankovní a kurzovní kalkulátory, aplikace pro výpočty daní, leasingu, vnitropodnikové aplikace, apod. Výhodou těchto aplikací je především jejich dostupnost, aktuální podkladové údaje a pak také jednotné přihlášení označované jako *Single SingOn*,
- *komunikace a spolupráce* - mezi tyto možnosti se zahrnuje chat, e-mail, sdílené kalendáře, tématické diskuse, webové konference a management zdrojů. V některých případech spolupracují aplikace s komunikačními nástroji.

Podnikové portály Manhartsberger |30| a Gála |10| dále rozčleňují do podkategorií, které přestavují různé modifikované přístupy k tvorbě takovéhoto informačního média podniku:

- *aplikační podnikový portál* - provozují se zde systémy jako *ERP* (Enterprise Resource Planning), řízení vztahu se zákazníky - *CRM* (Customer Relation Management), řízení dodavatelských řetězců *SCM* (Supply Chain Management), řízení nákupu (procurement), ad.
- *informační podnikový portál*,
  - *portál pro podporu rozhodování* - jeho primárním úkolem je zpřístupnit uživateli informace, které používá při rozhodování,
  - *portál pro podporu spolupráce* - umožňuje vzájemnou spolupráci a komunikaci vybraných subjektů, která může být realizována elektronickou poštou, vývěskami, konferencemi, fóry, chaty, apod., dalšími složkami jsou sdílená data, workflow, týmová spolupráce,
  - *znalostní portál* - základním cílem tohoto typu je spojit „ty, kteří vědí, s těmi, kteří potřebují znát“, proměnit znalosti jednotlivců ve znalosti organizace.

- ▷ *vertikální portál* - VORTAL - dochází zde ke spolupráci mezi organizacemi ve stejném odvětví nebo v rámci projektů.

Woods [52]vidí pohlíží na budování rozvinutého (enterprise) portálového řešení jako na datovou základnu, nad kterou je možné budovat kromě výše zmíněných aplikačních portálů ERP, CRM, SCM, ... také odvozené portály jako jsou samoobslužný zaměstnanecký portál, samoobslužný manažerský portál, komunitní a zájmové portály, portály obchodního styku a informací, divizní portály, apod.

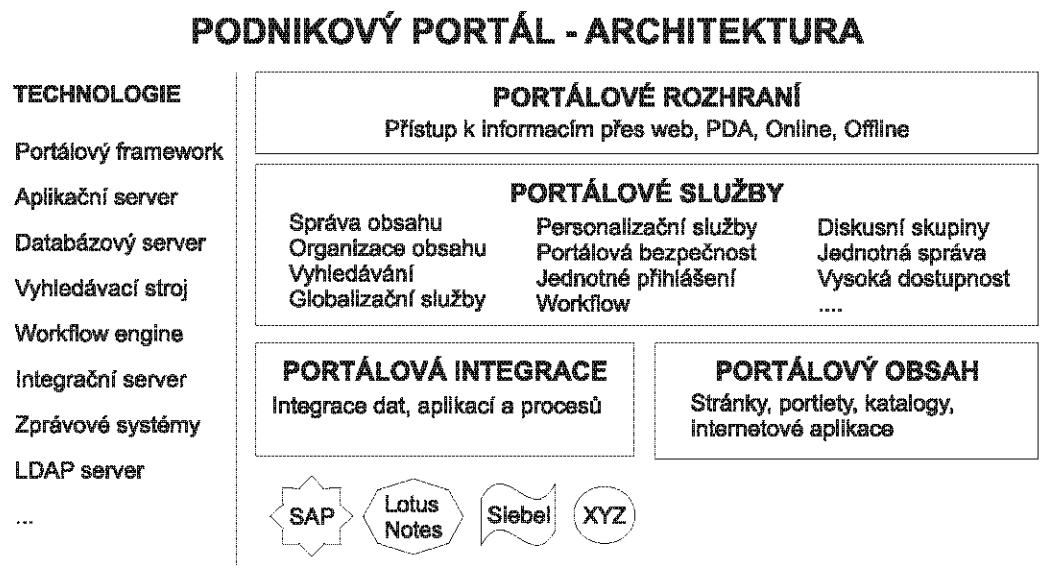
Kotek [25] definuje problém *informačního zahlcení*, které bylo nejsilnějším impulzem pro vznik podnikových portálů. Problém informačního zahlcení v podnicích je velmi podobný pohybu v „informačním moři" tvořeném sítí Internet, a proto se vzorem pro podnikové portály staly internetové informační a vyhledávací portály. Podnikové portály se ale pohybují v poněkud odlišném prostředí - musí zajišťovat přístup nejen k informacím uloženým ve webových systémech, ale také k informačním bázím (Lotus Notes, MS Exchange, ...), dokumentovým systémům (Stellent, Documentum, ...) nebo relačním databázím. Zároveň, a to je jedna z výrazných odlišností podnikových portálů, musí zpřístupňovat vedle prostých informačních zdrojů také nejrůznější podnikové aplikace a informační systémy (SAP R/3, Oracle PeopleSoft, Siebel, ...). To vše musí být zaštítěno odpovídajícím řešením bezpečnosti. Podnikový portál tak vedle personalizovaného přístupu k organizovaným informacím musí poskytovat ještě široké spektrum dalších funkcí a služeb sloužících ke zpřístupnění heterogenního IT prostředí podniku. Namátkou lze zmínit služby jednotného přihlášení a správy identity uživatelů nebo rozsáhlou oblast technologií pro podnikovou aplikační integraci a integraci podnikových procesů, které využívají portály pro zpřístupnění aplikačních balíků. Tyto požadavky na podnikový portál se pak odráží na použitých technologiích. Bohužel tato složitost podnikových portálů má negativní dopad na celkové náklady na vlastnictví podnikového portálu, a to jak v oblasti investic do software, tak zejména u nákladů na implementaci, rozvoj a údržbu portálového řešení.

1.6.3 Model ASP (Application Service Providing)

ASP přestavuje nový způsob dodávání a řízení aplikací, který se dívá na software zcela z jiného úhlu a pro internetové komerční aplikace se jeví jako nejvýhodnější. Jedná se o nové paradigma - software jako služba. ASP je ve skutečnosti obchodní model, který řeší otázku dodání aplikace zákazníkovi jiným způsobem než prodejem licencí. ASP je označením pro poskytování, nebo poskytovatele aplikačních služeb, kteří se zabývají pronájmem informatické a telekomunikační infrastruktury a poskytováním podpůrných služeb (servis, správa dat, atd.).

V podstatě se jedná o to, že poskytovatel si zajistí softwarovou aplikaci, zařízení pro její provoz a infrastrukturu pro vzdálené připojení. Potom je opakovaně pronajímá zákazníkům, typicky na omezenou dobu, a za pevný poplatek umožní klientům přístup k aplikaci. Ve světě IS/IT je v současné době obchodní model ASP velice diskutovaným tématem a to převážně v USA, Austrálii a západní Evropě. Rapidně přibývá poskytovatelů aplikačních služeb a velikost trhu roste.

Jako dobrý marketingový nástroj pro pochopení systému ASP a možností jeho úspěšného



Obrázek 1.5: Architektura podnikového portálu podle Kotka [25].

prosazení ve firmě je možnost bezplatného vyzkoušení (např. po dobu jednoho či dvou měsíců). Podle studie ČSSI touto vlastností disponuje necelá třetina produktů, což je poměrně malá část.

Díky faktu, že aplikace zůstává fyzicky umístěna na serverech poskytovatele, nemůže docházet k nelegálnímu kopírování a zneužívání autorských práv. Poskytovatel zprostředkovává po určitou konkrétní dobu zákazníku pouze přístup do aplikace. Po ukončení smlouvy je tento přístup zamezen zrušením přístupových práv.

1.7 Zhodnocení webových aplikací

1.7.1 Výhody

Hlavní výhody webových aplikací byly podrobněji zmíněny v oddílu 1.4. Jsou to především následující vlastnosti:

- nezávislost na použitém hardwarovém a programovém vybavení na straně klienta,
- možnost volného přístupu z libovolného počítače s připojením do Internetu,
- rychlá a bezproblémová aktualizace libovolné programové komponenty,
- velké rozšíření zobrazovacího klienta (internetového prohlížeče),
- možnost využívat výhod nejmodernějších postupů a technologií,
- rozšířené možnosti práce s jakoukoliv částí aplikace (tisk, kopírování obsahu, zvětšení písma, apod.),
- v současnosti velmi populární přístup - dynamický rozvoj technologií pro tvorbu, vysoká kvalita komponent,

- ochrana autorských práv tvůrce aplikace.

Darie |5|v úvaze nad vývojem desktopových a webových aplikací shrnuje následující přínosy webových aplikací:

- **Webové aplikace je jednoduché a levné dodat uživatelům**. S jejich pomocí mohou společnosti snížit náklady na IT, které spočívají v instalaci software na počítačích jednotlivých uživatelů. Vše, co uživatel pracující s webovou aplikací potřebuje, je počítač s webovým prohlížečem a spojením do internetu nebo intranetu.
- **Webové aplikace je jednoduché a levné aktualizovat**. Náklady na údržbu software byly vždy významné. Protože se aktualizace stávající části kódu dosti podobá nové instalaci, jsou zde přínosy webových aplikací také významné. Každý uživatel může mít k dispozici novou verzi aplikace chvilku poté, co byla aktualizována na serveru.
- **Webové aplikace mají flexibilní požadavky na uživatele**. Je-li aplikace instalována na serveru – na jakémkoliv moderním operačním systému – je možné ji prostřednictvím internetu nebo intranetu používat na jakémkoliv stroji, na kterém běží MacOS, Windows nebo Linux. Je-li tato aplikace správně vytvořena, bude dobře fungovat ve kterémkoliv moderním webovém prohlížeči, ať už se jedná o Internet Explorer, Mozilla Firefox, Opera, nebo Safari.
- **Webové aplikace umožňují snazší centralizovanou správu dat**. Je-li třeba přistupovat ke stejným datům z několika různých míst, je jednodušší mít všechna data uložena na jednom místě, než v několika oddělených databázích. Takto lze předejít možným rizikům a problémům se synchronizací a bezpečností.

1.7.2 Hlavní slabiny

Bezpečnostní manažeři pro IT podle Hrdiny [15] čelí neustále se měnícím a stále důmyslnějším hrozbám. Základní internetové služby (e-mail, FTP a HTTP) jsou dnes obklopeny řadou dynamických webových aplikací, serverů a databází, které jsou dostupné čtyřiadvacet hodin denně, sedm dní v týdnu. Tyto aplikace vytvářejí infrastrukturu digitální ekonomiky, což z nich činí velmi atraktivní cíle pro hackery. Internetová bezpečnost, která přerostla oblast síťové vrstvy dnes vyžaduje specifická opatření pro ochranu sítě, vnitřní bezpečnost a bezpečnost webových aplikací.

Seznam studie skupiny OWASP popisuje deset zranitelností, či bezpečnostních nedostatků, ke kterým se váže nejvíce úspěšných útoků po Internetu. Seznam je zaměřen na zranitelnosti na úrovni kódu aplikací. Snížení rizik pramenících z těchto nedostatků a zranitelností obvykle vyžaduje celou řadu nástrojů a technik, které jsou navíc založeny na různých technologiích. Hrdina [15] uvádí následující výčet, včetně krátkých popisů.

Neověřený vstup Webové aplikace používají vstup z požadavků HTTP pro určení toho, jak na ně reagovat. Informace HTTP lze zakódovat mnoha různými způsoby. Velice často nejsou webové požadavky ověřeny před tím, než jsou použity webovou aplikací. Útočníci tak mohou zfalšovat libovolnou část HTTP požadavku - včetně např. URL, vyhledávacího řetězce nebo záhlaví - a pokusit se tak obejít zabezpečovací mechanismy webových aplikací.

Narušení kontroly přístupu Problém spočívá v nedostatečném zajištění přístupových práv uživatelů. Útočníci mohou odhalením nedostatků získat přístup k účtům jiných uživatelů, citlivým souborům nebo kritickým funkcím. Mezi nejčastější problémy patří techniky průniku do adresářů (directory traversal), výchozí práva k souborům a nezabezpečená uživatelská jména a hesla.

Porušení správy účtů a relací Slabá ochrana uživatelských údajů sloužících k přihlášení a údajů identifikujících relaci představuje další slabé místo popisované skupinou OWASP. Útočníkům je umožněno získat hesla, klíče, cookies relací a další informace, které jim umožní obejít autentizační a restrikční mechanismy a získat cizí identitu.

Zneužití serveru k odesílání skriptů[19] Webových aplikací lze zneužít jako mechanismu k přenosu útoku na prohlížeč koncového uživatele. Úspěšný útok může vést k vyzrazení přihlašovacích informací koncového uživatele, útoku na jeho počítač či zfalšování obsahu webové stránky k oklamání uživatele.

Přetečení bufferu Nejsou-li komponenty webových aplikací v některých jazycích dostatečně chráněny ověřením vstupu, je možné způsobit jejich zhroucení, kterého je pak možno v některých případech zneužít k převzetí kontroly nad procesem. Mezi tyto komponenty patří CGI, knihovny, ovladače a serverové komponenty webových aplikací.

Vložení příkazů[20] Volané webové aplikace obvykle parametry převezmou a předávají je často i operačnímu systému webového serveru a externím aplikacím. Pokud se útočníkovi podaří vložit do těchto parametrů škodlivý kód, může být tento kód vykonán externím systémem v zastoupení webové aplikace.

Nekorektní zpracování chyby Chybová hlášení generovaná webovými aplikacemi mohou útočníkovi poskytnout cenné informace. Záměrným způsobením chyb a analýzou chybových hlášení cíle včetně chyb generovaných ochranným bezpečnostním systémem může hacker získat značný přehled o produktech a technologiích dané instalace. Na základě těchto informací pak může hacker sestavit patřičný exploit (program vytvořený pro zneužití daných slabých míst) a úspěšně napadnout cíl. Útok může mít podobu odmítnutí služby, vyřazení bezpečnostního systému nebo zhroucení serveru.

Nezabezpečené ukládání údajů Webové aplikace často používají kryptografie k ochraně informací o uživatelích. Ukázalo se však, že je poměrně obtížné tyto funkce správně naprogramovat, což může vést k oslabení ochrany. Mezi nejčastější chyby patří nezabezpečené ukládání klíčů, certifikátů a hesel; nedostatečná kvalita zdrojů náhodných čísel; snaha vymyslet nový šifrovací algoritmus; nezajištění podpory pro změnu šifrovacích klíčů a dalších procedur údržby.

[19] Zneužití serveru k odesílání skriptů = anglicky nazývané jako *cross-site scripting - XSS*
[20] Vložení příkazů anglicky nazývané jako *command injection*

Odepření služby Útočníci mohou zcela vyčerpat zdroje webové aplikace, takže legitimní uživatelé nemají k aplikaci přístup. Útočníci také mohou blokovat přístup uživatelů k jejich účtům nebo způsobit zhroucení celé aplikace. Tyto útoky lze vyvolat prostým otevřením dostatečného množství požadavků. Jediný počítač může generovat tolik požadavků, že dojde k spotřebování všech zdrojů webového či aplikačního serveru.

Nezabezpečená správa konfigurace Pro každou webovou aplikaci je kritickým faktorem zabezpečení konfigurace serveru. Existuje celá řada konfiguračních možností serverů, které mají dopad na bezpečnost, a které nejsou bezpečné ve svém výchozím nastavení.

1.8 Zvláštnosti internetových projektů

Voldán [48] uvádí, že u internetového projektu trvajícího čtyři měsíce je 95 % práce dokončeno během prvních dvou měsíců a další dva měsíce potom trvá schvalování konečné podoby. To je častý důsledek mnoha zvláštností, kterými se internetové projekty liší od „klasických" IS/IT projektů. Voldán [48] specifikuje nejvýraznější faktory odlišnosti, které můžeme pozorovat u internetových projektů, a to buď na straně zadavatele, nebo řešitele, a uvádí možnosti, jak se s těmito zvláštnostmi vypořádat.

1.8.1 Význam uživatelského rozhraní

U internetových (nebo intranetových) projektů je uživatelské rozhraní většinou velice důležité - z hlediska designu, z hlediska celkové logiky ovládání, rozmístění ovládacích prvků na stránce, jejich fungování, atd. Především ale proto, že internetová řešení velice často slouží několika zcela rozdílným cílovým skupinám. Internetové technologie také poskytují mnohem více možností pro realizaci řešení.

V případě, že je tento faktor opominut, má vliv především na *prodražení a zdržení projektu*, které je možné eliminovat vytvářením prototypových řešení, tvorbou grafických návrhů, náčrtků zobrazujících principy fungování a posloupnosti obrazovek, vytváření funkčních prototypů. Lidé si dokáží řešení představit teprve až ho vidí. Rozhodně není možné se spoléhat jen na strukturované dokumenty s detailní specifikací požadavků. Nikdy, a to ani v jednoduchých případech, není možné zapomenout na získávání připomínek zadavatele a budoucích uživatelů. Čím později budou zjištěny, tím horší dopady na rozpočet a časový rámec to bude mít.

1.8.2 Účast designerů

Důležitým předpokladem internetových řešení je efektivní komunikace s cílovou skupinou. Jejich neopomenutelnou součástí je proto grafický design. Komunikace s designery však pro řadu projektových manažerů z oblasti informačních technologií představuje značný problém, neboť dříve či později zjistí, že grafici komunikují úplně jiným jazykem a myslí jiným způsobem než programátoři nebo konzultanti.

Tento faktor může mít opět vliv na *zdržení projektu a nesplnění cílů*, neboť design nesplňuje požadavky, i když si strana zadavatele může myslet, že je sdělila naprosto jasně.

V tomto případě je vhodné cíle a cílovou skupinu popsat „v obrazech". I sebepodrobnější písemné zadání není postačující. Specifikace „cílem je prodej zájezdů mladým lidem mezi 25 - 30 roky s měsíčním příjmem nad 30 tisíc" není dostačující. I zde je důležitá vizuální komunikace a schopnost obrazného popisu - vhodným způsobem zadání je také debata o tom, kdo v okolí danému popisu odpovídá, jak se chová, jaké jsou jeho potřeby, co ho zajímá, jak by na něj řešení mohlo působit, co by pro něj osobně mohlo znamenat, apod. Vhodné jsou i příklady a přirovnání, ale i ověření, že cíle projektu jsou skutečně vnímány stejně.

1.8.3 Zadavatelem je marketingové oddělení

V současnosti bývá častěji zadavatelem internetových projektů marketingové než IT oddělení firmy. Lidé z oblasti marketingu přemýšlí o problému úplně jiným způsobem než lidé z IT - to je ostatně důvod, proč vykonávají danou práci. Objemné nabídky nebo rozsáhlé studie nejsou předmětem zájmu této skupiny.

Důsledkem špatné komunikace nemusí být *navrhované řešení tím, co zadavatel chce* a nebo v horším případě tím není to, co dostane. Lepšímu porozumění by měla opět pomoci vizualizace problému. Právě marketingové oddělení bude na vizuální část projektu klást obzvláště velký důraz. Rozsáhlé analytické dokumenty jsou vhodné pro potřeby řízení projektu, ale v tomto případě je třeba vybrat ty nejdůležitější body a převést je do vizuální formy nebo do reálných příkladů. Přesto je třeba, aby byl zpracovatel připraven odpovědět i na detailnější dotazy a to především na ty, které ze svého pohledu nepovažuje za příliš důležité. Je žádoucí, aby si obě strany ujasnily *způsob, jakým mají být informace prezentovány.*

1.8.4 Přechod projektu v průběžný rozvoj

Projekt se obvykle definuje jako řešení nějakého přesně specifikovaného problému nebo zadání v přesně daném časovém rámci a s přesně určeným rozpočtem. Od toho se také odvíjejí metody jeho úspěšného řízení. Internetové projekty však plynule přecházejí do fáze předem přesně nespecifikovaného, průběžného rozvoje na základě požadavků uživatele nebo zadavatele[21].

V případě, že se tento faktor v projektu vyskytne, je to v prvé řadě důsledek špatného řízení projektu, z něhož následně vyplývají *neshody ve finančních otázkách, nespokojenost s dodávaným řešením, nespokojenost se službami.* Pro úspěšnou realizaci projektu je třeba vytvořit systém řízení požadavků. Z tohoto důvodu jsou internetové projekty velice náročné na řízení změn, které mají samozřejmě dopad na rozpočet a časový rámec projektu a na řízení požadavků. Tento systém by neměl být v žádném případě podceňován. U projektů menšího rozsahu jsou občas podobné formalizované postupy odsuzovány jako zbytečná byrokracie. Důležité je proto hned od začátku systém nasadit a prosazovat jeho dodržování u všech zúčastněných.

Systém by měl zajistit především podchycení všech požadavků, podrobení každého z nich schvalovacímu procesu, sledování stavu realizace schválených požadavků, a v neposlední řadě i evidenci, zda zadavatel řešení požadavku akceptoval nebo ne.

[21] Toto je přisuzováno značným možnostem internetových řešení, a proto k nim lidé mají mnohem více připomínek, než ke standardním počítačovým systémům

1.8.5 Nedefinované referenční prostředí

V případě kvalitně řízeného projektu informačního systému je součástí smlouvy nebo dokumentace specifikace tzv. „referenčního prostředí". Dodavatel se v ní zavazuje k zajištění funkčnosti dodávaného řešení na daném hardware a software. Internet je ovšem v tomto případě specifický - uživatelé používají různé prohlížeče, různé způsoby připojení, nacházejí se na různých sítích, apod. To představuje větší problémy související s testováním vytvořených řešení.

Jestliže není náročnější testování zohledněno při plánování projektu, může tento faktor zapříčinit především *zdržení projektu*. Proto je nutné s uvedeným faktorem počítat již na začátku. Je třeba stanovit požadavky na fungování vytvářeného řešení, včetně konkrétních a detailních specifikací. Otestování a odladění systému takovým způsobem, aby nadefinované požadavky splňoval, bude vyžadovat určitý čas, který je potřeba započítat do celkových nákladů projektu.

1.8.6 Návrh systému

Internet je tvořený množstvím nejrůznějších technologií. Aby bylo možné navrhnout funkční, uživatelsky přívětivé, spolehlivé a bezpečné internetové řešení, ukazuje se, že je třeba jeho technologické stránce porozumět mnohem více, než jak je to běžné při návrhu podnikových informačních systémů. V opačném případě mohou být navržené věci velice pracné, drahé na realizaci, v krajních případech dokonce nerealizovatelné.

Tento faktor má opět vliv na *prodražení projektu*. Posouzení tohoto faktoru je plně v kompetenci zhotovitele, který také nese následky případného neúspěchu. V tomto případě by měl konzultant spolupracovat také s technology, aby měl povědomí o tom, co je možné prostřednictvím Internetu realizovat.

1.8.7 Rozpočet zadavatele

Omezený rozpočet není pouze specifikem internetových projektů, ty jsou však z tohoto úhlu pohledu opět určitým způsobem odlišné. Jak již bylo zmíněno, jejich zadavatelem je velmi často marketingové oddělení firmy. Marketingové rozpočty stále ještě reflektují „tradiční" reklamní model obchodu - reklamní agentury získávají prostředky z provize za prodej médií, a z ní pokrývají i náklady na vytvoření toho, co se v nich má prezentovat. Většina marketingového rozpočtu firmy jde proto na média - jenomže v případě Internetu představuje hlavní část nákladů vytvoření vlastního řešení, nikoliv médium jako takové. Internetová řešení se sice dají často zrealizovat velice levně, avšak téměř vždy na úkor kvality, spolehlivosti a především bezpečnosti.

Pokud si zadavatel vymezí nedostatečné prostředky na internetové řešení, může nalézt zhotovitele, který bude jeho poptávku akceptovat, ale pro zadavatele to bude vždy *na úkor spolehlivosti řešení*, což ani nemusí zjistit. Podrobnější plánování finančních nároků internetových aktivit a struktura rozpočtu určených na jejich financování může pomoci předejití tomuto nežádoucímu stavu.

1.9 Návrh webových aplikací

Kadlec [18] připouští, že pro projektování internetových systémů softwarově inženýrské techniky a metodologie přestávají v současnosti pomalu vyhovovat. Zejména s ohledem na rychlost vývoje zmiňuje agilní přístup[22] k vytváření internetových projektů. Uvádí základní odlišnosti a specifika internetových projektů v konfrontaci s běžnými (ve smyslu „neinternetovými") aplikacemi. S Voldánem [48] se shoduje v *nutné specifičnosti vývojového týmu*, který musí zahrnovat i netradiční profese a role, např. grafiky, Flash programátory, reklamní a propagační textaře apod. Je vyloučeno, aby jednotliví členové týmu byli izolováni: u webové aplikace ovlivní rozhodnutí přijaté jedním členem ostatní mnohem více. Prolínání činností navíc znemožňuje jednoznačné stanovení hranic odpovědnosti. U některých projektů je navíc nutné zajistit i vhodný aktivní marketingový web, i na tuto činnost se vyplatí mít v týmu experta.

Rozdíly Kadlec [18] spatřuje právě v *předpokládané budoucnosti aplikace*. U webových projektů je téměř jisté, že se budou postupem času vyvíjet, rozšiřovat, vylepšovat a upravovat; budou přibývat funkce a měnit se vzhled. Návrh webové aplikace by měl být proveden s vědomím této skutečnosti; změnit grafický vzhled website by nemělo vyžadovat měsíc práce. Obdobnou odlišnost uvádí i Voldán [48] u faktoru plynulého přechodu vývoje projektu v rozvoj.

S Voldánovým [48] nedefinovaným referenčním prostředím se Kadlec [18] shoduje v otázkách testování, platforem a prohlížečů. Zásadní rozdíl mezi internetovými a běžnými aplikacemi spatřuje Kadlec [18] *v přístupu k testování*. Testeři webových aplikací musí ovládat mnoho neotřelých postupů, např. testování rozvržení stránky, grafického návrhu, barev na obrazovce, rozlišení. Jedná se o oblasti, které lze otestovat výhradně okem, automatizované testy nepřicházejí v úvahu. Mnoho klasických projektů je dodáváno přímo pro jednu konkrétní platformu a požadavek multiplatformní podpory se řeší dodáním více verzí. Naproti tomu u internetových projektů existuje fyzicky pouze jedna verze webové aplikace, která musí podporovat několik prohlížečů běžících na několika operačních systémech.

Vývoj webové aplikace vyžaduje podle Kadlece [18] kladení značného důrazu na kreativní *návrh uživatelského prostředí*. V prostředí Internetu, kde konkurence je vzdálena pouze jedno klepnutí myší, musí aplikace rychle zaujmout svým vzhledem, v opačném případě zapadne do šedého průměru a nenajde příliš uživatelů. Stan Ward a Per Kroll uvádějí v materiálu o vývoji webových aplikací při použití metodologie RUP[23] [49], že nikdy předtím nebylo tak důležité uživatelské rozhraní aplikací, jako je dnes na webu.

Specifikum propracované navigace zmiňuje Kadlec [18], přičemž Voldán [48] jej opomíjí. Webová aplikace, která chce působit profesionálně, by měla dodržovat konzistenci ovládacích a *navigačních prvků*; měla by mít jasně definovanou koncepci a řídit se jí napříč všemi stránkami. U běžných aplikací je konzistence zajištěna implicitně např. použitím standardních systémových ovládacích prvků; požadavek kreativního grafického návrhu nutí vývojáře zvolit u webových aplikací vlastní ovládací prvky, takže konzistence musí být vědomě a cíleně budována.

[22] Nejznámější agilní metodikou je pravděpodobně *Extrémní programování* od Kenta Becka
[23] RUP = **R**ational **U**nified **P**rocess

Návrh a tvorba webových aplikací by měla být *orientována především na zákazníka (uživatele)*. Tento odlišný přístup k tvorbě aplikací popisuje Kadlec [18] následovně: Při vývoji klasické aplikace, typicky na zakázku pro konkrétní společnost, je důležité vysledovat pracovní procesy probíhající v dané společnosti a vytvořit aplikaci v souladu s nimi, jinak si koncoví uživatelé (z řad zaměstnanců společnosti) na používání aplikace nezvyknou. U webové aplikace je naopak důležité rozbít představu, že aplikace musí sledovat firemní procesy. Aplikace musí především vycházet z potřeb uživatelů, které např. nezajímá, jak je společnost interně rozdělena na oddělení. Není-li toto dělení intuitivní a logické i pro okolní svět, neměl by web být podle něj členěn.

Primárním cílem webu by nemělo být úsilí za maximálním počtem unikátních uživatelů; ti nejsou dlouhodobě přínosní. Důležitější jsou stálí uživatelé, kteří se na web vracejí. Agentura Forrester Research provedla výzkum [16], v němž se zeptala 8900 uživatelů Internetu na důvody, proč se vracet na některé weby. Výsledkem studie bylo zjištění, že lidé po webu požadují především 4 skutečnosti:

- kvalitní obsah,
- snadné používání,
- častou aktualizaci obsahu,
- minimální dobu nutnou k zobrazení.

Jacob Nielsen, světově uznávaný odborník v oblasti návrhu webu, ve své knize Web.Design |37| přidává k uvedeným čtyřem bodům další tři kritéria:

- uspokojení potřeb zákazníka (nestačí poskytovat kvalitní obsah; obsah musí mít vůči zákazníkovi určitou relevanci a musí mu poskytnout to, co hledá),
- unikátnost online média (způsob prezentace musí odpovídat charakteru online média),
- firemní kultura v souladu s obsahem webu (celá firma musí web přijmout a stát za ním).

Podrobnosti o splnění těchto kritérií, stejně jako doporučené konkrétní postupy návrhu website spadají podle Kadlece [18] do oblasti webdesignu a navigace. Některé metodologie, definují vlastní kroky k dosažení uvedených kritérií. Snaha o uspokojení potřeb zákazníka (přesněji různých potřeb různých zákazníků) může být realizována např. pomocí několika skupin případů použití[24], které jsou vytvářeny zvláště pro různé skupiny uživatelů.

1.10 Vývojový proces internetového projektu

Vývojovým procesem můžeme pojmenovat obecný postup, který je možné použít při tvorbě a aktualizaci internetových projektů, který má za cíl ušetřit čas a energii potřebnou k dosažení vytyčených cílů. Dobře definovaný postup je přínosný pro obě strany. Zadavatelé

[24]Případ použití je textový popis definující posloupnost akcí prováděných uvnitř systému či systémem, která poskytuje určitou hodnotu uživateli systému. Tento koncept používá např. Rational Unified Process.

(klienti) budou vědět, co mohou očekávat, co po nich bude během realizace projektu vyžadováno, mají-li být naplněny jejich požadavky a očekávání, a zpracovatel má jednoznačně definovaný postup, který bude použit při tvorbě projektu. Jedná se o průběh interaktivní, to znamená, že se opakuje a jeho průběh má vzestupnou a klesající tendenci.

Duyne [7] vývoj internetového projektu rozděluje do sedmi kroků (viz. obrázek 1.6):

1. *Určení.* Porozumění cílovým zákazníkům a jejich potřebám; vytvoření konceptu obchodních cílů a záměrů zákazníka.
   Výstupy tohoto kroku: Dokument o analýze zákazníků, Dokument o obchodní analýze (obchodní plán, analýza soutěže, měření úspěšnosti), Dokument o požadavcích (popis projektu, seznam záměrů, scénářů a panelů akcí, seznam navržených prvků, obecné cíle designu, měření).

2. *Průzkum.* Vytvoření několika hrubých návrhů webových stránek, ze kterých bude jeden nebo několik vybráno pro další postup.
   Výstup tohoto kroku: Středně kvalitní mapy webových stránek, panely akcí a schémata.

3. *Odladění.* Vyladění navigace, rozvržení a průběhu vybraného návrhu.
   Výstup tohoto kroku: Mapy webových stránek střední a vysoké kvality, panely akcí a schémata.

4. *Produkce.* Vývoj plně interaktivního prototypu a specifikace designu.
   Výstupy tohoto kroku: Dokument o designu, Interaktivní prototypy, Technická specifikace, Vodítka designu, Šablony webových stránek.

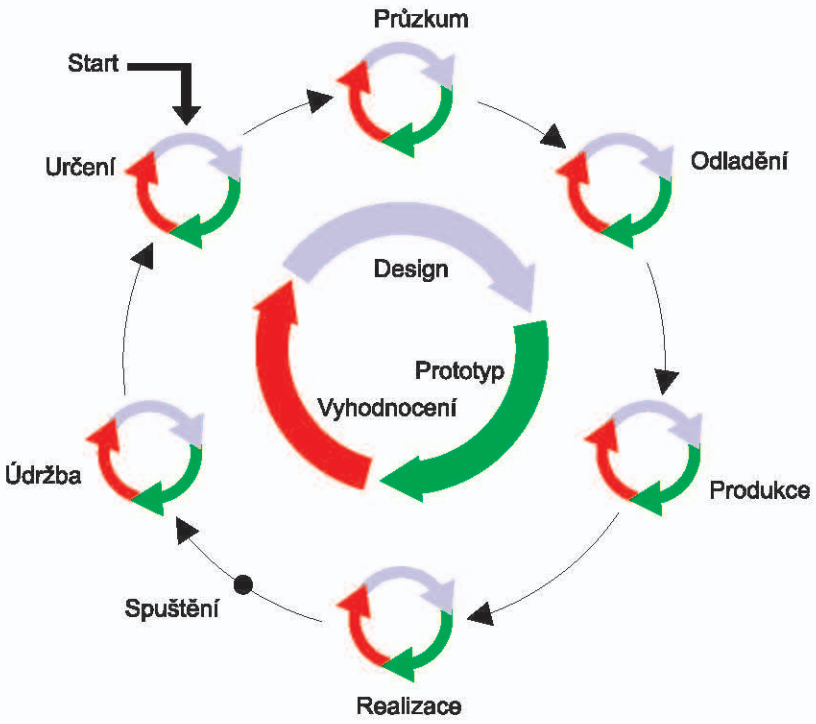
5. *Realizace.* Vývoj kódu, obsahu a obrázků určených pro Internet.
   Výstupy tohoto kroku: Kompletní webové stránky, Dokument o údržbě, Plán testů, Aktualizace.

6. *Spuštění.* Zpřístupnění webových stránek zákazníkům a uživatelům.

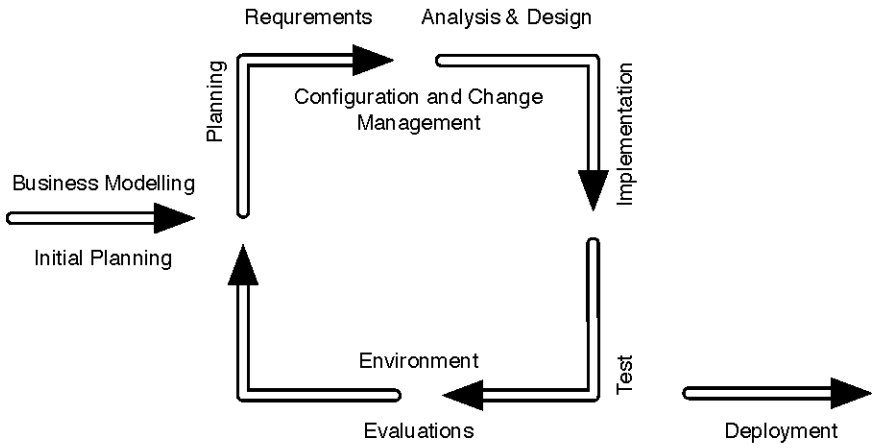
7. *Údržba.* Podpora existujících stránek, sběr a analýza dat vzešlých z měření úspěšnosti stránek, příprava nových doplňků designu.
   Výstupy tohoto kroku: Periodická měření, Zprávy o chybách, Periodické zálohování.

Proti výše uvedenému postupu můžeme postavit například klasický *Rational Unified Process (RUP)*. Jedná se o celosvětově používaný, velmi propracovaný, robustní a komplexní přístup k vývoji a k životnímu cyklu software. Kadlec [19] uvádí 6 obecných praktik, které RUP definuje. Tyto praktiky mají při svém použití zajistit efektivnější vývoj, propracovanější řízení kvality a lepší výsledky:

- *Iterativní vývoj softwaru*: fundamentální problém tradičního, sekvenčního přístupu spočívá v tom, že před sebou tlačí (často dosud netušená) rizika, jejichž odstranění je postupně čím dále dražší. Naproti tomu v iterativním a inkrementálním přístupu (viz obrázek 1.7), který vychází z Bochmova spirálového modelu, dochází k detekci rizik průběžně. Mezi další výhody iterativního vývoje patří snazší správa změn, lepší znovupoužitelnost, dokonalejší přiblížení k zákazníkovi, apod.



Obrázek 1.6: Proces vývoje internetového projektu podle Duyne [7] využívá cyklického návrhu



Obrázek 1.7: Iterativní a inkrementální přístup k vývoji podle Kadlece (RUP) [19].

- *Správa a řízení požadavků*: požadavky jsou typicky dynamické a je nutné počítat s tím, že se v čase mění (syndrom *IKIWISI* I Will Know It When I See It „budu to vědět, až to uvidím"; s nadsázkou vyjádřená neschopnost zákazníka specifikovat dopředu všechny požadavky na vyvíjený produkt).
- *Použití komponentové architektury*: je-li vyvíjený systém otevřený k použití jiných komponent, může být vývoj efektivnější; znovupoužití hotové komponenty znamená podstatnou úsporu zdrojů. RUP poskytuje metodickou a systematickou cestu návrhu, vývoje a ověření správnosti architektury. Nabízí šablony, architektonické styly a pravidla designu.
- *Vizuální modelování softwaru*: model je zjednodušením reality; je vytvářen za účelem dokonalejšího porozumění systému; u rozsáhlejších projektů také proto, že není možné pochopit systém v celé jeho celistvosti. Použití standardního modelovacího jazyka (UML) znamená zjednodušení komunikace uvnitř týmu, zlepšení konzistence projektu, zvýšení pravděpodobnosti správného uchopení systému.
- *Průběžné zajišťování a ověřování kvality*: nalezení a odstranění problému je mnohonásobně dražší po předání produktu než v úvodních fázích vývoje.
- *Řízení změn*: neřízené změny (ať již při vývoji nebo i po předání) vedou k chaosu.

I v tomto případě je patrné, že klasická metodika RUP je samozřejmě použitelná i pro internetové projekty, ovšem není v ní kladen takový důraz na sociální, uživatelské, marketingové a obchodní faktory, které nelze u internetových projektů opomíjet. Jejich analýza je především u internetových prezentací, elektronických obchodů, internetových portálů, veřejných webových aplikací obzvláště důležitá, jelikož se významně podílí na dosažení cíle, se kterým je internetový projekt tvořen.

Duyne [7] ve své metodice příliš nezdůrazňuje jeden důležitý faktor, který by měl být nedílnou součástí každého softwarového produktu a tím je *testování* výsledného produktu. Duyne [7] zařazuje testování do kroku realizace, ale testování by mělo být samostatným krokem, který se výrazně vyznačuje svojí cykličností. Duyneova metodika je vhodnější pro

statické internetové projekty (statické webové prezentace), kde implementační chyba nemusí mít tak závažné následky jako u dynamických aplikací.

Posloupnost a případné opakování jednotlivých fází začínajících specifikací požadavků a konče implementací a provozem aplikace je označováno jako **životní cyklus**. Varianty životních cyklů, které se při vývoji software využívají jsou shrnuty v následujících odstavcích [38]:

- *Okamžitý vývoj* - je aplikován především programátory, kteří se nechtějí zdržovat a zatěžovat zbytečnou analýzou. Ačkoliv se na první pohled může zdát, že takto není možné solidně pracovat, má i tento styl v tvorbě software svoje místo. Podobným způsobem je možné vytvořit malou aplikaci, kde není nutné se detailně zaobírat podrobným systémovým návhrem, ověřovat si předpoklady, apod. Bohužel je stejným způsobem možné vytvořit i poměrně rozsáhlou aplikaci. Problémy vyniknou a potíže vzniknou v okamžiku požadavků na určitou stabilitu a spolehlivost, při snaze systém nějak udržovat, doplňovat a rozvíjet. Protože chybí nejen dobrá dokumentace, ale chybí také určitý pořádací princip, či vedoucí myšlenka, koncept, metoda, bývá obtížné pochopit takové dílo i pro samotné tvůrce, natož pro ty, kteří se na jeho vzniku nepodíleli.

- *Vodopádový model* - se používá již mnoho let. Je považován za velmi dobré vodítko pro tvorbu software. Jeho předností je snadné plánování rozpočtu, možnost kontroly postupu etap projektu a celkovou přehledností průběhu prací. Určité případy však vyžadovaly zbytečně časově náročné práce a proto se hledaly takové vlastnosti, které by tento problém eliminovaly. Tak se objevil iterativní model.

- *Iterativní model* - vznikl doplněním vodopádového modelu jako odezva na potřebu urychlení procesu projektování. Urychlení spočívá ve dvou skutečnostech:

  1. Fáze analýzy je podpořena souběžným vývojem prototypu, který slouží k ujasnění zadání metodou ověření funkčnosti zamýšlené aplikace.
  2. První „ostrou“ verzi software je možno sestavit rychleji, protože se zde předpokládá, že možné chyby a problémy, které vzhledem k rychlosti vytvoření může mít, poslouží ke zpřesnění analýzy při následném opakování a vzniku další verze. Pro iterativní model je typické jedno až dvě opakování.

  Iterativní model je výhodnější pro tvorbu software tehdy, když na začátku projektování nelze zcela přesně poznat zadání do všech podrobností.

- *Spirální model* - ze všech modelů nejlépe odpovídá potřebám objektově orientovaného projektování. Je to upravný iterativní model, ve kterém je spojen prototyp s jednotlivými verzemi výsledného „ostrého“ software. Na rozdíl od iterativního modelu se tedy prototyp nezahazuje, protože je vytvořen stejnými nebo kompatibilními prostředky jako cílová verze. V tomto modelu dokonce v ideálním případě není žádná určitá finální verze, neboť každou verzi aplikace je potenciálně možné použít a současně vzít jako podklad pro zadání a nastartování případného dalšího vývojového cyklu. Typický počet opakování 2 až 3 je shodný s iterativním modelem. Jeden cyklus spirálního modelu v OOP má fáze zadání, analýzy, designu (návrhu), implementace, testování

a provozu. Fáze zadání a analýzy se označují jako stadium expanze. Zbývající fáze od návrhu přes implementaci k testování a provozu se označují jako stadium konsolidace. V těchto etapách se model postupně stává fungujícím programem. V tomto stadiu dokonce dochází k tomu, že některé z návrhů, nápadů a myšlenek z předchozího stadia bude nutno vypustit vzhledem k časovým, kapacitním, implementačním nebo intelektuálním schopnostem.

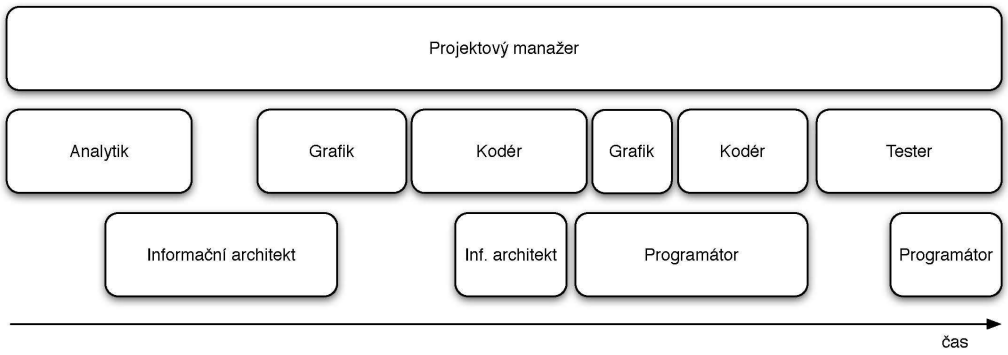
1.11 Projektový tým

Při řízení jakéhokoliv projektu je nutné sledovat tři klíčové faktory: *čas*, *rozpočet* a *rozsah*. Následující část se nebude věnovat rozsahu projektu ani časovým milníkům, které projekt limitují, ale zaměří se na *rozpočet*. S určitým zjednodušením je možné rozpočet určit na základě nákladů na práci strojů, práci lidí a ostatní související náklady. Využívání strojů, počítačů, je v oblasti webových aplikací zcela neodmyslitelné a jejich podíl na celkovém rozpočtu projektu je ve většině případů možné přepočítat víceméně procentně z režijních nákladů provozu firmy. Počítačem musí být vybaven každý pracující jedinec, v současné době nejsou již znatelné ani žádné zásadnější rozdíly mezi vybavením jednotlivých pracovních stanic nebo notebooků, tj. hardware. U používaného softwaru je již možné sledovat určité rozdíly ve vybavení pro grafické účely - kreslící studia, nástroje pro zpracování fotografií, animace a vizualizace, jejichž licence dosahují desítek tisíc korun. Opačným extrémem může být vybavení softwarem poskytovaným zdarma, a to od operačního systému až po vývojové prostředí. Software se ale většinou nepořizuje na míru řešenému projektu, ale pořizuje se pro vícenásobné použití na různých projektech. Proto je možné i tento náklad používat svým způsobem za paušální, neboť software se svým používáním neopotřebovává jako například stroje nebo jiné pracovní pomůcky.

Pokud se zaměříme na lidské zdroje, rozpočet je možné stanovit podle ceny daného zdroje a času po jaký bude alokován na projektu. V souvislosti se zvláštní povahou této „veličiny“ neplatí pouze standardní vynásobení, protože lidé mají oproti strojům omezené možnosti přesouvání se mezi různými pracovními úkoly v čase. Přesouvání členů projektového týmu mezi různými projekty v čase se většinou projeví na snížené produktivitě práce a obecně delším čase na zapracování se zpět do problému. Proto je vhodné tyto zdroje efektivně plánovat, aby mohly pracovat pokud možno v co nejdelších pracovních blocích.

1.12 Role v projektu webové aplikace

Stejně tak jako je tomu v jiných oborech a pracovních týmech, tak i v případě webových aplikací se vlastní tvorby aplikace účastní různé profese, které mají na projektu, a jeho úspěšné realizaci, svůj podíl a odpovědnost. V očích laiků se může zdát, že zde podstatný rozdíl není, protože všichni pracují s počítači a i výstup jejich práce se může zdát na první pohled *vizuálně stejný*. Pro rozlišení jednotlivých profesí je nutné mít alespoň základní přehled ve webových technologiích a používat ustálené označení těchto profesí, rolí. Protože je tato práce zaměřena především na změny ve vývoji webových aplikací, tedy na práci těchto rolí, je nutné jednotlivé role popsat a sjednotit jejich pojmenování.



Obrázek 1.8: Zapojení jednotlivých rolí v průběhu projektu

1.12.1 Analytik

Význam role analytika je především v počátcích v projektu, kdy připravuje zadání, komunikuje se zadavatelem, vytváří přípravné dokumenty pro vlastní přípravu a tvorbu aplikace. Role analytika v projektu může být dělena dle specializace na určité části aplikace (systémový návrh, provedení GUI, informační architektura, komunikační model, marketingový model) a toto rozdělení vždy závisí na schopnostech nominované osoby, na možnostech a pravidlech organizace.

Výstupem práce analytika je dokument analýzy nebo specifikace, diagramy představující návrh fungování celé, nebo části připravované aplikace. Diagramy mohou představovat např. databázové relace, procesní návrh, návrh datových toků, případy užití, stejně tak jako rozvržení grafických elementů a způsob ovládání aplikace. Hlavní část práce analytika je provedena především na počátku projektu, etapy nebo iterace - záleží na preferovaném způsobu vývoje[25]. V průběhu realizace pak působí spíše jako konzultant a operativně ve spolupráci s týmem doplňuje nebo upravuje části dokumentace tak, aby byla i po dokončení aplikace konzistentní.

Při tvorbě webové aplikace je úkolem analytika především prvotní sběr požadavků a vyjasnění všech očekávání od budoucího produktu. Po ukončení této části následuje návrh funkčního modelu a transformace sebraných požadavků do vlastností díla. Během této činnosti by měly být vyjasněny mnohé nepřesnosti a neurčitosti původních požadavků. Následují fáze tvorby systémového návrhu (viz např. [38]), s cílem navrhnout vnitřní fungování aplikace, zahrnujíce definici datového, relačního, objektového modelu, apod. a taktéž velmi důležitá příprava informační architektury. Existuje množství způsobů jak informační architekturu pojmout (viz např. [20]), výstupem musí každopádně být minimálně zadání rozvržení uživatelského rozhraní pro grafika např. v podobě wireframu (viz. obrázek 1.9).

1.12.2 Grafik

Narozdíl od analytika je účast grafika specifická pro vývoj webových aplikací. Při tvorbě klasických desktopových aplikací není ve většině případů grafik v projektovém týmu zastoupen. U webových aplikací se nepoužívají systémové elementy pro ovládání, nepoužívá se

[25] Např. vodopádový, spirálový model (popsané v oddílu 1.10), případně různé formy agilního vývoje.

standardní horizontální menu, neexistují systémové knihovny předdefinovaných prvků uživatelského rozhraní, ani obecné zásady designu aplikací (jako jsou například Windows/MacOS User Interface Guidelines[26]). Aplikace běží v jednom okně, je založena na sekvenčním přechodu mezi jednotlivými kroky. V neposlední řadě webové aplikace mnohdy „prodávají" a proto musí být pro uživatele atraktivně a snadno ovladatelné. Všechny toto faktory musí grafik při návrhu webové aplikace brát v úvahu. Na druhou stranu je grafik ovlivňován právě faktory webové prezentace: grafika nesmí výrazně zpomalovat načítání stránky, musí počítat s tím, že uživatel si může měnit velikost okna, měnit velikost písma, jednotlivé elementy na stránce mohou měnit svou velikost podle obsahu.

Grafik by měl při výtvarném zpracování vycházet z předem navrženého modelu aplikace a způsobu ovládání. V praxi se k tomuto účelu nejčastěji využívá tzv. *wireframe*. Wireframe zpravidla představuje prototyp každé unikátní webové stránky a neobsahuje žádnou grafiku, barvy, loga ani obrázky (viz obrázek 1.9). Wireframe připravuje zpravidla informační architekt ve fázi analýzy GUI. Grafik se tak může soustředit na výtvarnou stránku a srozumitelnost grafického vyjádření ovládacích prvků. Výstupem práce grafika je obrázek, který v sobě simuluje i generovaný obsah, který následně bude zajišťován programovým kódem. Obrázek je statický a tento výstup již v sobě zahrnuje práci analytika uživatelského rozhraní a je pak jednoduše prezentovatelný zadavateli, který grafický výstup schvaluje. Ve fázi diskuse nad grafickým návrhem může grafik relativně snadno provádět úpravy vzhledu, protože pracuje pouze se svým výstupem, který nemá žádné další návaznosti v projektu. Při změnách grafiky v průběhu projektu musí grafik intenzivněji spolupracovat zejména s *kodérem* a reflektovat, jak je webová aplikace vytvořena, aby navrhované změny neznamenaly zásadní zásah do celé struktury aplikace.

1.12.3 Kodér

Označení kodér je zřejmě nespornější označení role, neboť se pro ni v praxi používá mnoho různých označení: HTML specialista, HTML programátor, webdeveloper, CSS specialista, Front-End Web Developer, apod. V této práci je *kodérem* tedy míněn HTML/CSS specialista, který zná současné značkovací jazyky, tj. především HTML, XHTML, XML, zároveň ovládá kaskádní styly CSS a dokáže převést grafický návrh do podoby statické stránky. Zároveň u kodéra očekáváme znalosti související právě s HTML kódem, tj. znalost SEO optimalizace zdrojového kódu, nároky různých prohlížečů a cílových zařízení (webový prohlížeč, PDA, mobilní telefony, čtečky, apod.). V tomto ohledu musí kódér zvládat alespoň základní práci se softwarem pro grafickou tvorbu, neboť musí z tohoto formátu převést grafický návrh do podoby webové stránky. Z tohoto důvodu na kodéra většinou padne zodpovědnost za optimalizaci použité grafiky tak, aby velikosti stránek nepřesahovaly standardní meze dané především nároky na rychlé zobrazení stránky, tj. časem potřebným pro stažení všech jejich komponent.

Obecně lze tedy práci roli kodéra spatřovat v transformaci grafického materiálu na vstupu, do sady statických XHTML stránek s použitím formátování na úrovni CSS, případně jiného stylopisu na výstupu. Předávaný výstup by měl být již ověřen na různých

[26] `http://developer.apple.com/documentation/userexperience/Conceptual/AppleHIGuidelines/`

Obrázek 1.9: Wireframe - návrh rozložení obsahu webové aplikace

platformách a validní dle veřejných specifikací.

1.12.4 Programátor

Aplikace funguje, jestliže reaguje na vstupy od uživatele, zobrazuje relevantní výstupy a nabízí další možnosti pokračování v práci. A právě tyto požadavky musí ve webové aplikaci zajistit *programátor*. Pro svoji práci využívá různé programovací jazyky (PHP, ASP/.NET, Java, Ruby, Perl, Python, ...). Volba jazyka je závislá čistě na preferenci nebo znalosti programátora a lze říci, že nemá zásadní vliv na naplnění cílů projektu. Od programátora se očekává práce s datovými zdroji, zpracování vstupů uživatele, ošetření chyb, ovládání serverových částí, komunikace s okolními systémy, příprava dat pro uživatelské rozhraní (UI) a obsluha jeho chování, apod.. Na vstupu jeho práce se tedy nachází HTTP požadavky (HTTP requests) a na výstupu data vložená do stránek. HTTP požadavky vytváří internetový prohlížeč na základě vstupů od uživatele (provedených akcí).

Konkrétní způsob vedení vývoje záleží na možnostech firmy, volbě jazyka, serverové technologie nebo frameworku[27]. Programátor by měl být zdatný především v porozumění systémovému návrhu systému, reflektovat trendy vývoje (např. objektově orientované programování, principy model-view-controller, routing vstupů, apod.). Kromě požadavků na konkrétní funkčnost aplikace musí programátor vyřešit i dílčí úlohy, které nejsou exaktní součástí zadání - např. rychlost zpracování, bezpečnostní faktory (ochrana proti průnikům, napadením aplikace), ošetření chybových stavů a postup jejich nápravy, evidenci činnosti aplikace (logování), atd.. Pokud bychom porovnali programátora desktopové a webové aplikace, je programování webové aplikace náročnější především na úrovni ošetření vstupů a obsluhy uživatelského rozhraní. To je dáno především benevolencí internetového prohlížeče, kde mohou být funkčnosti rozhraní modifikovány různými nastaveními, doplňky nebo dokonce samotnými uživateli. Desktopový programátor např. nastaví pravidlo na textové políčko, že maximální počet znaků, které bude možné do něho zapsat je 5, a že to mohou být pouze čísla a může se spolehnout, že vstupní parametry budou tato omezení skutečně splňovat.

V oblasti webové tvorby je téměř nemyslitelné, že by programátor ovládal pouze svůj programovací jazyk a nikoliv jazyk HTML. Části programového kódu generují přímo elementy stránky, a proto se většině případů setkáme s použitím HTML značek přímo v kódu aplikace. Vstupní parametry jsou z formulářů převáděny do HTTP požadavků a proto i když by uživatelské rozhraní umožnilo omezit vstupy pouze na určitý typ nebo délku, je možné je změnit v HTTP požadavku. Proto musí webový programátor vždy striktně validovat vstupní data.

Role programátora může být v závislosti na typu organizace dále dělena. Při organizaci větších týmů softwarového vývoje může být dokonce účelné zavedení celých oddělení jednotlivých profesí (z historických důvodů se tyto profese označují souhrnně jako *programátor*) [38]: vedoucí programátor, ideový programátor, systémový programátor, výkonný programátor, specialista na jazyk, tester, oponent, dokumentátor a knihovník.

Při tvorbě webových aplikací se role programátorů mohou podrobněji členit dle techno-

[27] *Framework* je softwarová struktura, která slouží jako podpora při programování, vývoji a organizaci jiných softwarových projektů. Může obsahovat podpůrné programy, knihovnu API, návrhové vzory nebo doporučené postupy při vývoji.

logie, kterou ovládají na *databázové specialisty, JavaScript programátory* nebo *Flash programátory.*

1.12.5 Tester

Oblast informačních technologií je právě v oblasti této role velice specifická. Jedná se o roli a činnost, která je implementována i v jiných odvětvích než je IT a nikde není tak podceňována jako právě v IT. Testování se např. u webových aplikací provádí v mnohých případech pouze během vývoje, integrační testování před předáním díla zákazníkovi nebo spuštěním do ostrého provozu se většinou neprovádí vůbec, nebo jen omezeně. Tento fakt je dán především způsobem řízení a plánováním projektů. Testování je často opomíjeno ve všech klíčových faktorech projektového trojimperativu (viz. obrázek 1.1), tedy není zahrnováno do *rozsahu projektu*, z toho důvodu na něj v projektu nejsou alokovány potřebné *prostředky* ani *časová rezerva*. Případně se perioda určená pro testování rozpustí na pokrytí zpoždění projektu. Tester dostává hotovou aplikaci k ověření správné funkčnosti. Zpravidla ověřuje fungování aplikace vůči požadavkům zákazníka a zpracované analýze či dokumentaci. Pro vyhodnocení testů si připravuje sadu *testovacích scénářů*, ve kterých jsou definovány testované vlastnosti, sled jednotlivých kroků v aplikaci k ověření testované vlastnosti a očekávané chování aplikace. Převedeno do jednoduché úlohy – jaká data mají být na vstupu a jaká mají být na výstupu? Jak je patrné z obrázku 1.8, je role testera do projektu alokována až v jeho závěrečných fázích a jeho prací se tak uzavírá realizační část projektu. Obrázek 1.8 nepokrývá iterační část testování, kdy jsou chyby identifikovány, opravovány a znovu testovány. Této fáze se účastní zástupci všech projektových rolí v závislosti na jejich podílu na zjištěné chybě.

1.12.6 Správce

Role správce je oproti výše zmiňovaným rolím poměrně specifická. Jakmile je aplikace dokončena a předána do ostrého provozu, vznikají další požadavky na doplnění obsahu, určité funkčnosti, úpravu stávajících částí, apod. Standardním postupem by byla identifikace požadavku, jeho rozdělení na dílčí části dle rolí a následné zpracování konkrétně nominovanými osobami. Nicméně v případě webových aplikací se osvědčuje role správce, který by měl být schopen provést údržbářské práce *drobného* charakteru jak na úrovni programu, tak HTML kódu. U úprav dokáže správce identifikovat jejich rozsah a části aplikace, ve kterých bude nutné provést změnu. V případě úprav rozsáhlejšího charakteru, vyžadující systémový postup si tak dokáže požádat o alokaci správných osob. Zavedením a obsazením této role se řeší především flexibilita a rychlost řešení drobných požadavků zadavatele. Tato role nezapadá do systémového řešení projektů, ale jedná se o speciální kombinaci zavedenou ve většině firem zejména z obchodních důvodů. Co je důležité, ať už je správcem jedna osoba, nebo i tým specialistů, přebírají většinou kód napsaný někým jiným a pokud má být aplikace trvale udržovatelná je nutné postupovat systémově, což není vždy možné. V této práci je správcem nazývána osoba, která aplikaci nevyvíjela, ale provádí její rozvoj a správu.

1.12.7 Projektový manažer

Projektový manažer je jedinou rolí, která se účastní z pohledu schématu 1.8 všech fází projektu. Je primárně zodpovědný za specifikaci, vymezení rozsahu, času a alokaci zdrojů, a tedy i dodávku a úspěch celého projektu. Dohlíží na průběh realizace a porovnává dosažené výkony vůči plánu a milníkům, které byly stanoveny. V okamžiku, kdy se některá část začne zpožďovat, nebo je identifikováno navýšení odhadované pracnosti, musí projekt přeplánovat, což s sebou nese další rizika v nedostupnosti zdrojů v nově naplánovaném čase. I přesto že se pracovní alokace jednotlivých rolí na projektu prolínají, představuje nedostupnost jedné role značný problém, neboť není možné pokračovat na navazujících činnostech. Projektový manažer odsouhlasuje vstupy jednotlivých týmů, provádí průběžnou kontrolu kvality výstupů. Mimo řízení vlastního týmu je zodpovědný za komunikaci se zadavatelem, prezentaci výsledků a plnění projektového plánu. Zároveň domlouvá případné posuny projektu a další dopady. Nezbytnou administrativní částí práce projektového manažera je kontrola odpracované práce, reporting nadřízeným orgánům projektu, otevírání a uzavírání projektu, včetně jeho průběžného vyhodnocování.

Práci projektového manažera uvedenou v předchozím odstavci je možné shrnout do následujících činností a odpovědností [38]:

1. **Komunikace se zadavateli.** Manažer předkládá návrhy řešení zadavateli. Každý takový návrh musí obsahovat odůvodnění a časové, finanční a zdrojové nároky.

2. **Plánování projektu.** Úkolem manažera je vypracovat harmonogram projektu včetně jednotlivých etap a jejich návazností a stanovit pravidla, kterými se bude projekt řídit. Je třeba zajistit způsob ověřování a postupy předávání produktu, různé konfigurace, nároky na zaškolení a údržbu.

3. **Řízení rozpočtu** je jednou z klíčových činností manažera. Patří sem mimo nákladů na vlastní tvorbu i náklady např. na techniku, školení a cestování.

4. Úkolem manažera je rovněž **výběr osobností, zformování týmu** pro řešení projektu a jeho složek. Při výběru je třeba respektovat složitost požadavků i rozpočet projektu. Platí obecně uznávaný názor preferovat do klíčových pozic řešitelských týmů skutečné špičky bez ohledu na vyšší náklady.

5. **Kontrola stavu** projektu a schopnost reagovat na problémy, které vznikají při samotné tvorbě díla.

6. **Prezentace výsledků** během a na konci projektu jak zadavatelům, tak i uživatelům.

Kapitola 2

Metodika ETWA

„Nejdříve je třeba se naučit tomu, o čem píšeš, potom je třeba se naučit psát. Na jedno i druhé padne celý život.“
Ernest Hemingway

V této části práce přejdeme již k návrhu vlastní metodiky pro zefektivnění vývoje webových aplikací na základě překážek identifikovaných v oddílu 2.1. Návrh metodiky je v následujících oddílech rozpracován až do úrovně doporučených formátů a dílčích postupů, je implementačně nezávislý a je možné jej aplikovat na vývoj webových projektů na úrovni malých a středních podniků. Metodika je navržena a optimalizována pro webová studia a internetové agentury realizující projekty malého až středního rozsahu. Primárně není určena pro velká (enterprise) řešení, kde je nutné se soustředit především na propracovaný systémový návrh. Práce čerpá z informací a zkušeností získaných spoluprácí se třemi internetovými agenturami včetně Informačního a poradenského centra PEF ČZU v Praze.

2.1 Pohledy projektových rolí

Aby návrh metodiky nevycházel z nereálných předpokladů a bylo dosaženo cíle zefektivnění vývoje webových aplikací, je nutné se na celý proces přípravy, vývoje a následné údržby webové aplikace podívat z různých pohledů. Základním východiskem pro stanovení požadavků na metodiku je identifikace nejčastějších problémů z pohledu zástupců jednotlivých projektových rolí popsaných v oddílu 1.12.

2.1.1 Kodér

Nestálost zadání Ve většině případů je kodér první rolí, která začíná vytvářet finální produkt v cílovém prostředí - tj. webovém prohlížeči. A protože se při optimalizaci projektového plánu dbá především na rychlost dodávky začíná kodér pracovat na vzhledové stránce v okamžiku, kdy ještě zcela nejsou designéry připraveny všechny prototypové obrazovky. Prolínání činností, kdy předchozí fáze ještě zcela není dokončena, není typické jen pro tuto roli, ale dalo by postavit na obecnou úroveň. Protože jsou výsledky práce designéra prezentovány zákazníkovi, dochází v těchto podkladech často ke změnám, které mají mnohdy

zásadní dopady do HTML kódu, který je výsledkem práce kodéra.

Protoypování Kvůli ujasňování funkčnosti aplikace se stále častěji osvědčují *prototypy*, které simulují chování cílové aplikace. Prototyp je v podstatě sada statických stránek, které jsou vzájemně propojeny odkazy, formuláře zobrazují jinou statickou stránku s ukázkou výsledku odeslání formuláře, apod.. Ačkoliv je užitečnost prototypu téměř nezpochybnitelná, konzumuje jeho tvorba čas, který není zanedbatelný, přičemž vytvořený prototyp není v budoucnu využit pro cílové řešení. Na kodéra jsou tak vznášeny požadavky na integraci jisté formy funkční logiky, generování stejných stránek s různým obsahem (simulace výstupů), naplnění konkrétními logickými daty, obrázky, označení formulářových polí, apod. Změny v protypovaných stránkách se pak provádí poměrně složitě, protože mohou představovat změnu v obecné části, která je potřeba zkopírovat do všech stránek prototypu. Z prototypu se přitom pro účely finální aplikace použije jen zlomek.

Nerespektování konvence a optimalizace Jakmile je jednou výstup ve formě HTML/CSS šablon předán programátorům, ztrácí nad ním kodér kontrolu. Protože programátoři většinou nemají takové znalosti HTML/CSS, se často stává, že nerespektují kodérem navržené konvence, nedoplňují požadované údaje (např. popisky obrázků, velikosti elementů, apod.). Samozřejmě v tom nelze spatřovat špatný úmysl, nýbrž různou cílovou orientaci programátorů a kodérů. Obdobně jako konvence jazyka HTML se z programy generovaných stránek ztrácí i prvky optimalizace pro webové prohlížeče, které předepisují, kde je potřeba používat speciální způsob výpisů a zvláštní typy elementů, které z hlediska programátora nemusí být správně pochopeny. Kodér je také velice často specialistou na SEO[1] a proto také zdrojový kód navrhuje s ohledem na pravidla SEO. Tato pravidla musí respektovat i výsledná stránka, která je generována aplikací.

Datová integrace V okamžiku integrace šablon s výstupem aplikace se ze šablony stává „jazykový hybrid"; stránka se neskládá jen ze dvou různých jazyků, resp. jazyka a stylopisu (HTML, CSS), ale jsou do ní vloženy také části kódu programu, který do šablony doplňuje generované výstupy. Mohou to být jednoduché formulace např. PHP, ASP, Perlu, apod. nebo jde o zápisy zcela originální, definované použitým frameworkem nebo šablonovacím nástrojem (např. 1.3.3). V případě, že se ve stránce začnou objevovat i složitější konstrukce (jako například smyčky `for`, `while`, instancování objektů, apod.) je možné, že šabloně přestane kodér rozumět a přestává tak být schopen cokoliv ve stránce měnit, nebo následně doplnit. Změna je možná jen za asistence programátora, který může určité části kódu objasnit, aby mohl kodér navrhnout nejlepší řešení realizace požadované úpravy.

Omezené provádění změn Jak již bylo zmíněno v předchozím odstavci, tak integrace šablony s programem zpravidla ztěžuje, ne-li zcela znemožňuje, budoucí úpravy kodérem bez účasti programátora. Dalším problémem obdobného charakteru je rozdělení stránky na opakující se logické elementy. Programátor rozdělí zdrojový kód dle svého uvážení na dílčí

[1] SEO - *Search Engine Optimalization* - z pohledu kodéra se jedná o uzpůsobení zdrojového kódu a vlastního obsahu stránky pro účely snadného nalezení obsahu vyhledávači.

části, které se budou opakovat na více stránkách. Tyto elementy pak vkládá do každé nové stránky a pouze do prostoru mezi elementy generuje obsah. Kromě faktu, že programátor nemusí vždy HTML kód rozdělit správně, dělí tento krok zdrojový kód do množství dílčích souborů bez zjevné vazby. Tento fakt opět výrazně ovlivňuje možnosti zásahu kodéra do původního kódu.

Účast je vyžadována po celou dobu projektu Obzvláště u webových aplikací již není jediným cílem samotná funkčnost, správnost výpočtů a korektní nastavení algoritmů, ale jde především o komfort při jejím ovládání, o rychlost zaškolení obsluhy a kvalitu a efektivnost práce s ní. Webové aplikace neslouží pouze jako webový přístup k evidenčním, vykazovacím, logistickým a podobným systémům, které jsou nutné pro běh firmy, ale jsou to také přímé obchodní kanály, které prodávají zboží nebo služby a špatně navržené, nebo špatně fungující, uživatelské rozhraní tak může být významným kritériem úspěšnosti tohoto prodejního kanálu, nebo dokonce fungování celé firmy. Proto má uživatelské rozhraní, resp. prezentační vrstva aplikace nesporno důležitost a proto je účast jejího tvůrce – kodéra – většinou vyžadována po celou dobu projektu. Kromě výše uvedených důvodů je tato alokace dána také již dříve popsaným postupem realizace. Kodér odvádí největší část své práce na začátku projektu nebo iterace. Následně převezme hotový kód programátor, který jej doplňuje a rozšiřuje a v těchto případech musí kodér často asistovat a případně dle aktuálně identifikovaných potřeb doplňovat další elementy nebo celé obrazovky. Z toho vyplývá i omezená možnost plánování času kodéra, neboť zmíněné mikro úlohy na více různých projektech jej mohou zatěžovat naprosto nerovnoměrně. V otázce pracovní motivace je pak nutné najít optimální rovnováhu mezi prací na souvislých projektových dodávkách a uvedenými mikroúlohami.

Vzorová data a potřeby programátorů nejsou známy předem Vytvořená šablona v sobě obsahuje velké množství umělc vytvořených dat a hodnot doplněných na místa, která jsou následně nahrazena programovým výstupem. Některé části jsou naprosto evidentní a nepřehlédnutelné - jako je název obrazovky/článku, popis políčka, apod. Ve stránce je ovšem množství hodnot, které nejsou na první pohled vidět, jejichž doplnění nesmí být opomenuto pro správnou funkčnost uživatelského rozhraní aplikace. Takovýmito hodnotami jsou například velikosti a popisky obrázků, identifikátory elementů, způsob formátování seznamů, apod. Bylo by mnohem jednodušší pokud by mohl kodér přímo do stránky vkládat hodnoty sám. Toto bohužel v mnohých případech není možné, neboť kodér neví jakým způsobem budou hodnoty vkládány, což může být z datového pohledu případ jestli půjde o pseudoproměnnou (např. `{#jmeno#}`), hodnotu z prostředí programovacího jazyka (např. `$jmeno`), atribut objektu (např. `$osoba->jmeno`) nebo třeba pole (např. `$fields[$iCounter]['jmeno']`). Kromě vlastní podoby dat není předem známé ani jméno proměnné, ve které se bude daná hodnota předávat – zda-li půjde o `$name`, `$jmeno`, `$jmenoOsoby` nebo třeba `($fist_name . ' ' . $last_name)`.

2.1.2 Programátor

Protože je role programátora v oddílu 1.12.4 definována také jako role databázového nebo javascriptového programátora, v případě tohoto oddílu se jedná pouze na programátora ser-

verového skriptovacího jazyka, který generuje výstupy aplikace a zprostředkovává napojení výstupů na uživatelské rozhraní. Označení je uváděno pouze jako role v jednotném čísle, i když se převážně vždy jedná o skupinu programátorů, kteří na projektu pracují.

Pozdní zapojení do projektu Programátor vstupuje do projektu až v okamžiku, kdy jsou již uzavřeny předchozí fáze analýzy, informační architektury, designu i kódování. I když si většina projektových manažerů tuto chybu uvědomuje a je na ni pravidelně ve zpětných hodnoceních upozorňována, zapojuje tým programátorů do projektu až příliš pozdě. Z toho plyne určitá desorientace v řešeném problému a opožděně zjištěná omezení technického řešení. Programátoři nemají možnost již víceméně jakkoliv zasahovat do návrhu systému a často i kritické chyby v zadání jsou řešeny až ve fázi programování. Tento problém vychází jednak z omezených možností uvolňování vývojářů na fázi analýzy, jejímž cílem není vývoj, který představuje hlavní činnost této role. Je také ovlivněn otázkou finanční - tj. jestli je v rozpočtu započítán dostatečný počet hodin na účast programátorů na přípravných fázích, které jsou primární zodpovědností analytiků a architektů. Na přípravných pracích by totiž programátor působil spíše v roli konzultanta, neboť dle sekvenčního/vodopádového přístupu nemá ještě připraveny všechny vstupy tak, aby mohl vykonávat svoji skutečnou pracovní náplň, tj. programovat.

Špatně připravené výstupy předchozích fází Jako podklady pro zahájení programování jsou předávány analýza a HTML šablony. Obdobně jako je to v případě kodéra a grafika, tak i v případě programátora jsou často zmiňovány problémy s nekonzistencí předávaných výstupů. V šablonách jsou jiné informace, než jsou v analýze, nebo jinak než jak jsou zadávány úkoly. Toto může vyústit ve dva typy situací. Buď si je programátor schopen opravy provést sám, nebo jednoduše chybějící části do výstupů (uživatelského rozhraní) nedoplní, protože je není doplnit kam. Samozřejmě tento případ je v řešitelný hlavně dobrou komunikací uvnitř projektového týmu, ale příčina problému může být i v tom, že informace o zahrnutí/nezahrnutí určité informace do uživatelského rozhraní byla dohodnuta během přípravných fází a pouze se nedostala do dokumentu analýzy, ale již byla zohledněna ve fázi výroby šablon. V tomto případě vyžaduje přidání informace nebo funkčnosti další dodatečné úsilí, které zbytečně zvyšuje nároky na alokaci programátora.

Špatná orientace v HTML Stejně jako se neočekává od grafika, či kodéra, znalost konkrétního programovacího jazyka, neměla by být od programátora očekávána dokonalá znalost způsobů tvorby HTML a CSS uživatelského rozhraní. Proto nelze očekávat, že programátor bude doplňovat nové části tohoto rozhraní. Programátor by měl značkovacímu jazyku HTML rozumět pouze na rovině jeho zákonitostí, aby byl schopen zachovávat validitu kódu, uměl používat základní elementy pro generované výstupy. Každý programátor potvrdí, že nejhorší zkušeností je zorientovat se v cizím kódu. I v případě tak jednoduchého jazyka jako je HTML je skutečně problematické identifikovat místa, kam a jakým způsobem má být doplněn konkrétní výstup. K eliminaci tohoto problému pomáhají komentáře kodéra přímo uvnitř HTML kódu a vnitrofiremní pravidla pro jednotný způsob přípravy šablon. Tato opatření by měla zjednodušit orientaci programátora v dodávaném kódu.

Čistý kód z hlediska programování Hlavním cílem programátora, a také měřítkem kvality jeho práce, je správné a efektivní rozvržení kódu, použití metod zpracování, které jsou schopny zpracovat zadané vstupy a co nejrychleji vrátit výsledky zpět uživateli. Při práci jsou využívány tzv. *frameworky*, které nastavují celkový koncept práce a mají implementovány vlastní funkce zjednodušující např. práci s databází, standardními funkcemi pro práci s datem, textem nebo poli čísel. Výhod použití frameworků je nesporně mnoho, nicméně existuje také obdobné množství nevýhod. Obecně se v trendech programování přechází k objektově orientovanému programování (OOP) a určité abstrakce relačních databází, či přímé používání databází objektových. Protože jsou v projektu odděleny role programátora a kodéra, měl by i program být oddělen od vzhledové stránky. To znamená, že veškeré formátovací značky HTML by měly být až v prezentační vrstvě (šabloně). Poměrně často je možné se setkat s programy, které ve výstupech používají přímo HTML formátování, což způsobuje problémy nejen ve validitě celkového výstupního kódu, ale především výrazně omezuje možnosti změny vzhledu a údržby v budoucnu.

Konsolidace dílčích výstupů Pokud je možné funkčně rozdělit projekt na samostatné dílčí úkoly k programování, jedná se spíše o ideální případ. Většinou si jednotlivé moduly mezi sebou předávají data, která jsou základním předpokladem pro provedení určité funkce. Moduly jsou spouštěny dle pořadí elementů na stránce a následně předávány do uživatelského rozhraní. Velice zjednodušeně lze tedy říci, že postup zpracování a generování dat je závislý na rozvržení uživatelského rozhraní (strukturou stránky). Musí tedy existovat řídící element, který pořadí spouštění nebo generování výstupů koordinuje. V případě změny v prezentační vrstvě je tedy nutné upravit nejen vzhled uživatelského rozhraní, ale i tento koordinační prvek systému, případně i vlastní moduly.

Finalizace činností Jelikož je programátor poslední rolí ve vývojové fázi, připadají na něho také veškeré požadavky na dokončení, odstranění nedodělků a kompletace výsledného produktu. Před odevzdáním k testování musí identifikovat systémové nedostatky a chyby a následně musí být iniciováno jejich dokončení nebo oprava zástupci příslušných rolí. Jelikož se v tuto chvíli již jedná o hotovou webovou aplikaci, změny do ní musí zanést opět programátor na základě dodaných oprav. Pokud by např. kodér mohl zjištěné nedostatky odstranit sám, bez účasti programátora, zrychlil by se celý proces dokončení a finální zodpovědnost by zůstala na každé zapojené roli.

2.1.3 Tester

Testování se v případě webové aplikace skládá z testování chování uživatelského rozhraní, integrovaného fungování aplikace a validace čtení a zápisu dat (např. do databáze). Tester by měl mít možnost si vstupní data připravit (pokud je nemůže přímo zadat do aplikace) a mít možnost sledovat výstupy. První podmínka nemusí být zásadním problémem, neboť valná většina dat přichází od uživatele standardním HTTP požadavkem, který obsahuje vstupní hodnoty. Je tedy možné nasimulovat vstupy tímto způsobem. Nicméně někdy je potřeba mít připravena data v určitém kontextu (session) – tj. data vzniklá předchozím použitím aplikace. Simulace tohoto stavu vyžaduje postupovat cestou čistě uživatelskou – tedy

prostřednictvím uživatelského rozhraní testované aplikace. Testem se tak ověřuje funkčnost dvou částí najednou – uživatelského rozhraní a aplikace na pozadí.

Pokud není kontrola prováděna vůči záznamům v databázi je ověření správnosti výstupů v prostředí uživatelského rozhraní složitější, neboť data jsou aplikací různě transformována za účelem jejich poskytnutí do uživatelského rozhraní, stejně tak mohou být znovu upravována přímo při generování obrazovek rozhraní. Je tedy obtížné identifikovat původ chyby, není-li možné ověřovat fungování aplikačního jádra a uživatelského rozhraní zvlášť.

2.1.4 Projektový manažer

Z hlediska efektivního řízení projektů webových aplikací jsou základní problémy projektového řízení identifikovány především v oblasti využívání a alokace zdrojů. Na základě stanoveného projektového plánu a milníků jsou alokováni zástupci jednotlivých rolí na určitou část projektu ve vymezené nominaci, dané počtem pracovních hodin (WH - *working houres*) nebo pracovních dnů (MD - *man days*). V případě, že se provádění některé činnosti zdrží, znamená to posun začátků ostatních činností a tedy i posun alokace daného zdroje. Zmíněné posuny je ale třeba chápat v širším kontextu, kdy za alokaci zdrojů je zodpovědný jejich manažer a zdroje takto alokuje na více projektů a proto při posunu aktivity nemusí být zdroje již k dispozici vůbec, nebo mohou být k dispozici v podobě jiné osoby. To představuje dodatečné projektové náklady na zaškolení osoby a uvedení do problému.

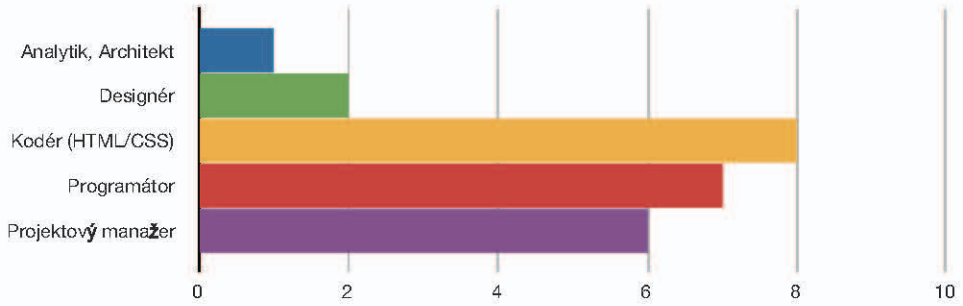
Obecně lze tedy zásadní problém z hlediska rolí spatřovat v problémech s alokací a výkonem týmů plnit dané úkoly v zadaném čase a kvalitě. Projektový manažer musí taktéž efektivně vést tým a činnosti vhodně navazovat, aby se tímto eliminovala neurčitost zadání pro navazující činnosti.

2.1.5 Průzkum

Výše uvedená úskalí tvorby webového projektu byla sestavena na základě diskuse a spolupráce s internetovými agenturami a osobní zkušenosti autora. Proto jsou klíčové problémy definovány poměrně nezaujatým pohledem. Pro ověření uváděných tvrzení a také kvůli možnosti nalezení případných dalších úskalí byl vytvořen *jednoduchý webový dotazník*, do kterého byla shrnuta všechna zjištěná úskalí a tento formulář byl prostřednictvím Internetu zpřístupněn k odpovědím.

Prostřednictvím e-mailu byli osloveni jak členové projektových týmů ze spolupracujících agentur, tak respondenti ze stejného sektoru, či najímají specialisté (freelancers). Respondenti jsou z oblasti internetových studií, marketingových agentur a vývojářských firem. Struktura respondentů dle rolí vykonávaných v projektech je naznačena v grafu 2.1. Nejvíce byli zastoupeni kodéři, programátoři a projektoví manažeři. Z celkového počtu 39 oslovených respondentů formulář vyplnilo a odeslalo celkem 24 z nich.

Byli požádání o zodpovězení šesti otázek ohledně zjištěných klíčových problémů jednotlivých rolí. Dotazník byl vytvořen prostřednictvím webové služby *Wufoo* a jeho distribuce a následný sběr dat byla realizována prostřednictvím Internetu. Adresa, na které je dotazník přístupný je `http://kremik.wufoo.com/forms/dotaznak-disertaaena-prace/` [cit. 2008-09-20] a dotazník je zároveň vložen jako příloha A této disertační práce.



Obrázek 2.1: Struktura počtu respondentů na dotazník dle rolí

V první části dotazníku byly otázky s předdefinovanými odpověďmi typu výběr z variant a bylo požadováno zaškrtnutí maximálně 3 možností z průměrně 9 nabízených variant. Nabízené odpovědi byly vytvořeny na základě rozboru uvedeného v úvodu tohoto oddílu. Ve druhé části bylo ponechána odpověď plně na respondentovi, aby se zamyslel, jestli z jeho pohledu způsobují ještě jiné faktory problémy efektivnímu vývoji, které nebyly v typizovaných odpovědích zmíněny. Do jednotlivých variant odpovědí byly přidány také tzv. „únikové body", které nepocházely z originálně identifikovaných problémů, aby tak bylo možné lépe ověřit, jestli jsou předpokládané problémy chápány stejně také respondenty. Předpokládá se, že tímto způsobem a omezením na maximálně 3 možné odpovědi se více oddělí skutečně chápané překážky od opakovaných, známých a nesouvisejících problémů (např. málo času, špatné zadání, ...).

Dotaz na každou oblast je formulován obdobně: „Při (návrhu designu, tvorbě layoutu, programování, projektovém řízení) se nejčastěji potýkám s následujícími problémy:". Odpovědi jsou pro každou oblast různé a jsou uvedeny v příloze A.

2.1.5.1 Zpracování výsledků

Průzkumu se zúčastnilo 61,5 % oslovených respondentů. Vzhledem k tomuto množství respondentů a množství otázek v průzkumu nebylo použito žádné statistické metody k očištění dat nebo jejich podrobnější analýze.

Získaná data byla sbírána systémem *Wufoo* a následně byla zpracována níže uvedeným způsobem. Každá sekce dotazníku odpovídá dotazům na konkrétní roli. Respondenti museli primárně odpovědět na dotazy ohledně své role, ale zároveň byli požádáni o odpovědi na otázky týkající se ostatních rolí. Kromě pohledu své vlastní role, nebyla odpověď na ostatní oblasti povinná.

1. U každé otázky byly vždy sečteny počty odpovědí (četnost) všech respondentů pracujících primárně v této roli.

2. Odpovědi ostatních rolí na danou otázku byly sečteny zvlášť. Odpovědi každého respondenta tedy byly vždy u jedné otázky sčítány pod hodnoty hlavní role a u ostatních do hodnoty ostatních rolí.

3. U každé otázky tak vznikly dvě sady odpovědí, které byly přepočteny na procentní zastoupení vzhledem k počtu odpovídajících respondentů, aby bylo možné obě skupiny oddělit. Přepočtem na celkový počet odpovídajících bylo možné zohlednit nepovinné odpovědi.

4. Procentní hodnoty, vztahující se k vlastní roli respondenta, jsou v grafu zobrazeny červenou linkou, odpovědi ostatních kumulovaně šedou.

5. Výsledky jsou prezentovány na sadě grafů na obrázku 2.2. Na grafu je možné sledovat nejen identifikované závažnosti daného problému z pohledu jednotlivých rolí, ale také rozdíly v chápání problému vykonavateli dané role oproti zbytku projektového týmu.

2.1.5.2 Výsledky průzkumu

Z hlediska **designérů** se obě skupiny víceméně shodly a jako nejproblematičtější byly hodnoceny následující 4 oblasti, přičemž poslední dvě byly hodnoceny naprosto stejně:

1. Málo času na tvorbu designu
2. Jedná se o první fázi projektu a požadavky se často mění
3. Musí být vytvořeno více zbytečných návrhů
4. Nevyhovující podklady od zákazníka

Zajímavou shodou mezi oběma skupinami byla v odpovědi, že na design nemá žádný negativní vliv grafický cit zákazníka a z toho vyplývající požadavky. U designu byla zvolen právě krátký čas jako nejkritičtější zřejmě kvůli závislosti rolí na začátku projektu, protože do doby, než je dokončen design, čekají všechny ostatní role a proto je vyvíjen tlak na rychlé dokončení, přičemž do tohoto procesu vstupuje schválení zákazníkem, který obvykle zadání ještě několikrát změní, nebo je nutné kvůli němu vytvářet různé další návrhy, které se pak nepoužijí a ve své podstatě prodlužují nejen samotnou fázi designu, ale i vývoje aplikace jako celku. Nicméně otázky krátkého času i grafického cítění zákazníka představovaly tzv. únikové otázky, popisované v oddílu 2.1.5. Tímto se jejich vliv vzájemně eliminoval.

Průzkum zcela nepotvrdil následující faktory, u kterých se obě skupiny shodly na téměř nulovém vlivu: není dokončena analýza, není se o co opřít; zákazník nemá grafické cítění; jsem omezen možnostmi webu a nemám pro svoji práci vhodné nástroje.

U otázky **kódování** jsou výsledky zajímavé v tom ohledu, že se obě skupiny shodly na většině otázek a proto se křivky na grafu téměř kopírují. Jako 3 nejzávažnější problémy byly identifikovány:

1. Málo času na přípravu šablon
2. Zbytečná tvorba prototypů (klikací šablony)
3. V grafice je něco jiného než v analýze (co je požadováno zákazníkem)

Jako nejméně ovlivňující byly označeny možnosti „Pro svou práci nemám vhodné nástroje" a „Moje účast je vyžadována po celou dobu projektu", na kterých se obě skupiny víceméně shodly. I u kodéra je patrné, že na jeho výstupy čeká další fáze, která nemůže začít bez připravených šablon. Překvapující je, že na druhém místě skončily prototypy, které jsou v současné době většinou substituovány grafickými návrhy a právě kvůli složitému vytváření a nepoužitelnosti prototypu v budoucích fázích projektu většina firem od prototypů na úrovni HTML upouští, i když je jejich přidaná hodnota nesporná. Třetí identifikovaná překážka reflektuje první validaci analýzy a návrhů vůči realitě, protože se nejen vytváří vizuální stránka aplikace, ale zároveň se začínají vyskytovat první logické nedostatky grafického návrhu nebo informační architektury.

Důležitá je také skutečnost, že žádné kritérium nedosáhlo extrémně nízkých, nebo nulových, hodnot, což znamená, že definované předpoklady jsou v případě kódování víceméně správné.

V případě **programování** byly opět ve shodě obou skupin označeny následující 3 nejvíce negativně působící faktory:

1. Nedostatečné, neodpovídající zadání
2. Nutná častá interakce s kodérem za účelem realizace požadovaného vzhledu, funkčnosti
3. Pozdní zapojení do projektu, kdy už nemohu vznést své připomínky

Obě skupiny se shodly, že jako zcela neovlivňující faktor je „Používané pracovní postupy, framework". Ani jeden z programátorů pak neoznačil odpověď „Obtížné udržet kód čistý od specifik layoutu (HTML)". Kromě samotných programátorů si ostatní role nemyslí, že bylo pro programátory obtížné orientovat se v HTML kódu a integrovat ho s výstupy aplikace. Zde je vidět zcela opačný názor, který může být také úskalím při tvorbě projektů a vzájemném porozumění si.

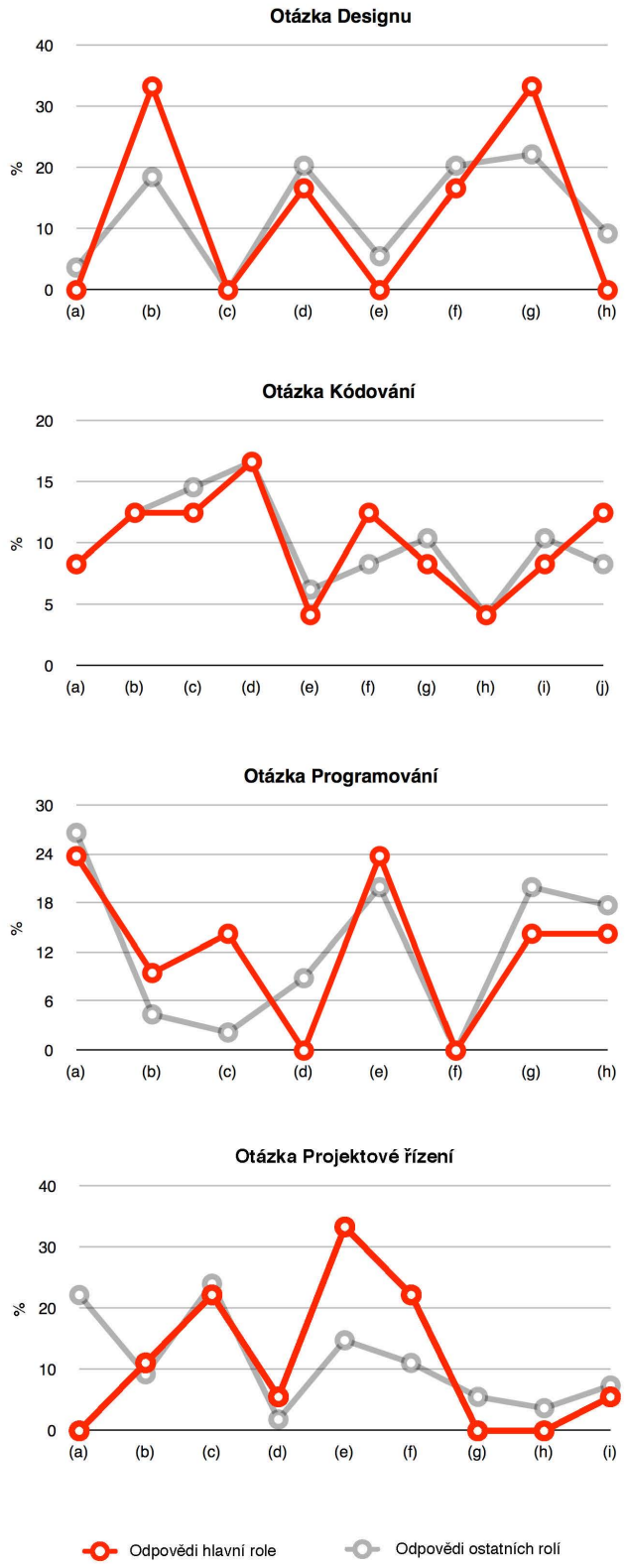
Nejčastější odpověď ohledně nedostatečného zadání představuje únikový bod, protože, dle zkušenosti, zadání nikdy nebude obsahovat popis všech funkcí do posledního detailu a tak bude neodpovídající rozsah zadání pravděpodobně zmiňován i nadále.

Problémy v interakci s kodérem jsou dány především obecnou závislostí těchto rolí – tj. že programátor jen s obtížemi, případně s nepřiměřeně vynaloženým úsilím, může upravit část uživatelského rozhraní, aniž by potřeboval pomoc kodéra, což platí i naopak. Prolínají se tu také výstupy práce kodéra a programátora, které jsou často pomíchány dohromady.

Protože programátor je až poslední rolí, která se díky návaznosti činností, zapojuje do projektu, vzniká tímto spousta dotazů a nepřesností, které vyvolávají opětovnou potřebu zanalyzování určité části.

Z hlediska **projektového řízení** se již odpovědi obou skupin více rozcházely hlavně co do pořadí a proto uvedeme jak byla úskalí efektivního projektového řízení ohodnocena jednotlivými skupinami samostatně:

Projektoví manažeři:



Obrázek 2.2: Odpovědi na jednotlivé otázky

1. Prodloužení jedné fáze automaticky prodlužuje celou dodávku
2. Obtížný souběžný vývoj více rolí, nebo více součástí projektu najednou
3. Neustálé změny zadání prováděné zákazníkem

Ostatní:

1. Neustálé změny zadání prováděné zákazníkem
2. Není dokončena analýza a není se proto o co opřít
3. Prodloužení jedné fáze automaticky prodlužuje celou dodávku

V projektovém řízení se většinou nejvíce řeší projektové plány, milníky a délky trvání aktivit. Právě posuny nebo prodloužení délky trvání jednotlivých aktivit ovlivňují časový plán projektu a ten pak musí projektový manažer diskutovat se zadavatelem. Řešením prodloužení jedné aktivity by bylo řešitelné předstihem, tedy vzájemným překrýváním aktivit, nicméně toto není ve většině případů vůbec možné. V průběhu projektu je nutné efektivně řídit změny ze strany zadavatele, což není, dle výše uvedených výsledků, pro projektové manažery jednoduché a obdobě to vnímají i ostatní respondenti.

Jako nejvýraznější je možné dle průzkumu tedy považovat prodlužování fází, neustálé změny zadání a obtížný souběžný vývoj více rolí. Tyto problémy představují spojené misky vah, protože změny zadání obvykle prodlužují délku trvání konkrétní fáze, nebo vyžadují neplánovanou alokaci některé role. Naopak paralelní alokace rolí by mohla tyto problémy řešit i bez nutnosti prodlužovat délku trvání fáze.

V oblasti projektového řízení byly označeny jako problémy s malým vlivem: obtížné nalezení role odpovědné za opravy zjištěných chyb, obtížná dohoda s členy projektového týmu, obtížná realizace formou iterací.

V oblasti volných odpovědí, kde byli respondenti požádáni o shrnutí překážek, které nebyly součástí sady typizovaných odpovědí, se jako nejvíce relevantní vyskytly následující příspěvky:

1. Podcenění projektu (po stránce finanční i zdrojové)
2. Minimální časové rezervy
3. Nekompetentní tým
4. Špatné řízení portfolia projektů a s tím související určování priorit
5. Časté změny nekonzistentní přístup v používaných technologiích

Celkově lze průzkum vyhodnotit pozitivně, neboť až na několik výjimek potvrdil, že členové projektového týmu vnímají překážky efektivního vývoje obdobným způsobem. To lze mimo odpovědí na typizované otázky pozorovat z obsahů volné části, kde mohli respondenti doplnit své postřehy a v 75 % případů se tak nestalo a pole zůstalo prázdné. Po diskusi s některými z respondentů se ukázalo, že poukazované problémy jsou způsobeny mnohdy také špatným

řízením projektů, což je patrné i z odpovědí v netypizovaných otázkách. U většiny případů je možné pozorovat, že se křivky víceméně kopírují, což znamená, že identifikované problémy pociťuje obdobně celý projektový tým.

2.2 Požadavky na navrhovanou metodiku

Jak již bylo uvedeno v úvodu tohoto oddílu, cílem této práce je navrhnout metodiku pro efektivní tvorbu webových aplikací. Na základě identifikace problémů jednotlivých projektových rolí, shrnutých v oddílu 2.1, a jejich ověření formou průzkumu, prezentovaného v oddílu 2.1.5, byl sestaven níže uvedený seznam požadavků, které zajistí zefektivnění popsaného vývojového procesu. Na základě těchto požadavků je navrhována metodika ETWA.

1. Zefektivnit celý vývojový proces webové aplikace
   - ▷ Zrychlit doručení produktu
   - ▷ Umožnit flexibilní alokaci zdrojů pracujících na projektu
2. Eliminovat závislosti rolí na sobě navzájem, umožnit paralelní alokaci
3. Striktně rozdělit a definovat odpovědnosti rolí za své výstupy
4. Vytvořit podmínky pro flexibilní stanovování priorit činností v rámci celého portfolia projektů i projektů samotných
5. Zefektivnit proces údržby a rozvoje aplikací
6. Metodika musí vycházet ze standardizovaných, podporovaných a dlouhodobě rozvíjených formátů, aby byl zajištěn i její vlastní rozvoj
7. Striktně oddělit prezentační a aplikační vrstvu
   - ▷ Oddělit posloupnosti zpracování v aplikačním kódu a uživatelském rozhraní
   - ▷ Odstranit prolínání použitých programovacích jazyků mezi jednotlivými vrstvami
   - ▷ Navrhnout metodu, kterou bude eliminováno vzájemné nesystémové prolínání vrstev
8. Připravit podmínky pro outsourcing jednotlivých projektových rolí
9. Podpořit specializaci týmových rolí
   - ▷ Oddělit pracovní náplň jednotlivých rolí
   - ▷ Rozvíjet schopnosti v definovaném oboru jednotlivých rolí
10. Aplikací metodiky podpořit vlastnosti vyvíjeného produktu (např. podporou různých výstupních formátů, víceJazyčných aplikací, rozložením zátěže a zpracování)

2.3 Předpoklady

Jestliže budeme vycházet z jednotlivých požadavků uvedených v oddílu 2.2 a komplikací vývoje, identifikovaných v oddílu 2.1, je primárním problémem vysoká závislost mezi prezentační a aplikační vrstvou. Tyto dvě části jsou víceméně jasně odděleny pouze v úvodních fázích projektu, ale postupně se během vývoje tyto dvě vrstvy čím dál více začínají prolínat, přičemž na konci vzniká jedna svázaná aplikačně-prezentační vrstva. Prezentační vrstva, při pohledu na aplikaci jako celek, slouží k účelům *prezentace, sběru vstupních dat a ovládání uživatelského rozhraní*. Aplikační vrstva naopak na základě definice ve zdrojovém kódu *sbírá, transformuje* a *generuje datové výstupy*. Zjednodušeně lze říci, že je tedy třeba oddělit zpracování dat od jejich prezentace. Aby bylo možné toto uskutečnit, je třeba identifikovat, jaká data kde ve webové aplikaci vznikají a také jakým způsobem. Za tímto účelem byly analyzovány webové aplikace různého typu, vytvářené různými internetovými agenturami, včetně používaných postupů jejich vývoje.

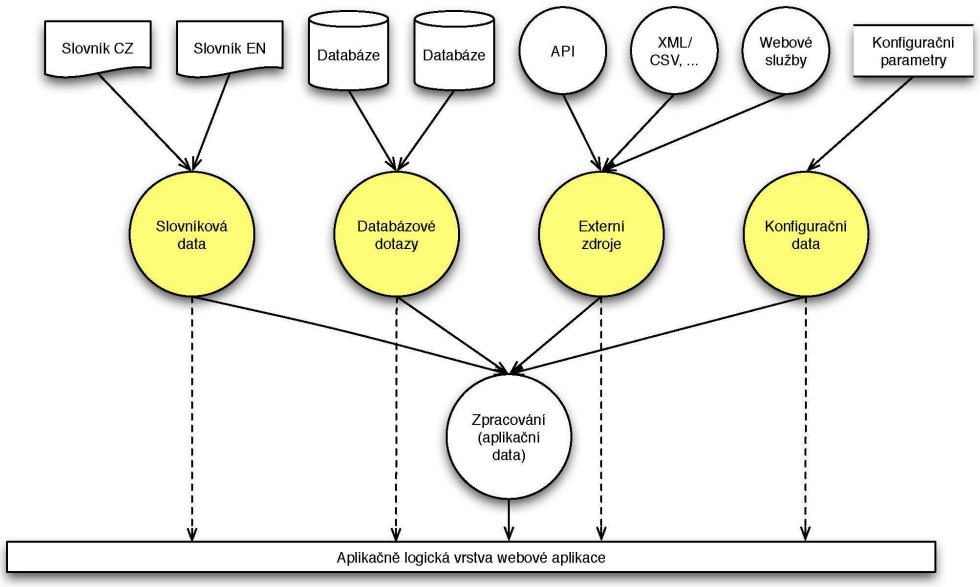
2.3.1 Data ve webové aplikaci

Daty v tomto ohledu rozumíme jak informace získané od uživatele, tak z databází, externích systémů, či data generovaná aplikací na základě jiných vstupů. Rozborem existujících aplikací a způsobů jejich fungování byly rozděleny na následující dílčí součásti:

- Části designu uživatelského rozhraní
- Statické texty (jakákoliv staticky vkládaná část HTML kódu)
- Jazykově variabilní části
- Data číselníkového charakteru (seznamy, roletky, navigační prvky, ...)
- Znovupoužitelné a opakující se elementy
- Výstupy databázových dotazů
- Oblasti prezentace výsledků provedených operací
- Formuláře pro zadávání vstupů

Vznik dat Aby bylo možné oddělit původ dat od místa jejich prezentace, musí být identifikována místa vzniku. Pro objasnění problému byla rozdělena dle způsobu, jakým jsou předávána, tj. dle formátu, případně protokolu. Data pro potřeby webové aplikace jsou:

- Vkládána formou konfiguračních souborů a parametrů
- Generována ze slovníkových souborů pro účely tvorby jazykově rozdílných částí
- Získávána z databáze formou databázových dotazů
- Získávána z externích zdrojů
  - Webové služby (externí volání rozhraní služby prostřednictvím protokolu HTTP, přičemž volání i data v odpovědi jsou zasílána prostřednictvím formátu XML)



Obrázek 2.3: Vznik dat pro účely prezentace ve webové aplikaci

  - Agregovaná data (data stahovaná převážně ve formě souborů ve formátech jako je XML, CSV, apod. Využívá se např. pro generování RSS kanálů pro syndikaci obsahu, nebo jako asynchronní API mezi heterogenními aplikacemi)
  - Konkrétní API služby (definováno na úrovni programovacího jazyka, včetně způsobů jeho implementace a volání)

- ▷ Vytvářena aplikací na základě výše uvedených dat (agregovaná data, výpočty, transformace, apod.)

Data jsou tedy pro účely prezentace, resp. zobrazení uživateli, vždy zpracována určitým procesem aplikačního serveru (v tomto případě programem na pozadí), jak je patrné z obrázku 2.3. Připojení na externí a interní zdroje dat konfiguruje a zprostředkovává vlastní aplikace a transformuje získaná data do výstupních datových struktur, aby mohla být následně jednoduchou cestou zaslána do prezentační vrstvy.

2.3.2 Používané postupy

Při tvorbě webových aplikací jsou využívány různé postupy, které reflektují způsob, jakým tým pracuje na webovém vývoji. Zohledněnými aspekty jsou také koncoví uživatelé, zdrojové i cílové systémy a možnosti společnosti. Způsob, jakým bude webová aplikace implementována je také částečně závislý na jazyku, platformě nebo frameworku. Jednotlivé přístupy je možné shrnout do následujících skupin, přičemž jednotlivé skupiny se mohou ve svých definicích překrývat.

Funkčně orientovaný vývoj Již několik let považován za překonaný – alespoň v oblasti webového vývoje – nicméně se stále jedná o významnou skupinu aplikací, především menšího rozsahu, nebo vyvinuté před několika lety. Veškerý kód aplikace je vytvářen a spouštěn

sekvenčně, s tím, že ke všem složitějším, případně opakujícím se operacím je nadefinována funkce v knihovně funkcí a jejím prostřednictvím je proveden např. výpočet nebo vygenerování části uživatelského rozhraní. Existují tedy např. funkce pro převod pole do tabulky, vygenerování nadpisu, odkazu nebo jiného elementu. Uživatelské rozhraní aplikace je generováno buď kompletně programovým kódem, nebo prostřednictvím vkládání kódu do HTML zdroje.

Objektově orientované programování (OOP) V současné době je velmi populární. Aplikace je složena z jednotlivých objektů, které jsou vytvářeny jako instance definované třídy. Každý objekt má své vlastnosti (atributy), stejně tak metody, prostřednictvím kterých se přistupuje k jeho atributům. Dále, z podstaty OOP, existuje v objektově orientovaném programu polymorfismus a dědičnost, které zjednodušují práci s objekty a třídami. Z hlediska realizace aplikační a prezentační vrstvy je i v objektovém programování možné stránky generovat kompletně prostřednictvím kódu aplikace, stejně tak vkládat objekty přímo do HTML kódu. Vlastní šablona může být realizována jako kolekce objektů, reprezentující jednotlivé typy šablon. Těmto objektům je pak prostřednictvím sady objektů zaslán obsah konkrétní stránky, kterou mají vygenerovat do prohlížeče. Ovšem ani toto není pravidlem a konkrétní použití závisí hlavně na programátorovi.

Model-View-Controller (MVC) MVC dělí interaktivní aplikaci na tři oblasti: zpracování (výpočet), vstupy, výstupy. Komponenta model zapouzdřuje základní data a funkcionalitu. Model je nezávislý na specifikacích vstupního a výstupního chování aplikace. Komponenta View (pohled) zobrazuje uživateli informace. View dostává data od modelu. Na model může existovat několik pohledů. Každá komponenta View je asociovaná s komponentou controller. Controllers dostávají vstup obyčejně jako události, které jsou způsobeny pohybem myši, stisknutím tlačítka myši, nebo stisknutím klávesy na klávesnici. Tyto události jsou transformovány na požadavky služeb pro model, nebo view. Uživatel má interakci se systémem výhradně prostřednictvím komponent controller. Separace komponenty model od komponent view a controller umožňuje násobné pohledy na stejný model. Pokud uživatel změní model prostřednictvím komponenty controller jedné komponenty view, všechny ostatní komponenty view závislé na těchto datech se také změní. Komponenta model proto uvědomuje všechny komponenty view, zda-li se jejich data změnila. Komponenty view na druhou stranu obnovují data z komponenty model a aktualizují zobrazované informace. Mechanismus propagace změn je také popsán ve vzoru Publisher-Subscriber [21]. MVC byl poprvé definován v jazyce *Smalltalk*. V současné době je pravděpodobně nejvýznamnějším a nejoblíbenějším frameworkem pracujícím na bázi MVC *Ruby on Rails*. Šablony jsou v tomto případě odděleny, nicméně ve většině frameworků se jedná spíše o oddělení formální, kdy je možné realizovat funkčnosti jedné komponenty uvnitř komponenty jiné a tomu účelu neodpovídající. I přes podobné nestandardní zásahy, nemá takováto chyba na celkové fungování téměř žádný vliv.

Generované rozhraní Bývá k vidění u velkých systémů, které původně sloužily pouze jako klasické aplikace. Nyní s rozšířením dostupnosti webového přístupu vzniká potřeba

i u takto velkých, a poměrně složitých, systémů mít také webové rozhraní. Existují proto roboti, kteří se specializují na překlad klasického uživatelského rozhraní na prostředí webové. V tomto případě se tedy jedná o naprostou realizaci na základě oddělení šablony a aplikačního jádra. Existují produkty (např. Novell), které umožňují pomocí XML dokumentu popsat určité rozhraní klasické aplikace, transformovat tyto informace do webové stránky a následně opět stejným procesem vrátit data z webového formuláře zpět do příslušných polí klasické aplikace. Díky poměrně složitému způsobu integrace a nároky desktopových aplikací se jedná - z hlediska kvality webového interface - o nestandardní aplikace s robustním a nekonzistentním HTML kódem. Do této skupiny spadá i např. SAP, který se zpočátku vydal popisovaným směrem.

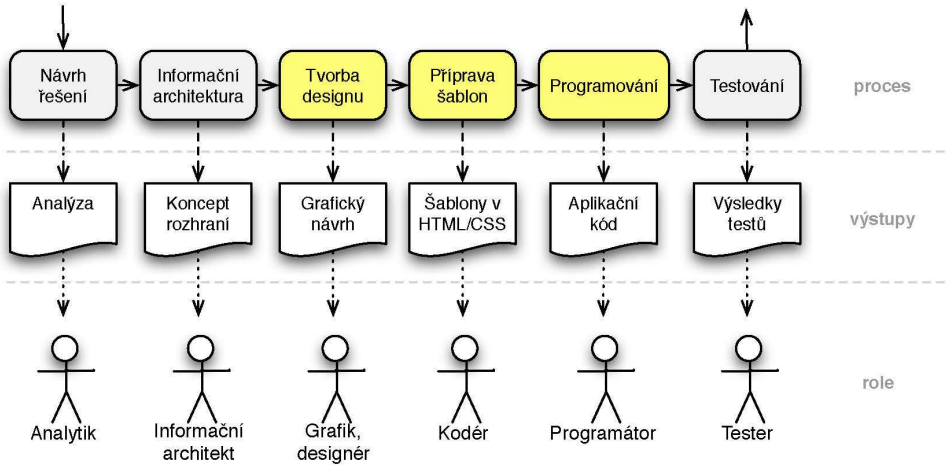
Webové služby Představují systém pro podporu vzájemné spolupráce počítačů v síti. V podstatě jde o komunikaci mezi 2 počítači, při níž má jeden funkci poskytovatele webové služby a druhý je klientem. Klient (requestor) si zjistí adresu služby (vyhledá v registru nebo má adresu přímo od poskytovatele), stáhne si popis služby a je ji následně schopen využívat. Poskytovatel (provider) služby poskytuje data specifikovaným způsobem prostřednictvím sítě. Data se přenášejí v XML formátu a díky němu jsou snadno rozšiřitelná. Jedná se o velice praktický případ využití formátu XML pro běžnou praxi a téměř libovolný typ záznamů. Aplikace poskytující webové služby je mohou realizovat bez uživatelského rozhraní, protože k jejich funkčnosti jej není třeba. V určitém případě mohou být webové služby využívány jako externí zdroj dat pro uživatelské rozhraní.

2.4 Definice metodiky

Na základě požadavků definovaných v oddílu 2.2 a analýzy webových aplikací (oddíl 2.3) je navržena metodika ETWA *(**E**fektivní **t**vorba **w**ebových **a**plikací)*, která je definována především modifikacemi na úrovni technologie zpracování. Aby se v následujícím textu usnadnila orientace a text nebyl zbytečně přetěžován, je dále používáno pouze označení ETWA, které vždy označuje navrhovanou metodiku.

Dotazovaní respondenti používají různé postupy vývoje webových aplikací a přesto identifikují obdobné problémy a překážky, bránící zefektivnění procesu. Je proto třeba začít revizí samotného způsobu vývoje a identifikovat tak důvody, kvůli kterým není možné některé procesy provádět efektivněji. Na základě tohoto poznání je navrhována jejich úprava. Vzhledem k tomu, že prostředí vývoje jsou značně heterogenní, co do používaných jazyků, technologií, platforem i samotného řízení projektu, je cílem navrhnout metodiku dostatečně univerzálně, aby nebyla vztažena ke konkrétní implementaci, jazyku, či platformě. Výsledkem opačného postupu by byl návrh *frameworku*, kterých je v současné době k dispozici nepřeberné množství.

Návrh metodiky vychází z univerzálních, ověřených a nezávislých formátů a doporučení, aby mohla být aplikována bez ohledu na konkrétní podmínky použití. Cílem, na který se metodika soustřeďuje nejvíce, je zefektivnění tvorby webových aplikací tak, aby bylo možné zkrátit časy dodávky, zefektivnit využívání a alokaci zdrojů, umožnit rozvoj aplikace do budoucna.



Obrázek 2.4: Konvenční vývojový proces webové aplikace

Na schématu 2.4 je naznačeno jak se v konvenčním vývojovém procesu zapojují jednotlivé role v čase a jaké svojí činností generují výstupy. Zároveň je zde naznačena závislost jednotlivých činností na sobě. Toto schéma je bráno jako výchozí stav, který si metodika ETWA na základě požadavků z oddílu 2.2 klade za cíl změnit.

V dalších oddílech je popsáno, jaký vliv na jednotlivé role, a jimi realizované fáze, metodika má a jaké překážky a omezení jsou tímto eliminovány. Metodika je popisována konkrétně, včetně vzorových příkladů její aplikace.

2.4.1 Pohled programátora

Cílem metodiky je roli programátora posunout čistě do roviny programového kódu definovaného v rámci zvoleného programovacího jazyka. Na výstupech programu požaduje mít čistá data, která se pro účely prezentace zformátují a upraví do výsledného formátu až v okamžiku transformace do cílové šablony. Tímto způsobem bude aplikace posunuta blíže k nezávislosti na konkrétním výstupním formátu, tedy i nezávislosti na konkrétním koncovém zařízení. Aplikace bude pracovat na základě volání příslušných procedur a funkcí a zajistí tak spojení se všemi zdroji dat prezentovanými na schématu 2.3.

Programátor musí veškerá data připravit v takovém formátu a kvalitě, aby na základě nich bylo možné sestavit všechny součásti uživatelského rozhraní, neboť na jeho straně nebude možné provádět jakékoliv aritmeticko logické operace. Na druhou stranu nebude od programátora vyžadována žádná přímá interakce s uživatelským rozhraním a to zejména z pohledu jeho tvorby. Formátování, validace i ošetření dat vůči pravidlům HTML/CSS, apod. není primárně vyžadováno.

Z hlediska zapojení do projektu, je programátor nominován také do přípravné fáze, tj. do období dokončování analýzy a před začátkem tvorby uživatelského rozhraní. Programátora je možné alokovat pro vlastní vývoj nezávisle na stupni rozpracovanosti uživatelského rozhraní. Ověření funkčnosti dodaného kódu je prováděno již během vývoje a ve výsledku pak na úrovni transportní vrstvy (viz dále).

2.4.2 Pohled kodéra

Kodérovi je v rámci metodiky dána kompletní zodpovědnost za uživatelské rozhraní a to včetně jeho naplnění aplikačními daty. Kodér je zcela odstíněn od aplikačního vývoje a pracuje pouze s konkrétně stanovenou sadou dat, která je předem dána a v rámci tvorby aplikace je striktně vyžadována.

Znalosti kodéra jsou tímto přístupem rozvíjeny především v oblasti značkovacích jazyků a v rámci aplikace metodiky je vyžadováno zaškolení do dalších značkovacích jazyků, zejména v doporučovaných jazycích XML a XSL. Bez tohoto zaškolení nemůže být kodér schopen dle metodiky pracovat. Na druhou stranu od kodéra není vyžadována znalost jakéhokoliv jiného šablonovacího nástroje nebo pseudojazyka pro realizaci šablon. Do výstupů kodéra nemusí jakkoliv zasahovat jiná role, v tomto případě především programátor.

Kodér je do projektu, obdobně jako programátor, zahrnut před dokončením analýzy a začátkem jakékoliv vývojářské nebo designérské práce. To mu umožňuje uplatnit nejen své připomínky, ale především si všechny strany musí v tento okamžik (na základě analýzy a vytvořených wireframů) dohodnout, jaká data budou pro účely vývoje aplikace potřebovat. Alokace kodéra na projektu je zcela nezávislá na programátorovi a i bez funkční aplikace musí být schopen zprovoznit a simulovat uživatelské rozhraní budoucí aplikace.

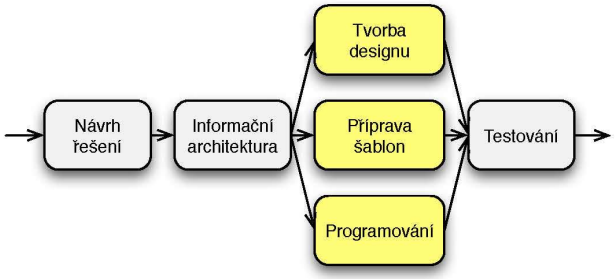
2.4.3 Pohled testera

Při vývoji webové aplikace jsou velice často podceňovány testy, proto je cílem metodiky vytvořit pro testování lepší podmínky a zefektivnit jejich provádění. Testeři musí mít možnost aplikaci testovat na úrovni uživatelského rozhraní i aplikačního kódu a to již v průběhu jejího vývoje. Výsledky testů tak jednoznačně určují, kdo je za danou oblast zodpovědný a kdo zjištěnou chybu opraví. Ideálním stavem je automatizace určité množiny testů, díky níž je možné testovat nejen správnost a integraci, ale také výkon a případné problémy s extrémní zátěží. Testování (ať už ruční nebo automatické) probíhá podle testovacích scénářů.

2.4.4 Pohled projektového manažera

Z hlediska řízení projektu nejsou na projektového manažera kladeny žádné nové požadavky. Naopak metodika umožňuje lépe alokovat zdroje, přesouvat je mezi různými projekty a případně určovat priority jejich nasazení, bez toho aniž by se nutně opozdil vývoj jiného projektu. Toto je umožněno oddělením závislostí aktivit a tedy i rolí v projektu navzájem. Návrh metodika si klade za cíl eliminaci množství chyb již průběhu samotného vývoje a také okamžitou opravu. Ve vztahu k zadavateli je tak možné prezentovat mezivýstupy mnohem dříve než po dokončení aplikační části a odhalit tak případné nesrovnalosti již v průběhu vývoje a tím mít dostatek času na jejich opravu. Tento přístup zajišťuje minimalizaci dopadu změn na množství souvisejících prací, neboť změnu, či nesrovnalost zjištěnou v raných stádiích vývoje, je možné opravit za mnohem menších nákladů než v dokončené aplikaci.

Díky minimalizaci závislosti rolí a tím i rozdělení zodpovědností, je možné lépe adresovat požadavky, podle toho jestli se týkají uživatelského rozhraní či fungování aplikace. Z hlediska projektového plánu je možné plánovat aktivity programování a tvorby uživatelského rozhraní paralelně a tím dosáhnout zkrácení celkové doby dodávky.



Obrázek 2.5: Cílový stav vývojového procesu dle metodiky ETWA

2.4.5 Cílový stav

Zavedením výše uvedených změn do činností jednotlivých projektových rolí bude dosaženo změny vývojového procesu webové aplikace z původního postupu uvedeného na schématu 2.4 na postup prezentovaný na schématu 2.5. Jak je možné při porovnání vidět, žlutě označené činnosti je možné provádět paralelně a víceméně nezávisle na sobě. K tomuto účelu je nutné do vývojového procesu zanést nové prvky, které tuto změnu umožní realizovat. Popis těchto změn je předmětem následujících oddílů.

2.5 Prvky metodiky

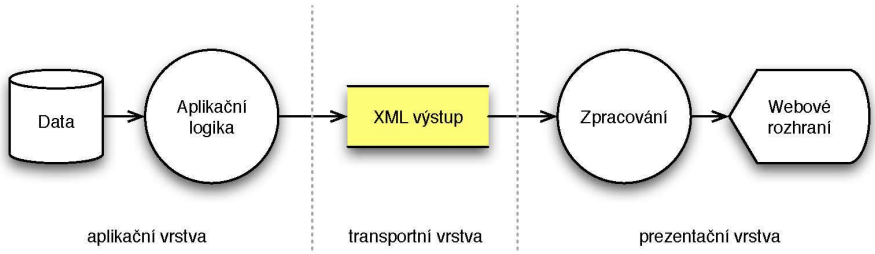
Základní myšlenkou celé metodiky ETWA je vložení **nové mezivrstvy** mezi vrstvu aplikační logiky a vrstvu prezentační. Nově vložená vrstva je označována jako *transportní* a obsahuje veškerá data potřebná pro vytvoření obsahu (nikoliv vzhledu) uživatelského rozhraní a zobrazení požadovaných výstupů. Obdobné přístupy jsou již používány v konceptu MVC, kdy vrstva modelu zajišťuje vytvoření datové struktury a její následné použití v jednotlivých šablonách, tedy views. V šablonovacích systémech je použit podobný přístup, kde se pro sestavení transportní vrstvy využívají různé typy proměnných nebo objektů programovacího jazyka. Vlastní šablona potom obsahuje speciální řetězce znaků, ohraničené např. složenými závorkami (příklad: `Titulek: {$title_text}`) a tyto řetězce jsou následně např. pomocí regulárních výrazů nahrazovány příslušnými hodnotami. Vzorový dokumenty takovéto šablony je uveden v příkladu 1.3.3.

Na příkladě 2.5.1, vytvořeného v MVC frameworku CakePHP, je demonstrován způsob jakým jsou data z controlleru (tj. aplikační logiky) předávána do šablony pro následné vygenerování do stránky. V tomto případě jsou v šabloně dostupné proměnné `$section`, `$mytitle` a `$events`, přičemž poněkud nestandardním způsobem bude do stránky předán také titulek okna jako `$pageTitle`. Datový formát každé proměnné je tímto nejasný, neboť není nikterak definován a proto ač jsou proměnné do šablony předány jednotným způsobem, musí být při generování uživatelského rozhraní bráno v úvahu jestli se jedná o proměnnou typu string, integer, pole nebo dokonce objekt. Proměnná `$mytitle` bude obsahovat obyčejný text a proměnná `$events` bude obsahovat asociativní pole.

Uvedený příklad demonstruje zásadní problém při transportu dat do šablony a omezuje tak možnosti diverzifikace rolí a činností na projektu: *názvy proměnných i datové typy*

Příklad 2.5.1 Plnění výstupních dat v aplikaci pro použití v šabloně (CakePHP).

```
$this->set('section', $this->Section->findBySlug($slug));
if ($slug == 'sport-akce') {
   $this->pageTitle = 'Sportovní akce';
   $this->set('mytitle', 'Sekce:<br />Sportovní akce');
   $this->set('events', $this->Event->typeEvents($slug));
} else {
   $this->pageTitle = 'Kulturní a společenské akce';
   $this->set('mytitle', 'Sekce:<br />Kulturní a společenske akce');
   $this->set('events', $this->Event->nonTypeEvents('sport-akce'));
}
```



Obrázek 2.6: Zavedení XML transportní vrstvy

volí pouze programátor. Neurčitost výstupů by mohla být eliminována možností zobrazení výstupů aplikace ještě před integrací se šablonou, což u většiny frameworků není možné a tak je obsah výstupů většinou prozkoumáván dumpem (přímým výpisem) obsahu objektu zajišťujícího transport dat, apod.. Takový výstup je ale značně velký, nestrukturovaný a srozumitelný většinou pouze programátorovi.

Definice metodiky ETWA je založena na předpokladu, že kodér rozumí tzv. markupu, neboli značkovacímu jazyku, tedy jazyku, který určité části textu (dat) ohraničuje značkami a tím dává těmto datům konkrétní význam. Značkovacím jazykem je HTML (Hypertext *Markup* Language), nebo XML (eXtensible *Markup* Language), které vychází ze stejné definice univerzálního značkovacího metajazyka SGML (Standard Generalized *Markup* Language), který umožňuje definovat značkovací jazyky jako své vlastní podmnožiny. SGML je komplexní jazyk poskytující mnoho značkovacích syntaxí. Protože kodér jako základ své práce používá HTML, resp. XHTML, není pro něho zásadní změnou používání jiného jazyka se stejnými pravidly. V návrhu metodiky je proto kladen důraz na využití SGML odvozených jazyků pro tvorbu prezentační vrstvy.

Pro metodiku ETWA je zvolen **standardizovaný formát jazyka XML** a vlastní napojení aplikační a prezentační vrstvy je realizováno jako další samostatný krok zpracování. To znamená, že zpracování může být ukončeno v okamžiku ukončení běhu všech úloh aplikačního kódu a výsledky tak mohou být validovány na úrovni vygenerovaných dat ve formátu XML. Na straně aplikace jsou veškeré výstupy transformovány do výstupního formátu XML. Zavedení transportní vrstvy je znázorněno na obrázku 2.6.

Ukázka možného postupu při generování XML výstupů je podrobněji popsána v oddílu

2.5.1.2. Metodika ETWA představuje definici na vyšší úrovni, bez omezení na konkrétní programovací jazyk a konkrétní způsob implementace v něm. XML mezivrstvy je tedy možné využít jak při klasickém strukturním/funkčním programování, tak v přístupech objektově orientovaného programování, apod..

Základním rozdílem mezi dříve popsanými způsoby tvorby webových aplikací a použitím *XML mezivrstvy* je jednoznačná identifikace předávaných dat, která aplikace produkuje a předává do XML transportní vrstvy, respektive šablony. Díky této identifikaci a struktuře je možné při tvorbě šablony vycházet ze známé struktury a vůči ní připravovat vlastní zobrazení. Uvedeným způsobem je možné zpracovávat XML data nezávisle na jejich pořadí, což je výhodou jak pro programátory aplikační části, tak pro zpracování šablony.

Zavedení XML mezivrstvy již samo o sobě naplňuje některé z požadavků, které byly definovány v oddílu 2.2. Konkrétně se jedná o následující přínosy, které je možné zavedením získat:

- Uvnitř aplikace není nutné řešit jakékoliv kolize vůči použitému značkovacímu jazyku, data jsou generována prostřednictvím konstrukcí programovacího jazyka (není tedy nutné ošetřovat výstupy na validnost HMTL kódu, ani převádět (escape) nestandardní znaky).
- Díky unikátní identifikaci každého údaje, obsaženého uvnitř vygenerovaného XML dokumentu, nezáleží na pořadí, ve kterém jsou jednotlivé prvky (data) do XML dokumentu vložena. Díky tomu je možné aplikační zpracování řídit skutečnými potřebami aplikace, nikoliv prezentační vrstvy.
- Vzhledem k výše uvedeným bodům je aplikační kód výrazně menší a přehlednější, neboť obsahuje pouze kód nutný pro zajištění aplikační logiky, který je i do budoucna mnohem lépe udržovatelný.
- Aplikační programátor se tak může soustředit na tvorbu logiky aplikace bez ohledu na způsob zobrazení.
- Transformace vygenerovaného XML výstupu nemusí být nutně realizována pouze do HTML, ale může být použito ke generování různých výstupních formátů a to vždy nad stejným aplikačním kódem.

2.5.1 XML kolektor

Pro účely shromažďování aplikačních výstupů je nutné zavést určitý „sběrač dat“, který je v rámci metodiky ETWA nazýván *XML kolektor*. Jeho primárním významem je strukturované ukládání dat z jednotlivých mezivýstupů aplikačních součástí. XML kolektor může být při konkrétní implementaci realizován např. jako sdílená funkce, třída (resp. objekt), kolekce nebo pole hodnot či objektů, apod. Ačkoliv je tento prvek nazýván jako *XML* kolektor, není nutné do něho na straně aplikace vkládat přímo XML kód, nýbrž hierarchická data sestavená na základě prostředků programovacího jazyka.

2.5.1.1 Požadavky

Pro formalizaci základní součásti metodiky ETWA pro zavedení XML mezivrstvy jsou vymezena následující pravidla a předpoklady *XML kolektoru*:

1. Prostředky kolektoru jsou dostupné všem funkčním částem aplikace (jsou tedy globální)
2. Strukturované úložiště dat je jednotné napříč všemi částmi aplikace
3. Poskytuje služby pro přidání objektu do kolektoru (hodnoty, pole, objekty, texty, ...)
4. Poskytuje služby pro kategorizaci vkládaných dat minimálně na následujících úrovních:
   (a) *data* - nejběžnější typ informace určený pro přímé zobrazení uvnitř webové stránky (je dále vnitřně strukturován),
   (b) *status* - statusové informace a hodnoty, které určují stav služby, výsledek zpracování, odpovědi externích systémů, apod.,
   (c) *chyba* - speciální oblast obsahující chybová hlášení, identifikace chybových stavů,
   (d) *servis* - informace, které nejsou určeny k přímému zobrazení ve webové stránce – slouží ke správnému nastavení výstupních formátů, zobrazení dat, apod.,
   (e) *řízení* - informace, které nejsou určeny k přímému zobrazení ve webové stránce – určují zdrojový systém, cílový systém, identifikaci zpracovávané úlohy, vstupních datech apod.

Jestliže je kolektor implementován např. jako třída, může obsahovat i další volitelné funkce, které sdruží veškerou práci s daty a XML do jednoho místa a XML kolektor tak bude zajišťovat veškeré funkce transportní vrstvy ve smyslu metodiky. Jsou to následující funkce, které budou podrobněji popsány v následujících oddílech.

1. Nástroje pro automatický sběr dat statických objektů (elementů)
2. Nástroje pro práci se slovníky
3. Nástroje pro práci s jazykovými elementy
4. Nástroje pro zpracování customizačních nastavení
5. Serializátor datových objektů do XML
6. Validátor dat - ověření dat vůči definovanému schématu

Vlastní funkčnost XML kolektoru by měla zajišťovat automatizaci řídících informací a zajišťovat rozhraní ke všem ostatním typům informací. Ačkoliv se jedná o kolektor XML dat, rozhraní musí být poskytováno na úrovni programovacího jazyka, aby nemusel programátor definovat strukturu XML a zaobírat se XML reprezentací dat. Zavedení XML kolektoru odstíní programátora od SGML jazyků. Naopak v kódu aplikace by se tak neměly již vyskytovat deklarace typu `print`, `echo`, apod. zobrazující výstupy přímo do stránky. Všechny tyto výstupy aplikace by měly být předávány do XML kolektoru. Porovnání dvou

Příklad 2.5.2 Porovnání standardního zápisu a s použitím XML kolektoru v jazyce PHP.

```
/* standardní přístup */
$result = mysql_query("select id, image, productname from products");
while ($item = mysql_fetch_object($result)) {
   echo 'ID produktu:' . $item->id;
   echo '<img src="'.$item->image.'" />';
   echo '<h3>'.$item->productname.'</h3>';
}

/* XML kolektor */
$result = mysql_query("select id, image, productname from products");
while ($item = mysql_fetch_object($result)) {
   $products[] = $item;
}
$xmlk = new XMLKolektor();
$xmlk->addData('products', $products);
$xmlk->addStatus('foundItems', count($products));
```

přístupů je patrné z příkladu 2.5.2. Jestliže jsou programátoři zvyklí na používání některého z frameworků, není pro ně podobný způsob ničím zásadně novým (porovnejte s příkladem CakePHP v příkladu 2.5.1) jen je striktně požadováno oddělení různých typů dat (v tomto případě *status* a *data*) a doplňování také všech aplikačně závislých informací (jako je například počet záznamů ve výpisu). XML kolektor musí být naplněn daty tak, aby na straně prezentační vrstvy nebylo nutné jakékoliv údaje dopočítávat, či získávat pomocí konstrukcí programovacího jazyka. I když se může na první pohled zdát, že tedy programátor musí doplňovat do výstupu další informace, které ve standardním přístupu nemusel, není tomu tak. Vezmeme-li například doplňování počtu nalezených záznamů, který se může zdát jako údaj doplňovaný navíc, je potřeba vzít v úvahu, že programátor tento údaj i v klasickém přístupu zjišťuje za účelem výpisu cyklů, počtu stránek a nastavení stránkování, apod., jen není vyžadováno jeho přímé zobrazení. Podobných příkladů by bylo možné uvést více.

2.5.1.2 Serializace

Serializace je obecně používaný postup pro uložení datových struktur (proměnných, polí, kolekcí, objektů, apod.) do formátu, ve kterém je možné tato data uložit do databáze, na disk, případně poslat prostřednictvím rozhraní jiné aplikace. Cílem serializace je objekt „zabalit“ a uložit pro použití v budoucnu (např. při opětovném přihlášení uživatele), nebo jej odeslat do jiného systému, kde je možné serializovaný objekt obnovit a dále s ním pracovat. Kromě možností překonání omezení transportních vrstev nebo relačních databázových úložišť, umožňuje serializace elegantní formu klonování objektů programovacího jazyka. Opačným procesem, tedy *deserializací*, se z došlých textových/binárních dat rekonstruuje původní datová struktura. Programovací jazyky podporují různé typy serializování, některé typy jsou pouze binárním kódem, jiné čitelným textovým výstupem ve formátu, který umožní zpětné převedení. Serializace se používá již delší dobu nicméně až s definicí XML formátu se podpora serializace rozšířila mohutněji i do použitelnější podoby v jednotlivých jazycích. Právě

Příklad 2.5.3 Ukázka obsahu XML kolektoru na úrovní programového kódu.

```
Array
(
    [general] => general Object
        (
            [task] => products.list
            [output] => products.list.template
            [parameters] => Array
                (
                    [parameter] => parameter Object
                        (
                          [name] => manufacter
                          [value] => Apple
                        )
                )
        )
    [service] => service Object
        (
            [loggedUser] => waUser Object
                (
                    [id] => 815
                    [wholename] => Ladislav Vomáčka
                )

        )
    [errors] => errors Object (      )
    [status] => status Object
        (
            [foundItems] => 5
        )
    [data] => data Object
        (
            [products] => Array
                (
                    [item] => product Object
                        (
                            [id] => 1547
                            [image] => img/mb-pro.jpg
                            [productname] => MacBook Pro
                        )
                    ...
                )
        )
)
```

Příklad 2.5.4 Serializace objektu do XML v PHP/PEAR.

```
$serializer = new XML_Serializer();
$serializer->setOption("addDecl", true);
$serializer->setOption("encoding", "utf-8");
$serializer->setOption("rootName", "etwa");
$serializer->setOption("indent", "    ");

// Serializace objektu XML kolektoru $xmlk
$serializer->serialize($xmlk);
print_r($serializer->getSerializedData());
```

XML umožnilo, aby byly serializované objekty čitelné nejen strojem, ale také člověkem a aby bylo možné serializované objekty přenášet nejen mezi systémy napsanými ve stejném jazyce, ale i napříč.

Výhod XML serializace využívá i návrh metodiky ETWA, kde je doporučena pro vytváření XML vrstvy mezi aplikačním kódem a prezentační vrstvou. Data nashromážděna do jednotného úložiště pomocí XML kolektoru představují datový vstup pro serializaci. Aby byla tato data následně dobře použitelná pro účely prezentační vrstvy je nutné zavést pravidla a ta prostřednictvím pravidel XML kolektoru vyžadovat. Do serializace tak vstupuje jeden velký datový objekt, obsahující veškerá data, která jsou požadována k předání do prezentační vrstvy, ve struktuře definované v oddílu 2.5.1.1. Jestliže je XML kolektor navržen dostatečně efektivně a obsahuje data ve struktuře podobné na příkladu 2.5.3, je pak vlastní vytvoření XML pro účely prezentační vrstvy velice jednoduché a postačí k němu využít některý ze standardně používaných serializátorů objektů.

Tyto jsou dostupné přímo v základní instalaci např. ASP.NET (`XmlSerializer` - nezávisle na zvoleném .NET jazyku) nebo Perlu (`XML::Generator::PerlData`, `XML::Simple`, nebo `XML::Writer`). Pro skriptovací jazyk PHP neexistuje nativní serializátor, ale obdobně jako v Perlu je možné využít možností `DOMDocument` (PHP5) a nebo je možné využít například `XML_Serializer` dodávaný v balíku rozšíření nazvaném PEAR[2]. Konkrétní implementace serializace v rámci metodiky ETWA je tedy naprosto jazykově nezávislá a jestli bude XML kolektor řešen v PHP a objektově jako na příkladu 2.5.2, nebo v Javě, Perlu či jiném jazyku na bázi např. globálního asociativního pole je necháno naprosto na rozhodnutí konkrétního uživatele. Vlastní převedení obsahu XML kolektoru do formátu XML může být implementováno jako funkce vlastního kolektoru, neboť se jedná pouze o triviální operaci serializace proměnné do XML. Ukázka, jak například serializovat data XML kolektoru (`$xmlk`) pomocí PHP/PEAR, je zobrazena na příkladu 2.5.4. Výsledkem serializace je pak XML výstup demonstrovaný na příkladu 2.5.5 (porovnejte obsah příkladů 2.5.3 a 2.5.5)

2.5.1.3 Business data

Při podrobnější analýze dat vkládaných prostřednictvím programového kódu do kolektoru, můžeme na tato data nahlížet jako business objekty, tedy data strukturovaná na úrovni

[2] `http://pear.php.net/`

Příklad 2.5.5 Ukázka obsahu XML kolektoru.

```
<?xml version="1.0" encoding="utf-8"?>
<etwa>
    <general>
        <task>products.list</task>
        <output>products.list.template</output>
        <parameters>
            <parameter>
               <name>manufacter</name>
               <value>Apple</value>
            </parameter>
        </parameters>
    </general>
    <service>
        <loggedUser>
            <id>815</id>
            <wholename>Ladislav Vomáčka</wholename>
        </loggedUser>
    </service>
    <errors />
    <status>
        <foundItems>5</foundItems>
    </status>
    <data>
        <products>
            <item>
                <id>1547</id>
                <image>img/mb-pro.jpg</image>
                <productname>MacBook Pro</productname>
            </item>
            ...
        </products>
    </data>
</etwa>
```

business požadavků na fungování aplikace. Cílový formát XML, který díky vlastnímu strukturování na bázi hierarchie objektů, zanořováním informací a pojmenováváním dat umožňuje data členit dle definice business dat. Následně generovaný formát XML je samopopisný a i bez interpretace do podoby výstupu webových stránek je srozumitelný pro účely testování a ověřování funkčnosti aplikace na bázi business objektů. Struktura dat uložených v kolektoru, tak jak je uvedená v oddílu 2.5.1.1, je poměrně univerzální a z hlediska zajištění provozu aplikace postačující. Kde musí každý tým využívající metodiku ETWA přikročit k personalizaci je oblast datová, kterou je doporučeno personalizovat a rozšířit o vlastní definice využívaných typů dat v závislosti na vytvářené webové aplikaci.

2.5.1.4 Rozšíření vlastností XML kolektoru

Jak již bylo zmíněno ve volitelných požadavcích na funkčnost XML kolektoru je z hlediska variability webových aplikací vhodné zavést také rozšiřující funkce kolektoru. Ty by měly řešit především zvláštnosti vícejazyčných webových aplikací a přizpůsobení nastavení aplikace konkrétně přihlášenému uživateli. V případě vícejazyčných a klonovatelných aplikací[3] je možné se setkat s kopírováním identických šablon za účelem změny některých popisků, které jsou jazykově závislé, zahrnutím, či vypuštěním některého elementu, apod.. Vzniká tak sada samostatných šablon, které se liší pouze v určitých datech, strukturně ale všechny generují stejné uživatelské rozhraní. Obdobný postup je nezřídkakdy používán i v případech, kdy se aplikace upravuje pro různé klienty, nebo účely (např. se vyměňuje logo, záhlaví, zápatí a některé další identifikační prvky). Na první pohled se může zdát tento postup jako korektní samostatná jazyková nebo obsahová mutace se generuje na základě vlastní šablony. Ale nejpozději ve fázi údržby se projeví největší úskalí, kterým je aktualizace a rozšiřování vlastností uživatelského rozhraní. Jestliže je například požadavkem vložit do všech jazykových verzí nový element, musí se toto provést násobně u všech šablon. Nemluvě o tom, že díky lidskému faktoru se na některé zapomene, nebo je v nich provedena implementace s chybou. Postupem času může tento postup vést až ke kolapsu celého systému.

Druhým případem, který částečně eliminuje nevýhody předchozího postupu, je jedna šablona s tzv. *rozhodovacími bloky*. V praxi to znamená, že v místě, kde má být např. statický popisek v určitém jazyce, je vložen blok podmínek, který na základě zvoleného jazyka naplní dané pole správným výrazem (viz příklad 2.5.6). Kromě toho, že už vlastní podmínka musí být napsána syntaxí použitého programovacího jazyka, která ale nepatří do šablony, ale do aplikační části, je na příkladě také demonstrováno, že i vlastní výpis jazykově rozdílné části je pak mnohdy realizován prostřednictvím aplikačního kódu. K tomuto přistupují většinou programátoři proto, že je zbytečné a nepřehledné ukončovat kontejner aplikačního kódu (ohraničený <? a ?>) kvůli výpisu textu a pak jej znovu začínat kvůli dokončení syntaxe podmínky. Tímto způsobem je ale šablona degradována a vznikne tak meziprodukt, který není ani dobrou šablonou, ani konzistentní části aplikačního kódu. Zodpovědnost za takovýto výstup pak není možné přiřadit konkrétní roli, což je v rozporu s požadavky na metodiku ETWA.

[3] Jedno aplikační jádro *klonované aplikace* se využívá pro větší množství vzhledově různých aplikací (např. obchody provozované a upravené pro značku konkrétního dealera.

Příklad 2.5.6 Vložení jazykově variabilních částí formou podmínek.

```
<h1><?= $out->category; ?></h1>
<p><?= $out->description; ?></p>
<h2>
<? if ($out->language == 'cz') { echo 'Seznam produktů'; } ?>
<? if ($out->language == 'en') { echo 'List of products'; } ?>
<? if ($out->language == 'de') { echo 'Produktskatalog'; } ?>
</h2>
```

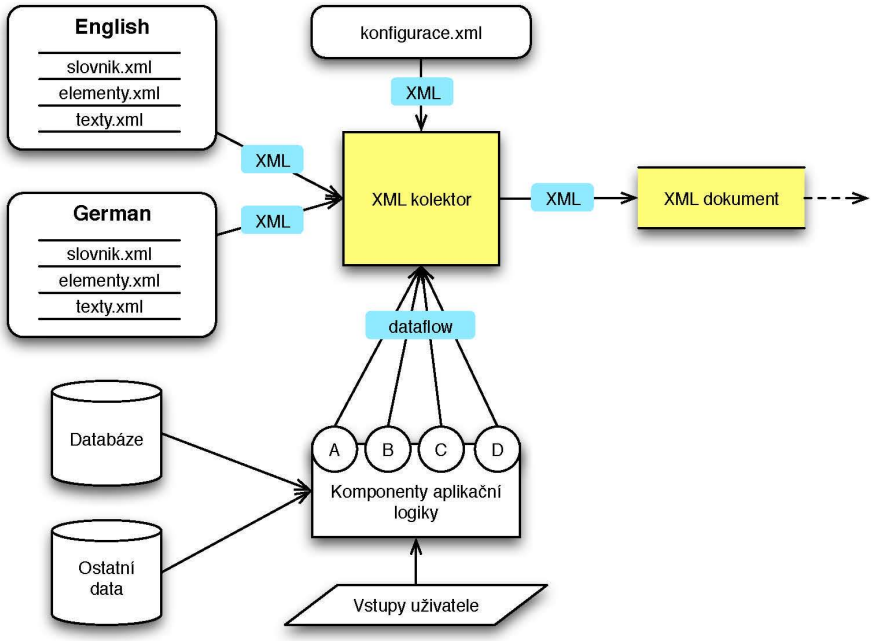
2.5.1.5 Vícejazyčné a klonované aplikace

Jestliže jsou webové aplikace vytvářeny se záměrem využití vícejazyčného rozhraní, je nutné v prezentační vrstvě zobrazovat určité informace v různých jazycích, které by v případě jednojazyčné verze byly statickou součástí šablony. Aby byly dodrženy požadavky ETWA, tj. aby šablona zásadně neobsahovala funkční logiku a ani části programového kódu je nutné vyřešit předávání jazykově rozdílných dat do šablony z aplikační vrstvy, resp. XML mezivrstvy. Rozličnost jazykových verzí nemusí být dána jen různým překladem určitého textu, popisku, či nadpisu, ale může být také na úrovni obsahové a strukturní. Jazykové verze nejsou většinou přesnou kopií verze v primárním jazyce a proto některé z nich obsahují určité elementy a jiné verze zase ne. Předávání těchto dat z XML mezivrstvy je vhodné pro udržení jednotného designu celé aplikace, použití jednotné šablony pro všechny jazykové verze, udržení jednotného vzhledu aplikace, ovládání, jednoduchosti, ale i potřebné flexibility pro budoucí úpravy prováděné v rámci údržby a rozvoje aplikace.

ETWA doporučuje zahrnout multijazyčnou podporu do rozšířených vlastností XML kolektoru, který při sestavování datové základny bude brát v úvahu nejen datové výstupy aplikace, jejichž jazyková rozdílnost bude zabezpečena již v rámci aplikačního zpracování a jejímž návrhem se ETWA nezabývá, ale i statické *XML celky*, které automaticky začlení je do výsledného XML výstupu. XML celky jsou zde míněny samostatné části XML kódu zaobalené do identifikačního elementu, získané např. ze samostatných souborů, konfiguračních databází, apod. Důležitým prvkem této rozšířené vlastnosti XML kolektoru je správné začlenění kódu v cílovém XML formátu.

Data, jakými jsou právě slovníky, obsahy statických elementů, nebo texty stránek, není nutné při každém požadavku na zobrazení stránky nutné opakovaně generovat. Proto by jejich zpracování představovalo zbytečný čas aplikačního zpracování. Musela by být provedena I/O operace pro načtení souboru z disku do paměti, z ní by byla data načtena do obsahu proměnných (např. objektu), tento objekt by musel být vložen do XML kolektoru a ten by tato data uložil opět do XML spolu s ostatními daty. Z hlediska struktury XML kolektoru by tato data měla být součástí úrovně *servis* (viz kap. 2.5.1.1), v případě rozsáhlejších dat (dlouhé texty, rozsáhlé číselníky, apod.) součástí úrovně *data*.

Způsob integrace samostatných XML souborů se statickým obsahem vychází z jednoho z předpokladů ETWA; za uživatelské rozhraní, tedy statická data, má zodpovědnost kodér. Jestliže by nebyla tato data různá v jednotlivých jazycích, integroval by je do šablony ve zvoleném jazyce kodér. Statické soubory s XML daty mají tu výhodu, že jsou v jazyce, který



Obrázek 2.7: Skládání dat do XML kolektoru

ovládá kodér a ten si je může pro tyto účely připravit sám. Oddělení jazyků používaných jednotlivými rolemi je dalším požadavkem metodiky ETWA. Statická data pro jednotlivé jazyky/klony v XML formátu jsou zodpovědností kodéra. Ten data připraví v jednotné formě pro všechny jazyky/klony a uloží do dohodnuté adresářové struktury. Vložení dat aktuálně používaného jazyka do XML zdroje pro prezentační vrstvu zajistí automaticky XML kolektor. Pro skládání obsahu uživatelského rozhraní tak jsou k dispozici informace pouze v jednom konkrétním jazyce. Názorně je způsob skládání dat do kolektoru znázorněn na diagramu 2.7.

2.5.2 Validace dat

Ve výchozích vlastnostech metodiky ETWA bylo v jednom z hlavních cílů stanoveno oddělení aplikační a prezentační vrstvy a zavedení možnosti paralelní práce kodéra a programátora. Toto bezesporu naplňuje již vlastní zavedení XML kolektoru a celé XML mezivrstvy.

Aby bylo možné eliminovat současné problémy v rolích kodéra a programátora identifikované v oddílu 2.1.1 a následující, je nutné dohodnout strukturu dat, která si mezi sebou budou oba zdánlivě nezávislé systémy vyměňovat. Toto by bylo možné zajistit procesně na neformální úrovni, přidat doporučení o zahrnutí do fáze analýzy, vývojových postupů, apod. Nicméně podle zkušeností a popsaných problémů současných webových frameworků, je to toto pouze nesplněné přání všech účastníků vývoje webové aplikace, přičemž každý z nich svým dílem přispívá k postupnému nedodržování a odchylkám od těchto dohodnutých pravidel. Proto metodika ETWA zavádí validaci generovaných XML dat ještě předtím, než jsou předána do XML mezivrstvy. Důvody pro zavedení validace dat jsou následující:

1. Jasné zadání obsahu a identifikaci dat pro kodéra i programátora

Příklad 2.5.7 XML dokument vyhovující Relax NG schematu na příkladu 2.5.8.

```
<?xml version="1.0" encoding="utf-8"?>
<etwa>
    <general>
        <task>products.list</task>
        <output>products.list.template</output>
        <id>15221</id>
        <parameters>
            <parameter>
               <name>manufacter</name>
               <value>Apple</name>
            </parameter>
        </parameters>
    </general>
...
</etwa>
```

2. Definice validačních schémat podporuje pohled na data jako na business objekty
3. Vlastní validace zajišťuje automatické testování aplikace na systémové úrovni
4. Výrazně snižuje riziko nekonzistentních dat při transformaci do výsledného formátu
5. Umožňuje v jednom kroku validovat nejen výstupní, ale i vstupní data
6. Podporuje dodržování definovaných pravidel po celou dobu vývoje aplikace

V oddílu 2.1.1 bylo uvedeno, že kodér při tvorbě uživatelského rozhraní neví jak se jmenují proměnné, které mají být zobrazeny, jestli je se jedná o hodnotu či pole hodnot, v jakém kontextu je použit název proměnné (např. `$nazev` může obsahovat název kategorie, stejně jako název produktu). Proto musí být na samém počátku dohodnuto a popsáno, jaká data a v jaké struktuře si aplikační a prezentační vrstva vyměňují. Toto je jedním z předpokladů pro důsledné rozdělení rolí programátora a kodéra. Pokud by bylo použito standardních šablonovacích nástrojů bylo by nutné pro definici a ověření vyměňovaných dat použít zvláštního funkčního prvku, který by umožňoval zapsat a vygenerovat popis datové struktury, která bude využívána pro zobrazení v šablonč. Aby kodér mohl připravovat své výstupy nezávisle na programátorovi – tedy nezávisle na prvku uvnitř programového kódu – musel by validační část připravit programátor hned v úvodu své práce. Data, jejich typ a vzorek by byla vygenerována ve formátu, který by byl kodérovi dostatečně srozumitelný. Tento způsob s sebou nese poměrně mnoho zbytečných, a ve finálním řešení, nepříliš efektivních, úkonů.

I v tomto bodě se ukazuje, že metodikou ETWA navržený formát dat XML, má v tomto řadu nesporných výhod. Jako efektivní metoda pro dohodu o podobě dat předávaných mezi aplikační a prezentační vrstvou, se jeví *vzorový XML soubor*, obsahující všechna potenciální data. Tato forma je pro člověka mnohem srozumitelnější než přímá tvorba validačních schémat a pravidel. Soubor s daty podle specifikace může, ba dokonce by i měl, připravit kodér. Tvorba XML souboru představuje práci se značkovacím jazykem typu SGML a proto je za ni, dle dříve popsaných pravidel, zodpovědný kodér. Vzorový soubor se všemi potenciálními

Příklad 2.5.8 Relax NG schema.

```
<element name="etwa" xmlns="http://relaxng.org/ns/structure/1.0"
   datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

   <element name="general">
      <element name="task">
         <text/>
      </element>
      <element name="output">
         <text/>
      </element>
      <element name="id">
         <data type="int"/>
      </element>
      <optional>
         <element name="parameters">
            <zeroOrMore>
               <element name="parameter">
                  <element name="name">
                     <text/>
                  </element>
                  <element name="value">
                     <text/>
                  </element>
               </element>
            </zeroOrMore>
         </element>
      </optional>
   </element>
...
</element>
```

datovými/business objekty je připravován na základě analýzy, po konzultaci s analytikem, který zpracovával zadání. V některých případech je možné přenést odpovědnost za vytvoření vzorového XML souboru na analytika, který zná používaná data v prvopočátcích projektu nejlépe z celého projektového týmu. Takto připravený soubor by měl být následně odsouhlasen programátorem a kodérem, aby odpovídal struktuře dat dle možností programového kódu a také s ohledem na definici business objektů a jejich prezentaci v uživatelském prostředí. Protože na základě vzorového dokumentu není možné provádět validaci budoucích dat, která budou v aplikaci vznikat, je nutné na základě něho připravit validační schéma. V případě XML existují standardizované nástroje, určené přímo pro validaci XML.

K vytvoření definice elementů bylo v minulosti využíváno převážně schéma dokumentu ve formátu *DTD* (Document Type Definition). Mezi nejčastěji zmiňovanými nevýhodami DTD je nemožnost kontroly datových typů a absence podpory jmenných prostorů. Proto vznikly další alternativy, na jejichž základě bylo možné vytvořit parser pro lepší validaci XML dokumentů. V zápise DTD se používá jednoduchá syntaxe připomínající regulární výrazy, ukládaná do speciálně formátovaných textových souborů. Tento způsob zápisu sám

odporuje důvodům proč je používáno samotné XML, a proto je u novějších jazyků pro popis schématu dokumentu používáno přímo zápisu v syntaxi XML.

Za nejdůležitějšího následovníka DTD lze považovat jazyk *XDR* (XML-Data Reduced). Vznikl jako zjednodušená verze návrhu jazyka *XML-Data*, který nebyl příliš vázán na XML, ale především umožňoval definovat charakteristiky tříd objektů. Obecně platným a přijatým standardem se později stal jazyk *XML Schema* konsorcia W3C[4], který se hned po DTD stal (a dodnes zůstal) nejpoužívanějším. XML Schema se vyvinulo právě z XDR, ale ani toto schéma není zcela ideální. Důvodem je opět složitost a občasná nepřehlednost tohoto schématu, což bylo některým jeho autorů zřejmé dříve, než byl jazyk XML Schema uznán standardem.

Proto již v průběhu vývoje schématu XML Schema vznikala poměrně odlišná definice pro *Relax NG*[5]. Jeho systematický přístup vytváření gramatiky nového schématu je podobný oblíbenému DTD a narozdíl od XML Schema, např. striktně nevyžaduje definici datových typů, apod..

Zcela odlišný přístup a ponekud jinou větev vývoje pak představuje schéma *Schematron*, které umožňuje definovat sadu podmínek, které musí být v dokumentech splněny. Schematronové schéma je velmi silným nástrojem pro validaci, protože žádné z předchozích schémat není např. schopno popsat taková omezení, kdy obsah jednoho elementu musí mít větší hodnotu, než obsah elementu jiného, nebo vztahy mezi daty uloženými v separátních dokumentech. Schematron bývá tedy používáno jako doplněk ke klasickým schématům vytvořeným v jazycích XML Schema nebo Relax NG. Obě zmiňovaná schémata umožňují vkládání výroků Schematronu přímo do popisného dokumentu.

K dispozici je tedy celá řada různě vhodných nástrojů a je třeba určit, který z nich je nejvhodnější k řešení konkrétního problému. Volbu je možné zjednodušit, jestliže je DTD vhodné pouze pokud není třeba podpory jmenných prostorů a v případě, že je dostačující použití proprietálního formátu zápisu DTD pravidel. V opačném případě je lepší volit mezi XML Schema a Relax NG. XML Schema je vhodné zvolit tehdy, pokud je řešení takto limitováno komerčními vývojovými nástroji. Naopak Relax NG poskytne větší jednoduchost a kromě XML zápisu podporuje i možnost zápisu kompaktní formou.

Metodika ETWA je, dle úvodních požadavků, orientována na využívání standardizovaných nástrojů a rozdělení na SGML a programovací jazyky, je proto požadavkem možnost zápisu validačního schematu ve formátu XML. Toto zcela určitě nesplňuje DTD. XML Schema bylo překonáno Relax NG a proto je možné v ETWA využít *Schematron* nebo *Relax NG*. Volba je ponechána implementátorům. V této práci je pro validaci XML zvoleno schema Relax NG a to zejména z následujících důvodů:

1. Dostupné zdroje v dobré kvalitě pro účely seznámení se s validačním schématem – výhoda pro zaškolení začátečníků
2. Dostupné nástroje pro provedení validace XML vůči Relax NG schématu
3. Možnost převést Relax NG schéma do jiného schématu, např. DTD

[4]World Wide Web Consorcium - `http://www.w3.org/`
[5]REgular LAnguage for XML Next Generation - `http://relaxng.org/`

Příklad 2.5.9 Relax NG schema v kompaktní formě.

```
element etwa {
  element general {
    element task { text },
    element output { text },
    element id { xsd:int },
    element parameters {
      element parameter {
        element name { text },
        element value { text }
      }*
    }?
  },

...
}
```

4. Možnost převedení z/do úsporné (compact) formy
5. Podpora jednoduchého zápisu, ale i datových typů z XML Schema (tj. standardizovaných W3C)
6. Umožňuje vytvoření schéma ze vzorového XML dokumentu

Dvojice příkladů 2.5.8 a 2.5.9 představuje stejné schéma, zapsáné dvěma způsoby, které Relax NG nabízí plnohodnotný XML zápis (RNG) a kompaktní zápis (RNC). Díky nástroji *Trang*[6] je možné schéma převádět do jiných schémat. Podporovány jsou Relax NG (RNG), Relax NG (RNC), DTD, XSD a XML. Možnosti transformace jsou uvedeny na příkladu 2.5.10. Použití Relax NG schématu má právě díky těmto konvertorům jednu nespornou výhodu. Jak již bylo zmíněno na začátku tohoto oddílu, připravují společně kodér, analytik a programátor definici dat, která budou zobrazena v šablonč na počátku realizační fázc. K tomuto účelu je využíváno vzorového XML dokumentu, který je v budoucnu nahrazen identicky strukturovaným dokumentem z aplikace. Do té doby může vzorový dokument využívat kodér k účelům tvorby šablony. Může být následně doplňován a rozvíjen, ale vždy pouze na základě shody všech stran, resp. v souladu s validačním schématem. Pokud bude využito javového nástroje Trang a jeho možnosti konverze XML → RNG, je tak možné velice jednoduše (viz příklad 2.5.10) vytvořit validační schéma použitelné pro následnou validaci dat plynoucích z aplikace do šablony. Jestliže se tedy zdálo, že pro zavedení XML validace bude nutné zavádět další jazyk, nebo složitě vytvářet validační schémata, bylo toto riziko díky použití jednoho nástroje výrazně eliminováno. Pro validaci dle Relax NG schématu existují již i nativní funkce např. v PHP5. Proto lze tedy Relax NG v kombinaci s ostatními nástroji doporučit při implementaci metodiky.

Do procesu vlastní validace vstupují dva soubory (dva datové proudy). Jedním jsou XML data, která aplikace produkuje a druhým je vlastní schema. Validátor na základě po-

[6] *Trang* je multiformátový konvertor schemat založený na RELAX NG `http://www.thaiopensource.com/relaxng/trang.html`

Příklad 2.5.10 Konverze Relax NG schema.

Konverze ze vzorového XML souboru do XML schema Relaxu:

```
$ java -jar trang.jar -I xml -O rng etwa.xml relax.rng
```

Konverze z XML formy do kompaktní formy Relaxu:

```
$ java -jar trang.jar -I rng -O rnc relax.rng relax.rnc
```

Konverze z kompaktní do XML formy Relaxu:

```
$ java -jar trang.jar -I rnc -O rng relax.rnc relax.rng
```

rovnání dat dokáže identifikovat, jestli se jedná o XML data podle pravidel definovaných např. v RNG, případně dokáže identifikovat chybu, která se v dokumentu nachází. Nejčastějšími chybami jsou:

- Použití nedefinovaného elementu – ve výstupu se nachází data, která nebyla dohodnuta při tvorbě schématu a proto by ani v šabloně nemohla být zobrazena. Pokud je nutné přidat další data do výstupu je nutné v tomto smyslu upravit jak vzorový XML dokument, tak validační schéma. Změny samozřejmě musí zohlednit také kodér v šabloně.
- Nekonzistentní výstup – jedná se většinou o chybějící uzavření elementu, překlepy v názvu otevírajícího a uzavírajícího elementu, chybějící uvozovky u hodnot atributů, apod.
- Neshoda datového typu – nejen v RNG validačním schématu je možné u kritických dat definovat také datový typ, ve kterém musí vložená data být. Není příliš účelné striktní nastavení pravidel na všechny předávané parametry, ale pouze u parametrů, kde by jiný datový typ mohl způsobit problémy ve zpracování.
- Chybějící povinná součást - v pravidlech validace je možné určit, jestli element musí být ve výstupním XML dokumentu vždy uveden, případně je-li volitelný. Je možné také definovat kolik potomků má element mít – označené jako *minimálně jeden, žádný nebo více*, nebo *pouze jeden*.

K validaci je možné opět použít různé nástroje, které podporuje daný programovací jazyk, stejně tak jsou různě prezentovány výsledky validace. Na příkladu 2.5.11 je využito nástroje *Jing*[7], který na základě validace dokumentu `etwa-sample.xml` identifikoval 3 chyby vůči schématu, definovaném v dokumentu `relax.rng`. První chyba identifikuje chybu konzistence, konkrétně překlep v názvu uzavíracího elementu, dále potom neshodu datového typu a chybějící povinnou součást. Zároveň jsou identifikovány řádky a pozice na řádku, kde se přesně daná chyba ve zdrojovém dokumentu nachází.

[7] *Jing* je konzolová aplikace implementovaná v Javě, podporuje Relax NG v obou syntaxích `http://www.thaiopensource.com/relaxng/jing.html`

Příklad 2.5.11 Ukázka výsledku validadace pomocí Jing a Relax NG schema.

```
$ java -jar jing.jar relax.rng etwa-sample.xml
./etwa-sample.xml:12:37: fatal: The end-tag for element type "name"
must end with a '>' delimiter.
./etwa-sample.xml:19:26: error: bad character content for element
./etwa-sample.xml:24:13: error: required elements missing
```

2.5.3 Realizace šablony

Ve standardních vývojových prostředích (Visual Studio, Builder, ...), ve kterých jsou vyvíjeny desktopové aplikace je vzhled aplikace řešen velmi triviálním způsobem. Do připraveného okna se vybere vhodný prvek (control) z předdefinované sady, která obsahuje všechny běžně používané prvky (např. strom, tabulka pro výpis seznamů - datagrid, posuvné ovladače - scrollery, textová editační pole, navigační menu, apod.). Vybraný prvek je umístěn na konkrétní pozici nebo ukotvení a na této pozici zůstává za jakýchkoliv okolností. Aby bylo možné do tabulkového výpisu zobrazit např. záznamy z databázového dotazu, postačí zavolat příslušnou metodu prvku datagrid a předat jí výsledek dotazu v proměnné, resp. objektu. Veškeré operace naplnění, zformátování, zarovnání, příprava posuvníků, obarvování řádků, apod. jsou již automaticky zajištěny a připraveny k použití v rámci daného prvku. Z hlediska rychlosti implementace se jedná o velice rychlý způsob vytvoření výsledku, bez sebemenší nutnosti alokovat jakoukoliv další roli pro vytvoření designu prvků, způsobu ovládání, zapojení do kontextu stránky, ... V tomto typu tvorby aplikací je kód uživatelského rozhraní generován automaticky a pouze v některých případech vyžaduje zásah informačního architekta, který prvky na obrazovce uspořádá tak, aby dávaly logický smysl, byly použity vhodné typy ovládacích prvků a aby uživatel nebyl zmaten při ovládání aplikace. Některá vývojová prostředí jako např. Visual Studio.NET umožňují vygenerovaný soubor pro popis uživatelského rozhraní zobrazit a editovat; tato možnost je ale využívána pouze minimálně.

Výstupním formátem webové aplikace je již od samotné podstaty webu, jak jej definoval Berners-Lee [2], jazyk HTML. V současné době již původní definice webového rozhraní pokročila a HTML se využívá již výlučně v kombinaci s kaskádovými styly CSS. Původní HTML, definované pro potřeby CERNu, používalo hypertextového značkování textu především na úrovni významové - nadpisy, odstavce, tabulky, definice, seznamy, apod. Postupně byly do HTML doplňovány další elementy, které rozšiřovaly možnosti úpravy vzhledu dokumentu (barvy, fonty, zarovnání, odsazení, apod.). To mělo za následek nárůst obsahu zdrojového HTML kódu a významové elementy byly zaobaleny do množství elementů formátovacích a nejeden takový byl obsažen uvnitř elementů. Tato situace byla nejen z pohledu rozvoje HTML neúnosná a neprogresivní, ale i pro vlastní údržbu webu byla poměrně komplikovaná. Proto bylo definováno CSS s cílem přesunout veškeré formátovací vlastnosti do separátních kaskádových stylů a vrátit tak HTML jeho popisnou funkci. HTML je obdobně jako XML jazykem vycházejícím ze SGML a proto tedy v metodice ETWA kompletně spadá do zodpovědnosti kodéra a to včetně CSS, které se SGML definici vymyká.

Jedním z cílů navrhované metodiky je striktnější oddělení rolí, ale zároveň také oddělení

jazyků. V mnoha případech, a to ať už se jedná o klasické funkční programování, nebo využití nejrůznějších MVC[8] framworků, nebo šablonovacích systémů, se hodnoty do šablony vypisují pomocí konstrukce pro vložení programového kódu (např. `<b><?= $jmeno; ?></b>`), nebo vlastním pseudokódem šablonovacího jazyka (např. `<b>#{jmeno}#</b>`), který ale stejně ve svém důsledku umožňuje vložení a spuštění programového kódu. Toto provázání prostředí šablony s programovým kódem, kromě vlastního míchání jazyků, představuje také hrozbu v možnostech vložit více než jen pouhý výpis obsahu proměnné. A jak říká jeden z originálních Murphyho zákonů: „jestliže něco může být špatně, bude to špatně", je i v případě snahy o oddělení šablon od programového kódu možné pozorovat, že buď z důvodu nepochopení principu nebo s přibývajícím časem, či složitostí, se do šablony přesouvají části kódu, které by v ní být neměly. Na výsledném vzhledu a fungování toto ovšem poznat není. Nezřídkakdy jsou do šablony vkládány celé části aplikace, a to včetně případů obsahujících přímé dotazy do databáze, výpočty, či jiné podobné konstrukce.

Proto je návrh metodiky ETWA připraven tak, aby byly případné možnosti integrace s programovým kódem eliminovány a programové konstrukce tak nebylo možné použít uvnitř šablony, čímž se eliminují i samotné pokusy o porušení definovaných pravidel a udrží se tak konzistentnost a rozšiřitelnost aplikace pro budoucí údržbu. Stejně jako tomu bylo při návrhu transportní mezivrstvy a formátu dat, předávaných do této mezivrstvy, bude i v případě šablony respektován požadavek stanovený na počátku a tedy že metodika bude v maximální možné míře využívat standardizovaných, podporovaných a rozvíjených formátů. Proto bylo pro účely mezivrstvy stanoveno XML, zaštiťované konsorciem W3C. Důvodem pro stanovení podobných předpokladů je možnost rozvoje jednotlivých účastníků vývoje projektu, jejich znalostí a dovedností v jazyce, který nebude využit jen v rámci jednoho řešení, jednoho šablonovacího nástroje, ale jeho znalost bude využitelná i v jiných případech. Obdobně tomu je u nově příchozích účastníků, kteří není nutné zaškolovat do používání určeného frameworku. Jestliže bychom se omezili na určitou formu zápisu hodnot do šablon (pseudojazyk), zaškolili na tento jazyk příslušné osoby a následně by vývojář frameworku jeho vývoj ukončil, znamenalo by to další dodatečné náklady na výběr nového frameworku, nové postupy práce, nové způsoby používání s opět nejistým koncem. V dnešní době lze velice dobře pozorovat trend k vývoji vlastních frameworků. I na ně se vztahují stejná pravidla a stejná úskalí jako u veřejně dostupných a volně použitelných frameworků. Myšlenku, rozvoj i vývoj zpravidla podporuje samostatný tým, který se touto činností v rámci firmy zabývá a k níž také dostal pověření od svých nadřízených. Ostatní týmy takto připravený framework pouze využívají a případně sdělují zpětně své připomínky, či poznatky. Na základě vlastní zkušenosti se ale postupem času se vývojový tým začne orientovat na jiný postup a budovat tak nový framework, případně zbytek firmy začne podporovat jiný, veřejně dostupný framework, nebo se vývojový tým frameworku rozpadne, či management přestane vývoj podporovat a firma je opět bez nástroje. Důsledky jsou tedy obdobné jako u frameworků „cizích". Z dlouhodobého hlediska je tedy vhodné využívat nástrojů a postupů s garantovaným rozvojem.

[8] *MVC - Model View Controller* - je softwarová architektura, která rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých komponent tak, že modifikace některé z nich má minimální vliv na ostatní.

2.5.3.1 XSL šablony

Pokud shrneme požadavky na konzistentní a efektivní vzor šablony - musí být vytvořena na základě podporovaného jazyka, nebo nástroje a musí umožňovat diferenciaci rolí zejména tedy kodéra a programátora. Za šablonu musí být zodpovědný kodér a tedy i za její naplnění dodanými daty, jejichž dodávku připravuje v dohodnuté struktuře programátor. Z uvedeného plyne, že ideální šablonovací jazyk by měl vycházet ze specifikace SGML, aby byl srozumitelný kodérovi a vycházel ze stejných principů jako HTML (případně CSS). Už sama tato kombinace ukazuje, jakým způsobem metodika ETWA navrhuje realizaci šablony webové aplikace. Výše uvedená kritéria kompletně splňuje právě XSL[9], které reprezentuje rodinu jazyků umožňujících popsat, jak se mají XML data formátovat a převádět do zvoleného výstupního formátu. Specifikace XSL v sobě v podstatě zahrnuje následující tři jazyky:

- *XSL Transformace (XSLT)*: XML jazyk k převádění XML dokumentů,
- *XSL Formátovací Objekty (XSL-FO)*: XML jazyk pro specifikaci vizuálního formátování XML dokumentů,
- *XML Adresovací jazyk (XPath)*: jazyk k výběru částí XML dokumentu, který ale sám není XML jazykem. Je používán například v XSLT.

Specifikace těchto tří jazyků jsou dostupné ve formě W3C doporučení[10]. A protože na transportní vrstvě jsou data připravena ve validním XML formátu a cílem šablony je transformovat tato data do formy HTML stránky, představuje XSL ideální prostředek, splňující všechny požadované vlastnosti. Pro účely popisované metodiky je možné se omezit pouze na použití XSLT a XPath.

Soubor s XSLT stylem je sám o sobě XML dokument, protože používá syntaxi XML. Pro jeho editování je možné využít běžné XML editory, stejné jako pro účely tvorby HMTL dokumentů. V tomto ohledu tedy pro práci kodéra není třeba využívat jakéhokoliv dalšího nástroje. Většina, v současné době používaných, editorů zároveň kontroluje validitu vytvářeného kódu, což je dalším pozitivním kritériem pro volbu XSL. V dokumentu se přitom setkávají dva druhy značek řídící příkazy pro procesor a značky výsledného dokumentu (např. HTML tagy), nicméně oba druhy respektují pravidla zápisu XML. Aby bylo možné v jednom dokumentu kombinovat dvě sady značek, používají se tzv. *jmenné prostory* (namespaces). Před názvy všech elementů, které má XSLT procesor zpracovávat, se musí vložit speciální prefix `xsl:`. Celý styl přitom musí být uzavřen v elementu `stylesheet` |24|.

Není cílem této práce podrobně seznamovat se syntaxí XSL, která je podrobně popsána na webu W3C: `http://www.w3.org/Style/XSL/`, cílem je poukázat na vlastnosti XSL, které podporují postupy popsané metodikou ETWA. Na příkladu 2.5.12 jsou demonstrovány nejčastěji používané transformace zdrojových dat do výsledného XHTML formátu. V hlavní části šablony se formátují obecné části stránky, které budou zaobalovat veškerý obsah, ať už se, jako v tomto případě, bude jednat o výpis katalogu produktů, či obsah košíku, nebo

[9] XSL - *Extensible Stylesheet Language* - `http://www.w3.org/Style/XSL/`

[10] *Doporučení W3C* je posledním stupněm ratifikačního procesu pracovní skupiny World Wide Web Consorcium (W3C) zaměřené na standardizaci dané oblasti. Jedná se v podstatě o ekvivalent k publikovanému standardu v ostatních odvětvích.

Příklad 2.5.12 Ukázka šabolny v XSLT.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output encoding="windows-1250" method="html"></xsl:output>
<xsl:template match="/">
   <html xmlns="http://www.w3.org/1999/xhtml">
      <head>
         <title>ABC internetový obchod</title>
      </head>
      <body>
         <h1>ABC - velkoobchod počítači</h1>
         <p>Přihlášený uživatel:
            <xsl:value-of
                 select="/etwa/service/loggedUser/wholename" />
         </p>
         <xsl:apply-templates select="/etwa/data/products" />
         <hr />
         <p>Vytvořeno na základě metodiky ETWA</p>
      </body>
   </html>
</xsl:template>

<xsl:template match="/etwa/data/products">
   <h2>Katalog produktů</h2>
   <table>
      <tr><th>Kód</th><th>Název</th><th>Obrázek</th></tr>
      <xsl:apply-templates />
   </table>
</xsl:template>

<xsl:template match="/etwa/data/products/item">
   <tr>
      <td><xsl:value-of select="id" /></td>
      <td><xsl:value-of select="productname" /></td>
      <td>
         <xsl:element name="img">
            <xsl:attribute name="src">
               <xsl:value-of select="image" />
            </xsl:attribute>
            <xsl:attribute name="alt">
               Foto: <xsl:value-of select="productname" />
            </xsl:attribute>
         </xsl:element>
      </td>
   </tr>
</xsl:template>

</xsl:stylesheet>
```

o textovou stránku s informacemi platbě, kontakty, apod. V hlavní části šablon jsou v místech, kde se má vypisovat určitý prvek (obsah) vloženy odkazy na samostatné části šablony, definující způsob zobrazení daného prvku. Tato zanoření mohou být aplikována opakovaně a vytvořenou část kódu tak lze opakovaně použít pro zobrazení různých dat stejného typu. Šablona je vždy definována pro jeden záznam (např. řádek, produkt), který se ale ve zdrojových datech násobně opakuje. Syntaxe skutečně odpovídá definici SGML jazyka a kodérovi ve většině případů stačí základní sada několika jazykových konstrukcí.

2.5.3.2 Výhody XSL šablon

Velice jednoduchá šablona, jako je uvedena v příkladě 2.5.12, by se v případě použití klasického programovacího jazyka a přidružených šablon neobešla **bez různých programovacích konstrukcí**. Pokud budeme tato místa identifikovat konkrétně, tak by bylo nutné ověřit, **jestli byla do šablony zaslána vůbec nějaká data** o produktech a pokud ano, tak zpracování pustit dále k vytvoření definice tabulky, záhlaví sloupců, apod. (např. `if (count($aProducts) > 0)`). V XSL je toto řešeno velice elegantně a kodér se nemusí zajímat o to jestli data budou nebo nebudou, protože je zodpovědný za korektní zobrazení dat. Jestliže ve zdrojovém XML data pro danou oblast stránky chybí, procesor zpracovávající výstup, jednoduše nevyvolá zpracování podle dané šablony a proto se do výsledné stránky nevygeneruje vůbec nic. Druhým místem, kde by bylo v příkladě 2.5.12 nutné sáhnout po programátorské konstrukci by byl vlastní výpis produktů. Jelikož není známo, kolik produktů bude vypisováno a protože by měl být vzhled všech řádků tabulky stejný, definuje se v šabloně vzhled opakujícího se řádku. V klasickém přístupu bychom se **nevyvarovali konstrukcí smyček, cyklického výpisu** a zápisům podobným: `for ($item=0; $item<count($aProducts); $item++)`, které bychom jen stěží vysvětlovali kodérovi. V XSL je opět řešeno vlastní šablonou, která se aplikuje na stejná data obsažená ve zdrojovém XML souboru a to opakovaně tolikrát, kolikrát jsou ve zdroji obsažena. Tento přístup je opět mnohem více srozumitelný a vychází ze samotné definice XSL, které popisuje jak transformovat data, nikoliv jak data integrovat do obsahu stránek.

Šablona je tak, díky uváděnému způsobu zpracování, intuitivně dělena na menší části, popisující způsob zobrazení jednotlivých oblastí uživatelského rozhraní, tedy jednotlivé typy dat. Dílčí části je možné držet buď v jednom strukturovaném souboru, nebo je možné jej **rozdělit na menší části**, uložené v samostatných souborech (viz. příklad 2.5.13) a z nich pak složit hlavní šablonu pro celou webovou aplikaci, případně její části. Kodér tak má možnost udržet jednotlivé typy stránek v přehledné formě a v případě změny některého elementu tak provést úpravu pouze na jednom místě. Protože již v definici každé dílčí šablony se uvádí, **jaká data má zpracovávat**, je i při pohledu na zdrojový kód možné jednoduše identifikovat, která část šablony zobrazuje jaký typ dat.

Vzhledem k tomu, že generování obsahu stránky je závislé na vstupních datech je možné přímo prostřednictvím vložení, či nevložení určité informace do zdrojového XML ovládat, jestli se daný element na stránce zobrazí či nikoliv, což je důležité jestliže je požadavkem metodiky úplné **oddělení programového zpracování** od způsobu zobrazení.

Na první pohled se může zdát, že je tedy na základě XSL šablon možné vytvořit pouze jeden typ zobrazení výsledku pro jedna data (tj. např. produkty mohou být zobrazeny pouze

Příklad 2.5.13 Skládání XSL šablony z dílčích částí.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output encoding="windows-1250" method="html"></xsl:output>

<xsl:import href="katalog.xsl"/>
<xsl:import href="clanky.xsl"/>
<xsl:import href="porovnani-produktu.xsl"/>

<xsl:template match="/">
   <html xmlns="http://www.w3.org/1999/xhtml">
      <head>
         <title>ABC internetový obchod</title>
      </head>
      <body>
   ...
   </html>
</xsl:template>

</xsl:stylesheet>
```

jako jednotně definovaná tabulka na různých místech webu). Ani v tomto ohledu ale není XSL nevyhovující, neboť umožňuje definovat různá **zobrazení stejných dat v různých módech**.

2.5.3.3 XSLT procesor

Aby bylo možné provést na základě zapsaných pravidel transformaci z XML do cílového formátu je k tomuto účelu nutné použít softwarový prostředek, tedy *XSLT procesor*. K dispozici je dnes celá řada těchto nástrojů. Mezi nejznámější procesory, které jsou šířeny včetně zdrojových kódů, patří:

- *Saxon* – jedná se o volně dostupný procesor, napsaný v Javě, což umožňuje běh na libovolné platformě s interpretem Javy (`http://saxon.sourceforge.net/`)
- *libxslt/xsltproc* – procesor napsaný v C je opět volně ke stažení jako součást libxml2 z původního projektu Gnome. Je hodnocen jako jeden z nejrychlejších procesorů, striktně respektující specifikaci (`http://xmlsoft.org/XSLT/`).
- *Xalan* – je dostupný jakou součást projektu Apache XML. Má dvě verze pro Javu a pro C. Nejčastěji je Xalan používán současně s parserem Xerces XML, který je taktéž součástí projektu Apache XML (`http://xml.apache.org/xalan-j/index.html`).
- *XT* Javový procesor, který patřil mezi jeden z prvních, který zcela úplně implementoval XSLT a XPath. Vývoj je již zastaven, šlo pouze o ověřovací implementaci standardu (`http://www.blnz.com/xt/index.html`).

- *MSXML* komerční procesor od Microsoftu, který je hojně používán v různých projektech (Internet Explorer, SQL Server, ...), zejména pro svoji rychlost. Jeho určitou nevýhodou je omezení pouze na platformu Windows (`http://msdn.microsoft.com/xml/`).
- *.NET System.Xml* – novější implementace od Microsoftu, která je součástí vývojového frameworku .NET. V rámci něho je možné jej dále rozšiřovat, omezení jsou stejná jako u ostatních produktů Microsoftu (`http://msdn2.microsoft.com/en-us/library/system.xml(vs.71).aspx`).
- *Sablotron* – podporuje XSTL, DOM a XPath, je napsaný v C++, ale snahou vývojářů je jeho maximální portabilita, takže jde zkompilovat téměř na všech současných platformách (`http://www.gingerall.org/sablotron.html`).
- *4XSLT* je procesor napsaný v Pythonu (součást 4Suite), opět se snahou portability, je možné spustit jak na Windows systémech, tak na Unix-like systémech. Prozatím neexistuje ve stabilní verzi (`http://sourceforge.net/projects/foursuite/`).

Výše uvedený výčet neobsahuje zdaleka všechny existující XSLT procesory, další je možné najít jako součást systémů jako je Oracle, IBM, apod. Zpočátku byly největším problémem XSLT procesorů, pokud se používaly pro převod XML dokumentů do HTML v reálném čase, časté výkonnostní problémy. Postupně se však objevovaly techniky, které proces několikanásobně zrychlily. Ke zrychlení se využívá např. vygenerováním zdrojového kódu jednoúčelového transformačního programu. Běh tohoto programu je pak mnohem rychlejší, protože není třeba opakovaně načítat a vyhodnocovat styl.

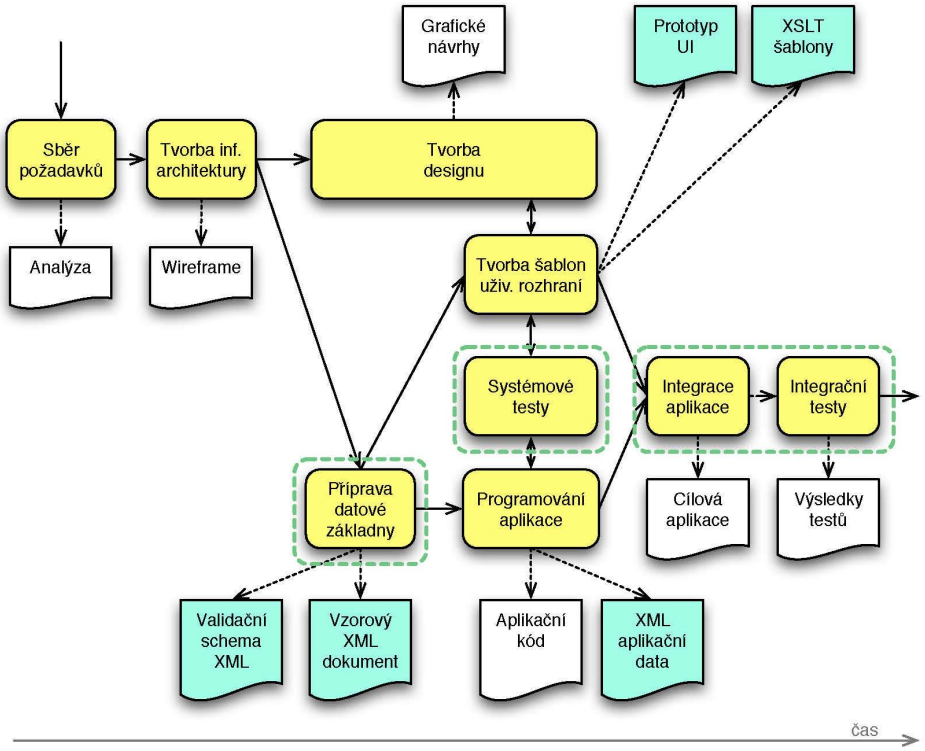
Většinu procesorů lze využít buď jako samostatné programy, nebo jako knihovnu vhodnou pro začlenění do vlastních programů. Při prvních pokusech je zřejmě nejlepší použití samostatných procesorů, aby si bylo možné prohlédnout výsledek transformace [24].

Pokud chceme použít XSLT pro převod XML dokumentů do formátů používaných na webu, může procesor běžet buď *přímo v prohlížeči*, nebo se konverze do HTML provede *na serveru*. I přes neustálý rozvoj internetových prohlížečů se dnes stále používá především druhý přístup. Podrobnější rozpracování obou přístupů je uvedeno v oddílu 2.6.5.

Z hlediska metodiky ETWA je volba procesoru čistě záležitostí implementace a záleží na konkrétním týmu jaký procesor využije ve svém projektu webové aplikace v závislosti na zvolené platformě a jazyku.

2.6 Aplikace metodiky

Obsahem tohoto oddílu je propojení všech popsaných částí do jednoho celku a navázání na konkrétní činnosti spojené s přípravou webové aplikace, realizované prostřednictvím jednotlivých projektových rolí. U každé fáze je definováno jakým způsobem na ni metodika ETWA pohlíží a jaké jsou požadavky metodiky na danou fázi. Zmíněny jsou také výhody a rizika použití metodiky pro daný případ.



Obrázek 2.8: Tvorba webové aplikace dle ETWA

2.6.1 Analýza řešení

Na začátku projektu je nutné získat podklady a požadavky od zadavatele, aby bylo možné projekt naplánovat jak po stránce časové, tak alokace zdrojů a požadovaných cílů projektu. V tomto ohledu se nebudeme konkrétně zabývat *vodopádovým* nebo *iterativním* způsobem vývoje (podrobněji popsaném v oddílu 1.10), neboť je možné následující postupy použít v obou případech. V případě vodopádového modelu by měly být výhody metodiky mnohem lépe pozorovatelné, neboť období tvorby aplikace jsou delší a problém s paralelní činností různých rolí je častější. Analýza vzniká na základě poznání požadavků zadavatele, podstaty daného požadavku a jeho integrování do návrhu cílového projektu. Sběr požadavků je nutné řídit a směřovat k integrovanému závěru a definici jednotlivých objektů v systému. Ve své nejobecnější části obsahuje analýza definice business objektů, které reprezentují objekty reálného světa, které zadavatel zná, a kterým v jeho zadání rozumí. Objekty jsou definovány jako součásti business procesů, které mapují jejich chování a jako celek spoluutváří chování celé cílové aplikace. U objektů je nutné identifikovat především všechny vlastnosti (atributy), které objekt má, a které budou v systému používány. Zpravidla je zadavatel schopen popsat všechny vlastnosti, které by chtěl mít v uživatelském rozhraní aplikace. Od definice business objektů se následně odvíjí struktura dat předávaných z aplikace do XML mezivrstvy; tedy do XML kolektoru. V této souvislosti je analýzou chápán detailní design aplikace, kde jsou již popsány veškeré detaily fungování celé aplikace. Jak je demonstrováno dále, je nutné se v tomto ohledu soustředit především na business objekty, jejich atributy a chování; ostatní součásti analýzy je možné díky ETWA zpřesňovat v dalších fázích. Výstupem této fáze je

dokument analýzy.

Požadavkem ETWA na tuto fázi je definice business objektů včetně vlastností a první pracovní verze vzorového XML dokumentu pro transportní mezivrstvu.

2.6.2 Informační architektura

Tato fáze je předpokladem pro zahájení fáze tvorby designu a jen omezeně je možné je navzájem prolínat. Informační architekturu zle chápat jako rozšířenou analýzu, zamčřenou již na konkrétní použití aplikace. Tvůrce informační architektury musí na základě získaných požadavků a zpracované analýzy vytvořit návrh uspořádání uživatelského rozhraní a to nikoliv s důrazem na estetičnost a grafiku, nýbrž z hlediska použitelnosti a pravidel chování uživatelů. Cílem této aktivity je rozmístění ovládacích prvků, a důležitých informací na místa, kde je uživatel skutečně spatří, bude je zde očekávat a najde je v souvislosti s určitým typem úlohy, kterou prostřednictvím aplikace provádí. Kromě vlastního nalezení určeného prvku uživatelem, je důležitým faktorem také schopnost uživatelů ovládat dané rozhraní bez nutnosti studovat manuály a učit se jak zacházet s neobvyklým uživatelským rozhraním. Práce informačního architekta je tedy částečně ovlivněna sociálním výzkumem, tedy znalostí chování uživatelů, určením cílové skupiny uživatelů, analýzy podobných aplikací a důvodů, proč je daná aplikace úspěšnější a lépe použitelná než jiná. Na toto nestačí pouze znalost internetových technologií. Na informačního architekta tedy přechází zodpovědnost návrh prototypových obrazovek po stránce logického uspořádání. Výstupem jeho práce je *wireframe* (viz obrázek 1.9) každé obrazovky, doplněný funkčním popisem, pokud je to nutné. Tvorba wireframů není nikterak formalizována, neexistuje pro ni žádný oficiální schématický jazyk a je tedy pouze na architektovi, jak schéma připraví. Jediné omezení existuje v používaných značkách formulářových elementů (např. textové pole, tlačítko, checkbox, dropdownlist, ...), které je dáno možnostmi vykreslovacího jádra internetového prohlížeče.

Z pohledu metodiky ETWA nejsou za tuto fázi požadovány žádné změny nebo nestandardní výstupy oproti běžně používaným postupům. Metodika pouze doporučuje tuto fázi vždy do projektu zahrnout, aby na základě jejich výstupů bylo možné jednoznačně a rychle zadat výrobu designu, a to i formou outsourcingu. A dále kvůli eliminaci rizik souvisejících se změnou zadání nebo struktury aplikace, které je daleko náročnější realizovat do vytvořeného designu než na úrovni informační architektury. Po dokončení návrhu informační architektury navazuje na nejen vlastní *fáze designu*, ale paralelně také *fáze přípravy datové základny*, tj. definice obsahu XML mezivrstvy.

2.6.3 Tvorba designu

Cílem této fáze je transformace wireframů, vytvořených informačním architektem, do podoby konečného vzhledu webové stránky. Výstupem ale není přímo webová stránka, ale pouze statický obraz, který je předán kodérovi k následnému zpracování ve fázi tvorby uživatelského

rozhraní. Grafik má na starost estetickou stránku webové aplikace. Jestliže se jedná o aplikaci, která bude sloužit zároveň jako marketingový nástroj (prodej zboží, služeb), je tato fáze jedním z důležitých faktorů komerční úspěšnosti celého řešení, a proto bude na tuto fázi kladen poměrně velký důraz. Zadáním pro grafika jsou vytvořené wireframy a jako doplňkový informační zdroj může být použit dokument analýzy, zejména pak funkční specifikace. Při tvorbě designu je vhodné vycházet i z ostatních doporučení informační architektury, než je jen použití konkrétných prvků na zvolených pozicích. Důležitá je i volba barev, barevných přechodů, zvýrazňujících symbolů, apod. Toto je právě přínosem grafika do projektu.

Vzhledem ke specifikům internetové tvorby, musí být také grafik seznámen s omezeními webu a design musí navrhovat s ohledem na cílové použití. Mezi specifika z hlediska designu patří zejména množství použití rastrových grafických prvků (obrázků) a jejich omezená velikost, ve výsledku určující rychlost načítání stránky uživatelského rozhraní; variabilní šířka a výška okna a s ní se přizpůsobující obsah/design okna; omezená rodina fontů dostupná na různých platformách, umožňuje se spolehnout, že vzhled bude víceméně stejný; omezená paleta barev je dána paletou tzv. web-safe colors, u kterých lze garantovat vzájemnou slučitelnost mezi použitím na rastrovém obrázku a uvnitř webové stránky. Bylo by možné definovat více omezení, nicméně uvedená představují nejčastější problémy. Výstupem této fáze je *grafický návrh* všech unikátních stránek v podobě obrázku.

Z pohledu metodiky ETWA nejsou za tuto fázi požadovány žádné změny nebo nestandardní výstupy oproti běžně používaným postupům – pouze důsledné respektování informační architektury, respektive analýzy.

2.6.4 Příprava datové základny

Jedná se o novou mezifázi samostatně oddělenou v rámci návrhu metodiky ETWA. V případě standardního přístupu k tvorbě aplikací by se pod podobně pojmenovanou projektovou fází skrýval databázový návrh, pravděpodobně ve formě E-R diagramu[11], případně class-modelu. Tato část by neměla být z projektu v žádném případě vypuštěna, ale její základ by měl být připraven již ve fázi analýzy. Z vytvořených modelů můžeme vycházet pro zjednodušení a zrychlení této fáze. Narozdíl od E-R diagramu je zaměřena na business objekty a datové objekty relevantní k uživatelskému rozhraní. Účastní se jí standardně nominovaní členové projektového týmu: analytik, programátor a kodér. Ačkoliv by tuto fázi bylo možné zařadit přímo do fáze analýzy, řadí ji přesto ETWA až za dokončenou fázi informační architektury. Důvodem je právě zohlednění výstupů informační architektury, kde jsou poprvé použity navržené datové konstrukce. Navíc definice vzhledu uživatelského rozhraní s sebou nese potřebu získávání dalších dat, která nemusela být v analýze identifikována a jedná se tedy primárně o data důležitá v kontextu uživatelského rozhraní.

Ačkoliv jde stále ještě o přípravnou fázi, jsou do ní přizváni kodér s programátorem, pro které je důležitý zejména její výstup. Na základě dohodnuté struktury a obsahu budou následně aplikací generována data do XML mezivrstvy a šablonou zobrazována do podoby

[11] E-R diagram - Entity Relation Diagram mapuje vazby/relace mezi jednotlivými databázovými entitami. Jeho prostřednictvím se modelují především databázové modely.

finálního uživatelského rozhraní. Při přípravě vzorového datového výstupu je nutné respektovat adaptovaná pravidla pro základní strukturu XML dokumentu, tak jak byla popsána v oddílu 2.5.1.1.

Výstup této fáze je vytvářen jako *vzorový XML dokument*, který obsahuje veškeré objekty (elementy), které se mohou počas provozu v aplikaci zobrazovat. Dokument je postupně zpřesňován a doplňován na základě vstupů z analytické fáze, nebo připomínek nominovaných účastníků. Vytvořený XML dokument by měl být dostupný všem členům týmu po celou dobu vývoje. Jestliže se v aplikaci něco změní, nebo se změní struktura nějakého business objektu, je nutné tyto změny zanést do vytvořeného XML dokumentu.

Pro každou typizovanou obrazovku uživatelského rozhraní existuje vlastní wireframe, lze při vytváření vzorového XML dokumentu postupovat následujícími kroky. V každém wireframe označit statické prvky, generovaná data a z jakého zdroje pochází. Pro každý typ takto označených údajů je vytvořen samostatný element, který je v XML dokumentu jednoznačně identifikován a dále jsou pojmenovány jednotlivé jeho vlastnosti (atributy nebo potomci). Díky účasti programátora i kodéra je možné identifikovat také data, která nejsou na první pohled z vizualizace uživatelského rozhraní identifikovatelná, ale jedna ze stran ví, že je pro vytvoření uživatelského rozhraní bude potřebovat (např. adresa, která bude vkládána do odkazu, nebo velikost obrázku v pixelech).

Na základě XML dokumentu je způsobem popsaným např. na příkladu 2.5.10 vytvořeno první validační schéma pro komunikaci skrze nově vytvářenou mezivrstvu. Vygenerované schéma je třeba ještě upravit, především s ohledem povinné či volitelné zahrnutí některých vlastností, je nutné zkontrolovat, případně doplnit datové typy u jednotlivých hodnot. Druhým výstupem této fáze je tedy *validační schéma* ve formátu XML. Dále je třeba zmínit, že zavedení této fáze není kritické vůči délce trvání projektu, neboť probíhá paralelně s fází tvorby designu a je ve většině případů výrazně kratší, tudíž neleží na kritické cestě projektu. Velmi pozitivně může být využito vzájemné iterace těchto fází.

V tomto případě se jedná o nově přidávanou část metodiky ETWA, která je základem přípravy XML mezivrstvy. Požadované výstupy, kterými jsou vzorový XML dokument a na základě něho vytvořené validační schéma budou využity i v dalších fázích vývoje aplikace.

2.6.5 Tvorba šablon uživatelského rozhraní

Start této fáze je plně závislý na plném, nebo částečném dokončení a odsouhlasení designu prototypovaných obrazovek. Druhým předpokladem je dostupnost role kodéra pro tuto aktivitu. Jedním z požadavků metodiky ETWA je oddělit závislost rolí kodéra a programátora a umožnit flexibilní alokaci zdrojů v projektu. Právě v této fázi začíná skutečné využití předností metodiky ETWA. Kodér nemusí zahájit svoji práci bezprostředně po dokončení designu a na druhou stranu také nemusí čekat na dokončení kompletní sady grafických výstupů. Jeho alokace je flexibilní. Díky předdefinovanému, v předchozí fázi vytvořenému, XML dokumentu může začít svoji práci nezávisle na programátorovi. Vytvořené šablony nejsou pouze jakýmsi meziproduktem, nýbrž konečným výstupem, který se pak pouze napojí na

Příklad 2.6.1 Vložení odkazu na XSL šablonu postačí k transformaci XML dokumentu přímo v prohlížeči.

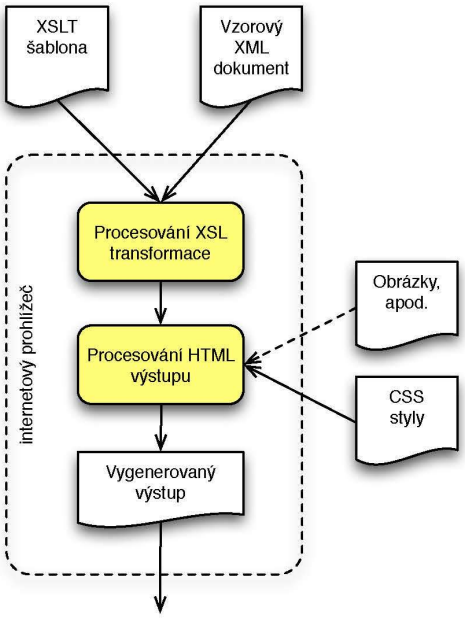
```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet href="sablona-seznam-produktu.xsl" type="text/xsl"?>

<etwa>
    <general>
        <task>products.list</task>
        <output>products.list.template</output>
        <parameters>
            <parameter>
               <name>manufacter</name>
               <value>Apple</value>
            </parameter>
        </parameters>
    </general>
    ...
</etwa>
```

aplikační kód. Tím, že v této fázi vznikají již skutečné součásti aplikačního rozhraní, šetří se čas programátora (nebo zručného kodéra), který tuto integraci ve většině případů musel provádět. Existují tak od počátku pouze jedny šablony, které má na starosti konkrétní člen projektového týmu. Mimo nezávislosti na fázi programování je také variabilní čas začátku fáze. U většiny internetových projektů tvoří čas, alokovaný na tvorbu šablon, cca 20-40 % času programování, a proto je možné aktivity programování a tvorbu šablon začít ve stejný čas, či krajně skončit ve stejný čas (vazby Start-To-Start nebo Finish-To-Finish).

Práce kodéra se v tomto případě omezuje pouze na používání HTML a stylopisů (stylesheets), které umožňují HTML výstup formátovat pro potřeby uživatelského rozhraní ve smyslu navrženého designu. Jako stylopis se využívá CSS (Cascading Style Sheets), který vychází z pravidel HTML, ale nejedná se o SGML jazyk. Na druhou stranu stylopis XSLT je SGML jazykem a veškeré zápisy v něm odpovídají pravidlům zápisu XML dokumentů. Každá šablona je pak celá jednou XSLT šablonou, která transformuje data ze vzorového XML dokumentu do jednotlivých stránek odpovídajících jednotlivým navrženým designům, resp. wireframe.

V této fázi metodika ETWA vyžaduje od kodéra znalost jazyka/stylopisu XSLT. Tento jazyk není kodér ve většině případů zvyklý používat a kromě osvojení si formálního způsobu zápisu a sady používaných elementů je nutné také začít pohlížet na šablonu z hlediska dat a způsobů jejich reprezentace v uživatelském rozhraní. Čas, který bude nutné vynaložit na zaškolení člověka do XSLT bude kompenzována rozšířením znalostí dané osoby a je potřeba od něho odečíst čas, který by daná osoba strávila studiem a zaškolením do jiného šablonovacího systému, který pro obdobný způsob integrace dat do šablony využívá svého vlastního jazyka/zápisu. Znalosti člověka vyškoleného v používání XSLT je pak možné využít nejen v oblasti tvorby šablon, ale také v oblastech datové integrace, webových služeb, aplikačních interfaců (API), apod. Co se týká náboru nových zaměstnanců přestavuje XSLT standard,



Obrázek 2.9: Transformace v prohlížeči pro účely prototypu

který není nutné každému nově příchozímu zaměstnanci vysvětlovat, ba ani není neobvyklé od nově nastupujících zaměstnanců znalost XSLT vyžadovat.

Při finální integraci šablon s aplikací je možné velmi jednoduše odhalit případné nekonzistence v datech. Datový výstup z aplikace (XML data z XML kolektoru) je možné si před transformací zobrazit přímo v internetovém prohlížeči a najít tak případné rozdíly mezi dohodnutým vzorovým XML dokumentem a skutečným výstupem aplikace. Díky striktnímu oddělení zodpovědností je i oprava nalezené chyby jasně adresovatelná.

Data, která nejsou do šablony zaslána se ve výsledné stránce vůbec nezobrazí, neboť není spuštěno renderování příslušné části šablony z důvodu chybějících dat. Naopak invalidní data, která by byla do šablony poslána při běžném zpracování, jsou odfiltrována již na úrovni validace dat vůči např. Relax NG schématu a v šabloně, resp. uživatelském rozhraní, se taktéž nezobrazí. Filtrování dat je prováděno až ve fázi integrace s aplikačním kódem, neboť do té doby jsou šablony vytvářeny na základě validního, vzorového dokumentu.

Prací kodéra není pouze vytváření graficky příjemného uživatelského rozhraní, ale také jeho přístupnost a použitelnost. Za tímto účelem je připravován HTML kód, který je navržen např. dle pravidel SEO, pravidel o přístupnosti webu, apod. Tato pravidla kodér zná a je tedy i v případě naplnění daty odpovědný za udržení konzistence v i této oblasti.

Nespornou výhodou šablony v XSL je možnost vytvořit výstup téměř v libovolném formátu. Pouhou výměnou šablony je tak možné stejná data zobrazit v jiném formátu (např. RTF, WML, ...).

2.6.5.1 Prototypování

Jak již bylo popsáno v oddílu 1.1.3, jakmile zadavatel poprvé uvidí a má možnost si aplikaci vyzkoušet, vzniká velké množství připomínek a v mnohých případech se odhalí různé pohledy

na stejnou věc, které vyústí až v soupis nutných změn. Tyto změny na straně projektového týmu výrazně navyšují pracnost a proto jsou přijímány s velkou nevolí. Požadovaná změna se může zdát jako jednoduchá úprava na úrovni uživatelského rozhraní, ale nese s sebou výrazné změny na úrovni aplikačního kódu.

Šablony jsou generovány na základě XML vzorového souboru, který je, co do struktury, identický s výstupem z aplikace a je možné v nich nasimulovat chování celé aplikace na bázi statického XML dokumentu, který je vytvořen ve fázi přípravy datové základny. Toto je umožněno díky standardní vlastnosti internetového prohlížeče, který umožňuje transformovat nejen samotné XML dokumenty, ale obsahuje i XSLT procesor, díky kterému je možné XSLT transformace provádět přímo v prostředí prohlížeče, bez nutnosti jakékoliv serverové části, nebo zvláštního aplikačního kódu.

V internetovém prohlížeči tak kodérovi vznikají skutečné stránky, přidáním nebo odebráním elementů ze zdrojového XML souboru mohou být simulovány různé stavy aplikace. Prostřednictvím odkazů obsažených uvnitř XML dokumentu je možné šablony navzájem propojit a vytvořit tím takzvaný *prototyp*. Ten samozřejmě neobsahuje aplikační kód a proto slouží pouze pro představu jak budou prvky uživatelského rozhraní vypadat, jak budou na stránce formátovány, co bude a co nebude vidět, atd.. Prototyp slouží právě pro interakci se zadavatelem, který tak má možnost vznést připomínky k ještě nekompletní aplikaci a tyto změny mnohem jednodušeji zapracovat do konceptu uživatelského rozhraní. Prototyp zároveň působí na zadavatele jako „hmatatelný“ výstup projektu v raném období vývoje a je jím vždy dobře přijat.

Metodika ETWA v této fázi navrhuje úplně jiný způsob realizace šablon. Předpokladem je existující XML vzorový dokument a výstupem je XSL šablona obsahující deklarace HTML a CSS. Pozitivně hodnoceným mezivýstupem je prototyp webové stránky, nebo možnost generování výstupů do jiného formátu než je pouze HTML. Prototypování přitom nevyžaduje zvláštní, pro jiné účely nevyužitelnou, sadu šablon. Ty jsou tvořeny zásadně v SGML odvozených jazycích, což podporuje specializaci projektových rolí.

2.6.6 Programování aplikace

Z programátorů byla odebrána přímá zodpovědnost za uživatelské rozhraní, což představuje hlavní přínos metodiky ETWA oproti standardním postupům. Aplikace pouze generuje data, která předává do XML kolektoru. Výraznou změnou (nikoliv oproti MVC) je nezávislost postupu zpracování a uložení dat do XML kolektoru. Není tudíž důležité, jestli jsou do XML kolektoru nejprve vložena data o objednávkách a až následně o zákaznících. Data jsou vždy identifikována svým označením, které se promítá i do následně vygenerované XML struktury. Ani v XML není nutné držet stejnou posloupnost dat, protože z hlediska XPath dotazů, používaných při XSLT transformaci není důležité jestli je prvek první nebo poslední v řadě.

Vytvořená data jsou vkládána do XML kolektoru. Způsob jeho realizace metodika ETWA striktně nedefinuje, musí pouze umožňovat vkládat jednotlivé datové elementy do rámcové struktury a umět z ní vygenerovat validní XML dokument podle dohodnutého vzoru. Vo-

litelně metodika doporučuje provádět validaci dat, která je za použití současných nástrojů velice jednoduše implementovatelná a omezuje tak riziko nesprávné interpretace XML dat při XSLT transformaci.

Díky oddělení uživatelského rozhraní a aplikace, umožňuje programátorům se lépe koncentrovat na vlastní vývoj, což dává větší prostor pro jeho zkvalitnění a použití pokročilejších metod programování. Dobře navržený controller, nebo řídící skript, umožní fungování aplikace pouze na bázi volání metod objektů s parametry získanými prostřednictvím URL, tedy zadání uživatele.

Přínosem tohoto návrhu je je striktní oddělení role programátora od role kodéra. Společně musí pracovat pouze na přípravě XML dokumentu který je základem transportní vrstvy. Dále ve fázi vývoje může programátor pracovat zcela nezávisle na kodérovi. Navíc umožňuje programátorovi začít pracovat na svém úkolu mnohem dříve, než je tomu u klasických přístupů. Programátor nemusí čekat na dokončení fáze tvorby designu a šablon pro uživatelské rozhraní. Je možné navázat okamžitě na fázi přípravy datové základny, kdy je definováno, jaké výstupy má aplikace produkovat. Tímto se zrychlí nasazení programátorů, což v důsledku představuje zkrácení doby potřebné pro vývoj produktu. Změny, které by bylo nutné v aplikaci provést, je možné realizovat i jako samostatný balíček, který přidá do XML kolektoru další element a zobrazení obsahu tohoto elementu do stránky poté zajistí šablona.

Díky XML kolektoru je možné skládat data v různých jazycích a využít rozšířených vlastností kolektoru, který může integrovat nejen aplikační výstupy, ale zároveň i samostatné XML dokumenty, uložené na serveru. Samostatné dokumenty představují statická, jazykově odlišná data, která není nutné v aplikaci při každém požadavku znovu a znovu generovat. Této rozšířené výhody XML kolektoru je možné využít i k jiným účelům, jako je např. konfigurace nebo personalizace.

Metodika ETWA eliminuje vazbu HTML rozhraní na aplikačním kódu. Veškeré výstupy jsou pouze v „čistém textu“, mají svoje jméno a patří do konkrétní, pojmenované sady dat. V aplikačním kódu se nevyskytují prvky uživatelského rozhraní, kód se skládá jen z elementů jednoho zvoleného jazyka. Dle doporučení metodiky je třeba implementovat XML kolektor, který zajišťuje sběr veškerých výstupů aplikace do jednotného úložiště. To je následně transformováno (např. serializací) do předem definovaného XML formátu, který je validován vůči dohodnutému schématu. Výstupem fáze programování je generovaný, validní XML dokument s veškerými daty aplikace, získanými v rámci daného zpracování.

2.6.7 Systémové testy

Hlavním úkolem systémových testů, prováděných v průběhu vývoje aplikační části a přípravy šablon, je ověřovat, že výstupy obou částí jsou systémově správné, odpovídají definovaným pravidlům. Při použití metodiky ETWA je možné provádět systémové testy velice efektivně, což u standardních přístupů není příliš možné. U šablon je možné systémové testy provádět na základě úpravy vzorového XML dokumentu a na jeho základě porovnávat skutečný a předpokládaný výsledek. Pomocí simulace různých extrémních stavů (např. příliš dlouhého

textu, velkého množství řádků v tabulce, apod.) je možné odhalit již v okamžiku provádění systémových testů nekonzistence, které by se projevily např. až v okamžiku spuštění.

Systémové testy aplikační části je možné provádět generováním různých vstupních URL a kontrolou následných výstupů v generovaném XML. Je tak možné ověřit různé způsoby použití aplikace, včetně extrémních případů na úrovni vstupních parametrů. Systémovými testy by se měl validovat také vygenerovaný XML dokument vůči schématu. Tj. jestli dopovídá definované struktuře, případně i konkrétním datovým typům. Jestliže není ještě v rámci aplikace implementován validátor vůči např. Relax NG schématu, je možné postupovat způsobem vygenerování XML výstupu a jeho validaci prostřednictvím některého z volně stažitelných validátorů. Systémové testy by měly ještě před integrací aplikace prověřit jak budou výsledné části schopny spolu komunikovat.

Systémové testy se u webových aplikací často provádí pouze na straně programátora, který si různými způsoby zjišťuje, jestli je výstup jeho programu správný. Metodika ETWA tuto část vyčleňuje a umožňuje už v rámci těchto testů nekonzistence odhalit. Výhodou je možnost testování nejen aplikačního kódu, ale také šablon. Výstupem systémových testů by měl být *protokol o chybách*, který slouží jako vstup do probíhající vývojové fáze, za účelem odstranění zjištěných chyb ještě v průběhu této fáze.

2.6.8 Integrace aplikace

Aplikace jako celek začne fungovat v okamžiku, kdy je prostřednictvím transportní XML vrstvy spojena aplikační vrstva se šablonou. Z hlediska řízení projektu je tedy nutným předpokladem pro zahájení integrace dokončení fází přípravy šablon a programování aplikace. Integrační fáze se účastní jak kodér, tak programátor. Smysl vyčlenění této fáze je zejména v alokaci zdrojů ve stejném čase. Aplikaci na šablonu je tak možné napojovat až v okamžiku dokončení předcházejících částí.

V modelu MVC se většinou o integraci stará prvek *C*-Controller, který zajišťuje jednak transparentní mapování vstupních adres a parametrů, ale zároveň obsluhuje transport dat do vrstvy *V*-View a určuje také, které view (šablona) bude pro zobrazení dat použita.

Ve fázi integrace se mapují konkrétní šablony (typy stránek) na akce generované aplikačním kódem. Výstupem integrované aplikace je již korektní XHTML formát, který je odeslán uživateli. Současné prohlížeče podporují transformace XML dle XSLT šablon, proto by bylo vhodně provést akci zpracování až na straně klienta. Návrh metodiky ETWA tento přístup umožňuje a je standardně využíván během vývojové fáze pro účely tvorby šablon bez napojení na aplikaci na pozadí. Posun ke zpracování na straně klienta je možné v budoucnu očekávat, nicméně v současné době není možné se na takovýto způsob transformace (obzvláště u komerčních aplikací) zcela spolehnout. Proto je transformace do cílového XHTML formátu provedena na straně serveru a je k ní využito některého z procesorů uvedených v oddílu 2.5.3.3. Cílem transformace nemusí být jen formát XTHML, ale i libovolný alternativní formát (CSV, XML, PDF, ...).

V průběhu integrace dohází k definici přemčny zdrojových XML+XSLT dat do cílového formátu, kterým je např. HTML+CSS. Transformace je prováděna na serveru, což může zvyšovat jeho zátčž, na druhou stranu není myslitelné, že by u aplikací, zamčřených na širokou veřejnost, bylo možné odesílat data prostřednictvím čistého XML+XSLT. V průběhu integrace je vytvořen nebo nakonfigurován controller aplikace, který dokáže spojit šalonu s voláním příslušné metody. Integrační fáze je zvlástní krok zavedený metodikou ETWA. Hlavní část integrace je především na straně aplikačního kódu, nicméně zde dochází k ověření funkčnosti aplikační a prezentační vrstvy. Transportní vrstva a data v ní jsou, dle definice metodiky, otevřená, a proto je spojování výstupů aplikace s jejich uživatelskou prezentací jednoduché a velmi názorné.

2.6.9 Integrační testy

Integrační testy jsou zpravidla jedinými testy, které jsou nad webovými aplikacemi prováděny, pokud jsou vůbec prováděny. Mají za cíl ověřit, jestli aplikace funguje jako celek, tj. jestli všechna rozhraní, prostřednictvím níž aplikace komunikuje s ostatními objekty, fungují správně, jestli aplikace počítá správně, jestli se správně zobrazuje uživatelské rozhraní, ... Právě v oblasti integračního testování přispívá metodika ETWA ke zjednodušení a zrychlení.

Vlastní propojení aplikace s uživatelským rozhraním je kontrolováno již na úrovni XML validace oproti definovanému schématu. Druhým stupněm automatizované validace je transformace prostřednictvím XSL šablony.

Jestliže aplikace vykazuje ve fungování určité nekonzistence, jako je například nekorektní formátování datumu, chyby ve stránkování, chybějící, či proházené údaje je potřeba ověřit, jestli chyba vzniká již na úrovni programového kódu, nebo jestli se jedná o chybu v uživatelském rozhraní. Díky mezivrstvě ve formátu XML je možné pozastavit zpracování aplikace před transformací prostřednictvím XSLT a vrátit tak do prohlížeče přímo XML výstup aplikace. Protože prohlížeče disponují XML parserem, je výstup strukturovaně a také barevně zobrazen. V XML datech je možné nalézt konkrétní problematický údaj a ověřit, jestli je na úrovni XML dat v pořádku. Jestliže je nekonzistence nalezena také v XML datech, jedná se o chybu na straně aplikačního kódu a je na programátorovi tuto chybu odstranit. Pokud jsou naopak XML data v pořádku, jedná se o chybu v šablonách a je tedy na kodérovi, aby tuto chybu odstranil.

Odstraňování chyb je díky nezávislosti rolí, definované v rámci metodiky ETWA, možné provádět nezávisle v čase, podle toho kdy je daný zdroj dostupný. Testeři mají tímto způsobem usnadněnu práci, neboť ověří, že např. venkovní rozhraní aplikace (API) funguje korektně, neboť se jeho prostřednictvím do aplikace předávají správná data, která ve skutečnosti nemusí být v uživatelském rozhraní zobrazena vůbec. Zjištěný fakt, že uživatel ale požadovaná data nevidí, nebo vidí data jiná, je tak možné evidovat jako samostatnou chybu, ke které je tak automaticky přiřazen zodpovědný řešitel v tomto případě kodér. Stejně tak by fungoval i případ opačný.

Při integračních testech je možné, díky návrhu dle ETWA, testovat odděleně aplikační a prezentační vrstvu. Tester může libovolně zapínat nebo vypínat transformaci do HTML a sledovat tedy chování aplikace buď jako celku nebo na úrovni dat poskytnutých uživatelskému rozhraní. V této fázi se mění pouze přístup k testování, nikoliv vlastní proces. Ten metodika zefktivňuje a umožňuje flexibilněji realizovat opravy zjištěných chyb.

2.7 Vedlejší přínosy metodiky

Při implementaci a pilotním provozu dle definice metodiky ETWA byly identifikovány také další přínosy, které na počátku nebyly definovány jako požadavek, nicméně velice vhodně doplňují vlastnosti celého konceptu a zvyšují tak hodnotu celkového přínosu. Jednotlivé případy nebudou rozebrány dopodrobna, jsou zde uvedeny pouze jako doplňující prvek.

2.7.1 Připraveno pro AJAX

AJAX představuje zkratku z anglického výrazu *Asynchronous JavaScript and XML*. Tato technika je využívána velice často v souvislosti s Web2.0. Podstatou je aktualizace obsahu určité části stránky bez nutnosti přenačítání celého jejího obsahu. JavaScriptová funkce vytvoří na pozadí spojení se serverem, předá mu formou URL vstupní parametry a server vrátí zpět XML data a ta prohlížeč (resp. JavaScript) zobrazí do konkrétních částí stránky. Tímto způsobem lze ukládat data na pozadí, vyhledávat na pozadí, apod..

Díky tomu, že metodika ETWA již od své podstaty definuje jako výstup aplikačního kódu XML dokument, představuje integrace s AJAXem minimální množství práce. Běh aplikace se ukončí před serverovou transformací do výsledného HTML formátu a data se místo do XSL šablony předají v netransformované podobě prohlížeči. Výhodou je jeden, stejný, aplikační kód, který slouží jako ajaxový engine a stejně tak může zpracovávat požadavky na vygenerování celé stránky.

V tomto ohledu je třeba vést vždy v evidenci, jaká data a v jaké formě jsou v XML obsažena. Nebylo by příliš vhodné, aby prostřednictvím XML dokumentu pro AJAX bylo zasíláno heslo v nešifrované podobě, seznam uživatelů a ostatní nechráněné informace. V případě, že jsou do XML dokumentu (XML mezivrstvy) takovéto informace vkládány, doporučuje se spustit XSL transformaci s vlastní šablonou, která zredukuje zdrojové XML na formu, kterou je možné posílat prostřednictvím AJAXového volání (viz oddíl 2.7.2), stejně tak jako v případě zvláštních požadavků na strukturu XML.

2.7.2 Transformace do jiných XML formátů

Jestliže aplikační data již jednou existují ve formátu XML, je možné je poskytovat dalším systémům na bázi automatizované komunikace (webové služby, EDI, ...) nebo syndikace obsahu (RSS, Atom, ...). Jistým omezením je rozdílná struktura XML dokumentu, který jednotlivé systémy požadují. Protože jsou ale data v XML dokumentu identifikovatelná

a protože XSL šablona umožňuje transformaci zdrojových XML dat i do jiných formátů než je HTML, je možné nad aplikačním XML dokumentem připravit zvláštní XSL šablonu. Ta data, která se standardně používají např. pro výpis aktualit na homepage, transformuje do formátu cílového systému – v tomto případě formátu XML/RSS. Aby aplikace poskytovala nějaké své údaje např. prostřednictvím formátu XML/Atom, nevyžaduje toto jakýkoliv zásah programátora a tento úkol může vyřešit kodér zcela samostatně.

2.7.3 Rozložení zátěže

V případě, že je webová aplikace provozována na systémech s velkým zatížením, nebo výkonově slabších serverech a za předpokladu, že se jedná o uzavřenou síť, kde jsou definována koncová zařízení (tedy např. internetové prohlížeče určitého výrobce a verze), je možné rozložit zátěž mezi klienta a server. Server v tomto případě bude pouze generovat data v XML formátu a ta bude s odkazem na příslušnou XSL šablonu odesílat do webového prohlížeče. Transformaci dat a tím i veškeré vytvoření uživatelského rozhraní zajistí webový prohlížeč. Uživatel přitom nezpozoruje žádný rozdíl mezi oběma přístupy. Ve spojení s AJAXem je pak možné vyměňovat pouze části XML za jiné a díky transformaci na straně prohlížeče budou tato data okamžitě zformátována do vzhledu uživatelského rozhraní.

Čistě teoreticky by bylo možné nechat server přegenerovat kompletně celé XML pro danou obrazovku webového rozhraní, tu pak AJAXem poslat do stránky a zde vyměnit pomocí JavaScriptu původní XML za nové, čímž lze dosáhnout poměrně rozsáhlé změny, bez nutnosti přegenerování stránky jako celku.

Kapitola 3

Případové studie

„Uč se chtít, ne aby se věci přizpůsobovaly tvým přáním,
nýbrž aby se tvá přání přizpůsobovala věcem.“
Marcus Aurelius

3.1 Záměr případových studií

Jak již bylo demonstrováno na několika dílčích příkladech návrhu metodiky, nejedná se pouze o teoretický návrh, ale je možné jej prakticky implementovat pomocí dostupných technologií a vybavení. Dílčí ověření jednotlivých kroků a postupů ovšem neověřuje praktickou použitelnost metodiky na celém vývojovém procesu webové aplikace. K tomuto účelu byly ve spolupráci s internetovou agenturou vybrány 2 reálné projekty obdobného rozsahu. Projekty jsou pracovně označeny jako Alfa a Beta a jejich cílem je vytvoření elektronického obchodu.

Projekt Alfa byl realizován standardním způsobem, dle postupů, které společnost pro své projekty standardně používá. Jedná se o platformu Linux, programovací jazyk PHP, databázový server mySQL, Model-View-Controller framework a vlastní redakční systém (CMS). Hlavním úkolem u tohoto projektu tedy nebyla změna metodiky práce, ale hlavně monitorování postupů, výskytu překážek uvedených v oddílu 2.1 této práce, sledování pracností a celkového průběhu projektu za účelem porovnání s projektem Beta.

V případě *projektu Beta* se jednalo již o ověření metodiky ETWA. Z pohledu projektu a společnosti se jednalo o krok s poměrně vysokým rizikem. Nejen že byl projekt vytvářen novým, ještě neověřeným způsobem, ale i možnosti budoucí údržby a správy takovéhoto projektu jsou omezené na znalost použité metodiky. Před zahájením projektu bylo nutné seznámit nominovaný tým s metodikou vývoje. Vlastní MVC framework bylo nutné upravit o zavedení nové XML transportní vrstvy a validaci. Zpracování view muselo být zcela nahrazeno. Kodér se musel detailněji seznámit s fungováním XSL šablon a konstrukcemi tohoto jazyka, což bylo díky množství dostupných zdrojů poměrně bezproblémové. Tyto činnosti bylo nutné provést ještě před započetím prací. K vlastní realizaci pak bylo využito stejných technologií a dílčích, neměněných postupu jako u projektu Alfa.

Autor u obou projektů působil v roli projektového manažera a proto prezentované infor-

mace nebyly poskytnuty zprostředkovaně, ale zaznamenané přímo v průběhu obou projektů.

Protože jak už z definice *projektu*[1] vyplývá jedná se o neopakovatelnou a jedinečnou činnost a proto ani výsledky dvou uváděných projektů není možné absolutně porovnávat. Cílem těchto případových studií je ověřit praktickou použitelnost metodiky ETWA, získat připomínky od projektových týmů a identifikovat tak její slabá místa. U konvenčně vyvíjeného projektu identifikovat okamžiky, kdy by metodika ETWA mohla fungovat lépe a najít tak příležitosti jejího uplatnění.

3.2 Projekt Alfa

3.2.1 Popis projektu

Projekt byl připravován pro společnost zabývající se prodejem a distribucí papíru, kancelářských potřeb a souvisejícího sortimentu, jehož rozmanitost se postupně rozšiřuje v souvislosti s vyššími potřebami zákazníků. Společnost působí mimo exponované metropole a proto vsadila na obchodní model partnerské spolupráce a dealerství. Zákazníky eviduje na 3 úrovních: *koncový uživatel* je návštěvníkem internetového obchodu, který přímo na své jméno nakupuje sortiment za koncové ceny. K nakupování mu stačí pouze registrace; *dealer* nakupuje zboží za zvýhodněné ceny, svoje ceny může poskytnout své skupině koncových uživatelů, přičemž z každého nákupu získává provizi ve formě bonusových bodů (budoucí absolutní sleva v Kč), k registraci je nutné ověření dopisem generovaným obchodem a minimálně 3 registrovaní koncový uživatelé; *zákazník* může nakupovat za stejné ceny jako dealer, nicméně jeho ceny (a ceny jeho dealerům) se snižují na základě výkonu všech jeho dealerů a navíc získává z nákupu každého dealera peněžní bonus, podmínkou registrace je ověření dopisem a minimálně 3 registrovaní dealeaři s určenou minimální útratou za poslední měsíc.

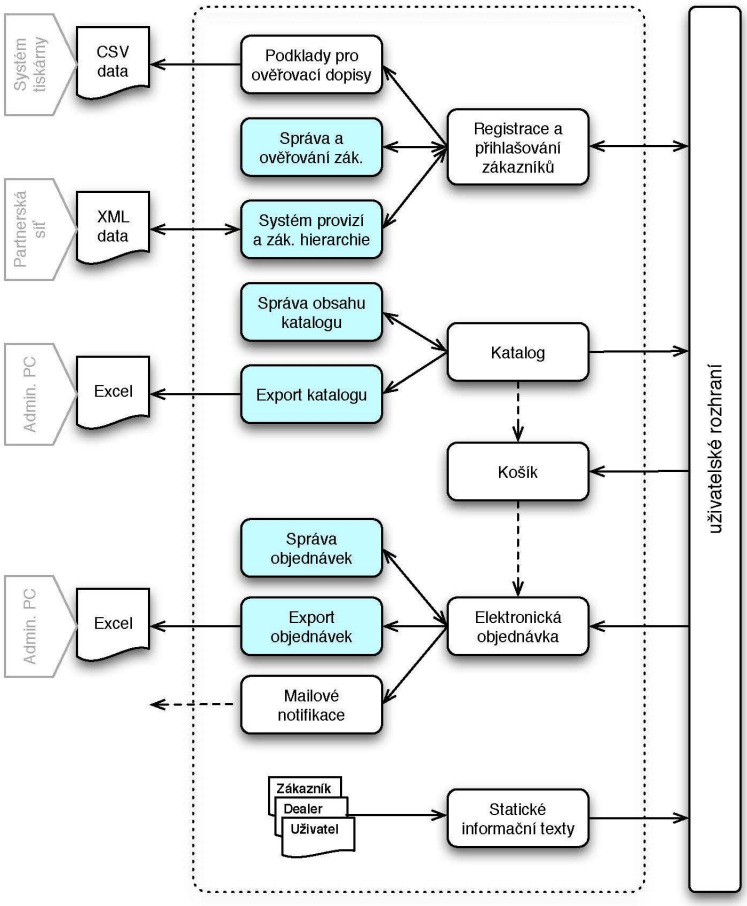
Tisk, zabalení a distribuci registračních dopisů realizuje pro společnost externí tiskárna, která zpřístupňuje pro každého svého zákazníka vlastní FTP konto, kam systém automatizovaně 1x denně nahrává CSV soubor se všemi nově registrovanými zákazníky. Tiskárna dopisy tiskne v okamžiku, kdy je počet větší než 30 nebo je nejstarší soubor starší než 3 týdny.

Mimo provizního systému firma spolupracuje s ostatními obchody, kde mohou registrovaní zákazníci získat bonusové body za nákup. Síť obchodů využívá ke sdílení a vzájemnému započtu bonusových bodů centrální výměnný systém, do něhož všechny obchody odesílají data o uskutečněných obchodech a zpět získávají data o připsaných bonusech. Tato synchronizace obchodů probíhá na denní bázi prostřednictvím XML souborů definovaného formátu a HTTP rozhraní.

Obchod nepoužívá žádný vlastní informační nebo skladový systém, stav skladu, dostupnost i vlastní katalog je spravován přes administrační rozhraní internetového obchodu, přičemž na žádost umožňuje stáhnout offline zálohu aktuálního stavu celého katalogu do Excelu a dále s ním pracovat v oblasti tištěných médií, apod.

Nákup je uskutečňován přes procházení katalogu, který je tříděn do kategorií podle druhu

[1] Projekt je naplánovaná, *neopakovatelná* a časově vymezená činnost, či soubor činností, na jejímž konci je *jedinečný* produkt nebo služba [10].



Obrázek 3.1: Funkční schéma projektu Alfa

sortimentu, detail produktu obsahuje obrázek, textový popis a cenu. V katalogu lze fulltextově vyhledávat, nicméně není implementováno žádné pokročilé vyhledávání, nebo porovnávání výrobků, apod. Prohlížet katalog si může jakýkoliv uživatel, nakupovat pouze uživatel registrovaný. Nákup je realizován přes standardní nákupní košík, perzistentní mezi přihlášením a odhlášením. Pro úhradu zboží se u ověřených registrací používá standardní fakturace a bankovní převod, u koncových uživatelů je to buď platba předem, u zásilek do určité hodnoty je povolena dobírka.

Objednávky jsou evidovány opět primárně administračním rozhraním internetového obchodu, stejně tak zde jsou evidovány aktuální stavy, včetně vlastní historie. Objednávky lze také exportovat do excelového sešitu a stáhnout je na počítač administrátora. O uskutečněné objednávce, úspěšně doručené do systému, je zákazníkovi odesílán potvrzující e-mail, stejně tak jako při každé změně stavu objednávky. E-maily jsou distribuovány online, tj. okamžitě při změně stavu.

Vzhled celého obchodu je víceméně stejný, zákazníkům s prémiovými účty (*zákazník* a *dealer*) je umožněno vložit na určená místa vlastní texty, které uvidí všichni jeho přímo podřízení uživatelé. Protože každá úroveň může upravovat různý počet textových elementů, je uživatelské rozhraní jiné pro každou kategorii, aby na stránkách nevznikala prázdná místa.

3.2.2 Funkční schéma

Na schématu na obrázku 3.1 je zobrazeno základní fungování obchodu kancelářskými potřebami, klíčové procesy a vazby. Dílčí subsystémy jsou umístěny pod sebou ve sloupečku podle toho, jestli se jedná o systém pro komunikaci s uživatelem, nebo jestli je to systém podpůrný. Procesy označené modrou barvou jsou administrovány správcem aplikace, tj. mají administrační/redakční systém (CMS) a umožňují tak zásah správce. Aplikace má navíc ještě 4 rozhraní pro komunikaci s externími systémy, nebo pro export dat.

3.2.3 Průběh projektu

Aplikace byla vyvíjena od 12. 2. 2007 do 3. 5. 2007; vývoj byl tedy dokončen za 80 dní (počítaje i dny pracovního klidu). Na projektových aktivitách bylo spotřebováno 105 MD (840 WH). Pro vývoj bylo využito programovacího jazyka PHP, databázového serveru mySQL, webového serveru Apache. Vlastní aplikace byla vyvíjena prostřednictvím frameworku CakePHP. Vývoje se účastnilo následující složení projektového týmu:

- 1 Projektový manažer
- 1 Analytik
- 1 Grafik - externí
- 1 HTML kodér
- 2 Programátoři

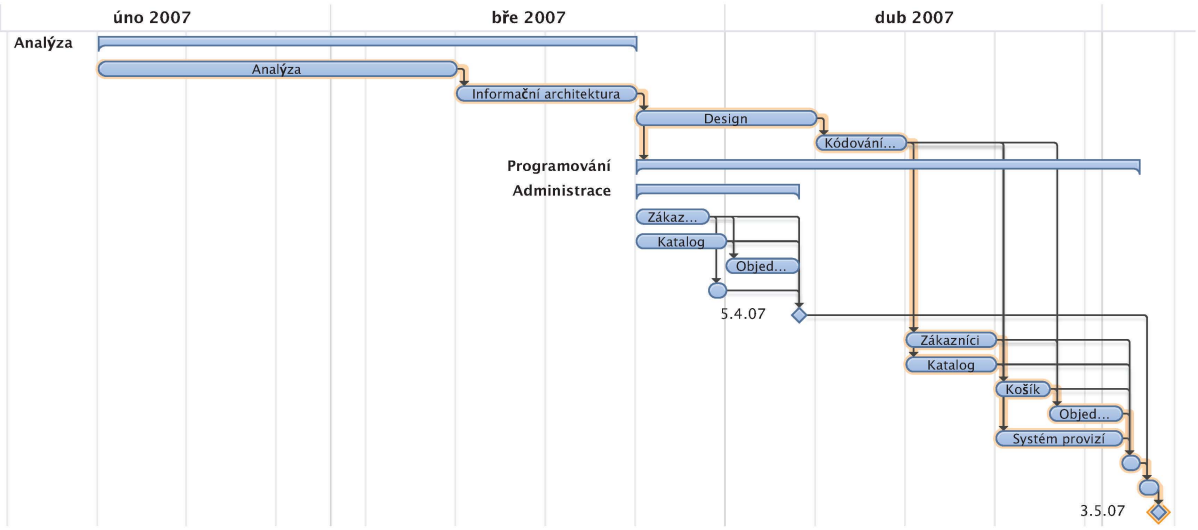
Náklady na implementaci řešení byly v době podání nabídky *odhadovány* na CZK 610 000 (– rezerva možného navýšení ve výši 9 %, tj. včetně rezervy cca CZK 665 000) a termín dokončení na 10. 4. 2007. *Finální náklady* na vývoj, zjištěné na základě skutečnosti, vyjádřené v interních cenách, zahrnující odměny členů týmu, režijní náklady, přepočtené náklady na získání zakázky byly vyčísleny na CZK 756 000 (nárůst o 24 % oproti původnímu plánu a o 14 % oproti plánu s rezervou). Zákazník za dodávku zakázky zaplatil CZK 797 000. Aby projekt dosáhnul požadovaného zisku 20 %, musela by být prodejní cena cca CZK 907 000, z čehož vyplývá, že projekt byl neúspěšný, kdy výnosy pokryly pouze náklady. Projekt byl podceněn během svého plánování a byl dodán téměř s měsíčním zpožděním.

3.2.4 Zhodnocení

3.2.4.1 Přínosy použitého řešení

Řešení postavené na frameworku CakePHP je *příkladem standardní MVC* architektury. Framework disponuje *množstvím funkcí*, které *urychlují práci* během programování a nemusí tak být zajišťovány samostatnými funkcemi. Výhodou je již nativní *podpora překladu srozumitelných URL adres* na případy volání aplikačního rozhraní. Šablony i programový kód jsou v *oddělených adresářích* a spravují se taktéž odděleně. Je možné definovat *samostatná práva* přístupu k jednotlivým složkám.

Task	Effort	Start	End
● 1) **Analýza**	6w	**12.2.07**	23.3.07
● 1.1) Analýza	4w	12.2.07	9.3.07
● 1.2) Informační architektura	2w	12.3.07	23.3.07
● 2) Design	2w	26.3.07	6.4.07
● 3) Kódování šablon	1w	9.4.07	13.4.07
● **4) Programování**	8w 4d	**26.3.07**	2.5.07
● **4.1) Administrace**	3w 3d	**26.3.07**	5.4.07
● 4.1.1) Zákazníci	4d	26.3.07	29.3.07
● 4.1.2) Katalog	1w	26.3.07	30.3.07
● 4.1.3) Objednávky	4d	2.4.07	5.4.07
● 4.1.4) Texty	1d	30.3.07	30.3.07
◆ 4.1.5) Předání		5.4.07	5.4.07
● 4.2) Zákazníci	1w	16.4.07	20.4.07
● 4.3) Katalog	1w	16.4.07	20.4.07
● 4.4) Košík	3d	23.4.07	25.4.07
● 4.5) Objednávky	4d	26.4.07	1.5.07
● 4.6) Systém provizí	1w 2d	23.4.07	1.5.07
● 4.7) Statické stránky	1d	2.5.07	2.5.07
● 5) Školení	1d	3.5.07	3.5.07
◆ 6) Spuštění		3.5.07	3.5.07



Obrázek 3.2: Finální průběh projektu Alfa

Protože v projektu používaný redakční/administrační systém (CMS) má jednotný, již *předem definovaný design* a dynamicky *generované uživatelské rozhraní*, bylo možné zahájit práce na jeho *realizaci v předstihu*, tj. před dokončením designu a HTML šablon, což vyplývá z přiloženého projektového plánu na obrázku 3.2. Připravené modely bylo možné následně *využít při tvorbě vlastního rozhraní* obchodu.

3.2.4.2 Identifikované problémy

Na začátku nebylo příliš jasné zadání a proto probíhala *dlouho iniciační fáze* analýzy (– 2 týdny). Prvním skutečným problémem při vlastním vývoji bylo *zahrnutí externího designera*, který se *opozdil* se svojí dodávkou o týden. Programátoři byli alokováni na přípravu administračního rozhraní a v mezičase měl být dokončen design a připraveny alespoň základní šablony. Nicméně design byl dokončen až v okamžiku dokončení administračního rozhraní. Programátoři byli tedy z projektu *převedeni na jiný projekt*, aby mohly být připraveny šablony, které byly předpokladem tvorby frontendu aplikace. Příprava šablon se opět díky opravám a špatným výstupům od grafika *zpozdila* a díky tomu byl *opožděn i celý vývoj* uživatelské části aplikace.

Díky schopnostem programátorů se podařilo poté šablony integrovat do aplikace v kratší, než plánované době, ovšem za cenu nestandardního přístupu, kdy šablony integroval programátor a nikoliv kodér, což si navýšilo spotřebu času (alokaci) programátora ve fázi, kdy se měl věnovat spíše aplikační logice. Při finalizaci řešení se opakovaně objevovaly problémy s konzistencí webového rozhraní, které musel kodér ve spolupráci s programátory pracně a opakovaně opravovat.

Během vývoje frontendové části musely být opravovány některé chyby týkající se synchronizace zákaznických účtů s partnerskými shopy. Jednalo se o časové údaje a validitu vymčňovaných dokumentů. Z tohoto důvodu musel být přepracován koncept systému provizí a partnerských bodů.

Zpoždění vývoje zapříčinilo, že testování nemohlo být provedeno v plném rozsahu a omezilo se pouze na základní systémové testy. Zjištěné chyby, množství nedodělků, které bylo třeba odstranit před předáním díla zákazníkovi, generovaly další potřebu alokace zdrojů na jejich opravu.

3.2.4.3 Příležitosti pro použití metodiky ETWA

Všechny klíčové činnosti projektu se při konvenčním způsobu vývoje nachází na kritické cestě. To je dáno závislostí činností na sobě a nemožností pracovat na některých částech výrazně v předstihu. V průběhu projektu se toto také projevilo na opožděné externí dodávce grafiky. Celkové zpoždění pak bylo eliminováno intenzivní spoluprací programátorů a kodéra. Tímto způsobem se ovšem zvyšuje alokace jednotlivých rolí. Metodika ETWA by tomto případě umožnila:

- Zahájit vývojové práce frontendu ještě před začátkem prací na šablonách,
- jasně oddělit práce programátorů a kodéra, bez výrazného navýšení alokace,

- odhalit problémy nekvalitní specifikace v okamžiku přípravy vzorového XML dokumentu,
- dokončit přípravu šablon a programování ke stejnému termínu, díky nezávislosti činností,
- provádět během vývoje a na konci projektu alespoň systémové testy (včetně metodikou navržené validace).

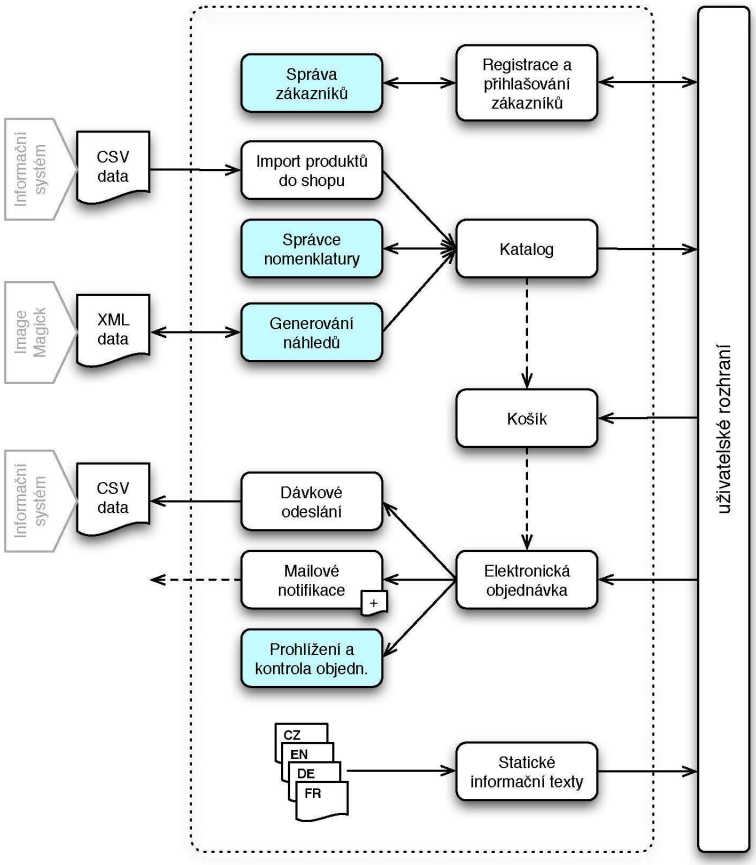
3.2.4.4 Ohrožení použití metodiky ETWA

V tomto projektu bylo využíváno veřejně dostupného frameworku CakePHP. Díky jeho vestavěným funkcím a fungujícímu konceptu model-view-controller je vývoj rychlejší, menší pracnost je například s databázovými dotazy, validací a převody dat, apod. Určitá ohrožení spojená především s přechodem na novou metodiku je možné shrnout do následujících bodů:

- Protože metodika ETWA nedisponuje implementací v žádném frameworku, znamená její použití vzdát se některých pokročilých funkcí běžných frameworků.
- Metodiku by bylo možné ji implementovat přímo do CakePHP, nicméně by bylo v budoucnu náročné přejít na jeho novější verzi, protože by se musela implementace metodiky opakovat a případně upravovat dle nových funkcí frameworku.
- Přechodem na XSL verzi šablon stránek se výrazně snižuje možnost zástupu kodéra programátorem, což může být v kritických okamžicích projektu (zpoždění, nedostupnost zdrojů, apod.) nevýhodou. V projektu Alfa tedy především v jeho závěru.
- Programovací jazyk PHP obecně není připraven na podporu XML, proto by některé běžně používané úkoly vyžadovaly zvláštní (a pravděpodobně i delší) způsob programování, aby bylo kolektorem produkované XML správně hierarchicky členěno.

3.2.4.5 Závěr

Jak je vidět z Ganttova diagramu na obrázku 3.2 jsou činnosti Analýza, informační architektura, design, šablony a programování frontendu na kritické cestě. Zpoždění jakékoliv z těchto činností způsobuje prodloužení projektu, což se také v tomto případě stalo a odrazilo se ve zpoždění data dodání projektu. Neefektivně byly využity programátorské zdroje, které musely být převedeny na krátkou dobu na jinou práci, nicméně pozitivní pro průběh projektu byla možnost jejich návratu okamžitě, jakmile bylo vše pro jejich práci připraveno, což nemusí být pravidlem. Negativní vliv na celý průběh projektu měla poměrně dlouhá analytická fáze, kde bylo vyjasňováno přesné fungování webového rozhraní s rozpadem na funkčnost jednotlivých obrazovek, aby bylo zadání, pokud možno, co nejkomplexněji popsáno. Metodika ETWA by v tomto projektu mohla být použita a zcela určitě by pomohla eliminovat některé negativní vlivy zmíněné v oddílu 3.2.4.3. Současně se při pohledu na konvenční vývoj a jeho průběžné porovnání s metodikou navrhovaným způsobem vývoje objevuje spousta otázek a ohrožení, která bude nutné při přechodu nutné vyřešit, což s sebou přináší každá obdobná změna.



Obrázek 3.3: Funkční schema projektu Beta

3.3 Projekt Beta

3.3.1 Popis projektu

Projekt Beta je také internetovým velkoobchodem s náhradními díly na domácí i průmyslové elektrospotřebiče. Společnost postupně přesouvá svůj současný, úspěšný obchod, založený na klasických tištěných katalozích a nabídkových listech, do prostředí internetu. Z toho vyplývá, že již má svoji databázi zákazníků, kterým chce nabídnout pohodlnější a rychlejší způsob nakupování, doplněný i o fotografie a technické popisy dílů. V první fázi chtěla společnost zpřístupnit internetový obchod pouze současným zákazníkům, protože ti zboží dále přeprodávají, včetně své marže. Nicméně byla připravena i možnost registrace nových zákazníků, přístupná jen po ověření pracovníkem společnosti.

Veškeré údaje a skladové hospodářství má firma již hotové a provozuje na něm současné dodávky. Řešení je ovšem postaveno stále ještě na bázi DOSu, což umožňuje jeho běh na terminálech ve skladu, nicméně tento systém má jen omezené síťové funkce. Výrobky v systému nejsou zařazovány do kategorií, zaměstnanci rozpoznávají kategorie dle čísla nomenklatury, nebo její struktury. Jedná se o hlavní systém společnosti a proto se s ním musí internetový obchod synchronizovat hlavně kvůli dostupnosti některých výrobků, cenám a novým výrobkům v nabídce. V rámci administrace systému byla vytvořena správa kategorií produktů.

Výrobky se do jednotlivých kategorií vkládaly na základě definice masky nomenklatury a logických spojek. Systém pak nabízí odpovídající výrobky a umožňuje některé jednotlivě vyřadit. Toto představovalo největší úskalí celého návrhu.

Synchronizace s interním informačním systémem je prováděna na základě výměny CSV souborů s kompletním obsahem katalogu. Díky tomu, že jsou kategorie definovány jako maska nomenklatury je možné při každé synchronizaci vyčistit starý seznam produktů a přehrát ho novou verzí z CSV.

Protože fotografie tohoto typu výrobků jsou poměrně cenné – neboť nejsou běžně k dispozici na internetu rozhodla se společnost obrázky chránit vodoznakem, s tím, že malý náhled bude bez vodoznaku a větší s ním. Obrázky proto fotograf nahrává do speciálního adresáře a následně manuálně spustí proces generování. Ten vytvoří dávkový XML soubor se seznamem nezpracovaných obrázků a předá jej konzolové dávce, která na základě něho připraví dvojici obrázků, umístí je do nového adresáře a vytvoří obdobný XML dokument, který obsahuje informace o tom, ke kterým obrázkům se podařilo náhledy vygenerovat.

Objednávka je prováděna prostřednictvím standardního košíku, do kterého jde mimo přímého výběru z katalogu přidávat produkty zadáním nomenklatury. Díky tomu, že firma bude distribuovat svým existujícím partnerům, kterým už v současné době zboží rozesílá obchodními balíky, nebo rozváží vlastní službou, není v obchodě implementována žádná platební brána a systém tak víceméně pouze sbírá objednávky registrovaných uživatelů. Objednávky jsou v dávkách odesílány ve formě CSV souborů do interního informačního systému. Dále systém generuje objednávku ve formátu CSV a odesílá ji e-mailem jak zákazníkovi, tak do dispečerské fronty společnosti, kde si vloženou přílohu zpracuje automat a zařadí ji do fronty zpracování, protože dispečink nemá přímý přístup do informačního systému. Objednávky jsou již jen pro kontrolu zobrazovány také v administračním rozhraní.

Vzhledem k tomu, že firma působí na celosvětové úrovni, má také svůj obchod ve více jazykových mutacích (konkrétně česky, anglicky, německy, francouzsky a rusky). Protože katalog obsahuje pouze technické údaje, které není nutné překládat, je jazyková mutace pouze věcí šablony, kde musí být přeloženy jazykově různé části a vynechány některé elementy relevantní pouze pro českou verzi.

3.3.2 Funkční schéma

Na schématu na obrázku 3.3 je zobrazeno základní fungování velkoobchodu náhradními díly na elektrospotřebiče. Dílčí subsystémy jsou umístěny pod sebou ve sloupečku podle toho jestli se jedná o systém pro komunikaci s uživatelem, nebo jestli je to systém podpůrný. Procesy označené modrou barvou jsou administrovány správcem aplikace, tj. mají administrační/redakční systém (CMS) a umožňují zásah správce. Aplikace má navíc ještě 3 rozhraní pro komunikaci s externími systémy a 1 rozraní založené na mailové komunikaci.

3.3.3 Průběh projektu

Internetový obchod byl vyvíjen od 28. 5. 2007 do 27. 7. 2007; vývoj aplikace byl tedy dokončen za 60 dní (počítaje i dny pracovního klidu). Na projektových aktivitách bylo spotřebováno 101 MD (= 808 WH). Pro vývoj bylo využito programovacího jazyka PHP,

databázového serveru mySQL, webového serveru Apache. Vlastní aplikace byla postavena na existujícím vlastním frameworku, založeném na bázi objektového programování, který byl modifikován dle požadavků metodiky ETWA. Tato úprava nebyla kalkulována do časové spotřeby projektu. Vývoje se účastnilo následující složení projektového týmu:

- 1 Projektový manažer
- 1 Analytik
- 1 Grafik
- 1 HTML kodér
- 2 Programátoři
- 1 Tester

Náklady na implementaci řešení byly v době podání nabídky *odhadovány* na CZK 655 000 (rezerva možného navýšení ve výši 9 %, tj. včetně rezervy cca CZK 714 000) a termín dokončení na 15. 8. 2007. *Finální náklady* na vývoj, zjištěné na základě skutečnosti, vyjádřené v interních cenách, zahrnující odměny členů týmu, režijní náklady, přepočtené náklady na získání zakázky byly vyčísleny na CZK 727 000 (nárůst o 11 % oproti původnímu plánu a o 2 % oproti plánu s rezervou). Zákazník za dodávku zakázky zaplatil CZK 856 000. Aby projekt dosáhnul požadovaného zisku 20 %, muselo by být prodejní cena cca CZK 872 000, z čehož vyplývá, že projekt sice ani tak nesplnil očekávání, ale rozdíl v odhadu byl víceméně pokryt projektovou rezervou a tudíž bylo dosaženo minimálního zisku. Velice pozitivním signálem je zde zkrácení doby vývoje o téměř měsíc, při stejné pracnosti v MD a to včetně integračního testování.

3.3.4 Zhodnocení

3.3.4.1 Přínosy použitého řešení a metodiky ETWA

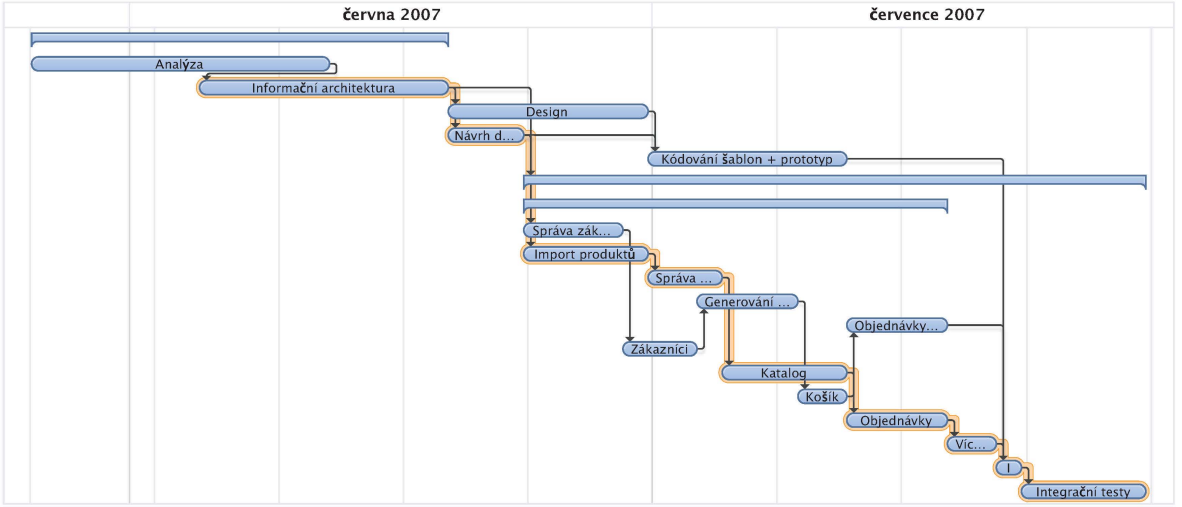
Ačkoliv na počátku projektový tým některým činnostem nerozuměl, např. proč se má dohodnout na formátu XML souboru, když se ještě bude v budoucnu může měnit, nebo proč je nutné výstupní soubor z aplikace validovat. Nakonec byl ale právě tento krok hodnocen pozitivně zejména z toho důvodu, že byl celý tým zapojen do projektu již na začátku a díky definování vzájemného rozhraní si ujasnil mnoho otevřených bodů před zahájením prací.

Velice efektivní se, obzvláště u takto mnohojazyčného webu, ukázal formát XML, díky kterému bylo možné editovat statické texty přímo v azbuce ve statických XML souborech. Samotná transformace různých kódování fungovala naprosto bezproblémově na všech úrovních. Kolektor statických jazykových dat usnadnil práci a ošetřování výjimek.

Díky samopopisnosti a jednoduchosti formátu XML byl dokonce zadavatel schopen dodat přeložené statické texty a číselníky přímo ve vzorovém XML souboru, což ušetřilo čas nutný k případnému převodu z jiného formátu.

Již na samotném obrázku Ganttova diagramu (3.4) je vidět, že programátoři se neomezovali na rozdíly mezi administračním rozhraním frontendem. Činnosti byly naplánovány za sebe tak, jak jdou logicky bez ohledu na typ výstupu, takže posloupnost činností byla ve

Task	Effort	Start	End
1) Analýza	4w 2d	**28.5.07**	19.6.07
1.1) Analýza	2w 2d	28.5.07	12.6.07
1.2) Informační architektura	2w	5.6.07	19.6.07
2) Design	1w 3d	19.6.07	29.6.07
3) Návrh dat. základny	3d	19.6.07	22.6.07
4) Kódování šablon + prototyp	1w 3d	29.6.07	11.7.07
5) Programování	8w 2d	**22.6.07**	27.7.07
5.1) Administrace	4w	**22.6.07**	17.7.07
5.1.1) Správa zákazníků	4d	22.6.07	28.6.07
5.1.2) Import produktů	1w	22.6.07	29.6.07
5.1.3) Správa nomenklatur	3d	29.6.07	4.7.07
5.1.4) Generování náhledů	4d	3.7.07	9.7.07
5.1.5) Objednávky interface	4d	11.7.07	17.7.07
5.2) Zákazníci	3d	28.6.07	3.7.07
5.3) Katalog	1w	4.7.07	11.7.07
5.4) Košík	2d	9.7.07	11.7.07
5.5) Objednávky	4d	11.7.07	17.7.07
5.6) Vícejazyčné moduly	2d	17.7.07	19.7.07
5.7) Integrace	1d	19.7.07	20.7.07
5.8) Integrační testy	1w	20.7.07	27.7.07



Obrázek 3.4: Finální průběh projektu Beta

výsledku jiná, než jak se původně nadefinovala (plánovala se bez ohledu na použití metodiky ETWA). Jakmile si tým osvojil konverzi prostřednictvím XSL, bylo velice jednoduché implementovat rozhraní pro externí systémy - tj. konverzi z XML do CSV.

Díky naplnění časování projektu na konci zbyl i přesto čas na integrační a systémové testy. Při testech se bylo možné soustředit více na integrační testy, protože základní systémové nedostatky byly odstraňovány již v průběhu vývoje - díky validaci i pokusnému napojování na rozpracované šablony.

Použití metodiky ETWA mělo na projekt následující pozitivní vlivy:

- Jednotlivé role pracovaly na projektu víceméně nezávisle na sobě,
- zvolený formát XML se osvědčil jako velmi vhodný, nejen z pohledu vlastní tvorby, ale také zahrnutí různých jazykových kódování,
- dílčí výsledky práce byly průběžně validovány a testovány a nedošlo tak k odhalení závažných systémových chyb na konci projektu,
- implementace rozhraní s ostatními systémy v různých datových formátech byla díky XML/XSL velmi jednoduchá,
- řešení bylo průběžně ověřováno přímo se zákazníkem na základě funkčního prototypu,
- na vzorových XML datech bylo možné ověřovat a testovat různé stavy aplikace bez zprovozněné aplikace na pozadí,
- při testování bylo s pozitivním výsledkem využito otevřené XML transportní vrstvy, kde mohlo být zpracování zastaveno a data aplikace validována nezávisle na uživatelském rozhraní.

3.3.4.2 Identifikované problémy

Nejnáročnějším úkolem bylo přesvědčení týmu, aby vůbec metodiku použil a dodržel všechny její požadavky. Fáze tvorby šablon se prodloužila, kvůli nutnosti naučit se nový jazyk a osvojit si jeho konstrukce přímo na konkrétních případech. Na konci ale stály funkční prototypy stránek.

Po celou dobu byla nutná intenzivní spolupráce autora (projektového manažera) s týmem, aby zodpověděl všechny dotazy na konkrétní postup, případně ujasnil kam daná zodpovědnost spadá. Vzhledem k prolínání administrační části a frontendu nad jedním aplikačním kódem bylo nutné vyřešit i způsob jakým budou řešena práva a přístup k jednotlivým částem.

Ačkoliv bylo poměrně jednoduché transformovat XML data do CSV formátů určených pro přenos, musel být na straně aplikace proveden zásah, který umožní vrátit transformovaný výsledek zpět do aplikační vrstvy, která zajistí jeho transport, neboť v tomto případě se vždy jednalo o přenos iniciovaný vytvářenou aplikací.

Poměrně náročné bylo vytváření vzorového XML dokumentu a validačního schématu, neboť postupem času tým teprve zjišťoval, co skutečně ve výstupu potřebuje a proto byl

opakovaně doplňován a měněn. Na vině byla také chybějící zkušenost s podobným způsobem přípravy vývoje. Protože výroba šablon trvala delší dobu, mohly na změnu vždy reagovat obě strany (kodér i programátoři). Při programování se zjistilo, že podpora XML v PHP není příliš propracovaná a musely se tak kombinovat různé postupy, aby bylo možné z PHP jednoduchým způsobem vygenerovat dokument v dohodnutém formátu a hierarchické struktuře. Problémem bylo i nalezení vhodného validátoru RelaxNG schématu pro PHP a aplikační ošetření identifikovaných chyb.

Průběžně bylo nutné také doplňovat vlastní framework o nové funkce související s výstupem aplikace do XML kolektoru. Ačkoliv byl framework připravován pro účely nasazení metodiky již před zahájením projektu, bylo úložiště obecných funkcí a tříd na konci projektu rozšířeno o zhruba polovinu dalších podpůrných funkcí a objektů.

Při implementaci šablon v XSL byly identifikovány problémy v různých implementacích XSL parserů a procesorů, kdy např. procesor internetového prohlížeče provedl transformaci jinak než procesor PHP. Proto bylo nutné provést v integrační fázi také změny v hotových XSL šablonách. Nicméně se jednalo o opakující se typizované změny.

3.3.4.3 Závěr

Je překvapující, že ačkoliv byla aplikace vyvíjena zcela novým způsobem, s minimální možností přípravy a neustálým hledáním vhodných nástrojů či konstrukcí jazyka, byl projekt dokončen v takto krátké době. V tomto případě je za úspěch možné považovat dokončení ve stejném čase jako u projektu vyvíjeného konvenčním způsobem. Protože měl ale projekt velkou podporu, nebyl tím ovlivňován jinými činnostmi, které standardně vykonává a mohl se tak plně věnovat této práci. Většina problémů a otevřených otázek se vyskytovala na implementační úrovni, tj. ve volbě nástrojů, rozdělení odpovědností, omezeních jazyka, apod. Kromě cílového chování aplikace vznikaly při vývoji také dílčí části frameworku pracujícího na základě metodiky ETWA.

Metodika ETWA byla na tomto projektu ověřena jako aplikovatelná. Během vývoje nebyl identifikován žádný nelogický, nebo bezvýchodný prvek. Projekt byl dodán v termínu úměrném nasazování nové technologie. Vývoj produktu probíhal přesně podle navrhované metodiky. V rámci implementace bylo využito dílčích příkladů uvedených v této práci, které byly upraveny pro spolupráci s vlastním firemním frameworkem. Ačkoliv se při prvním vývoji dle navržené metodiky nepodařilo docílit evidentního zkrácení celkové doby vývoje, očekává se, že osvojením práce dle metodiky, bude celková doba skutečně kratší a průběh projektu plynulejší. Celkově lze také očekávat vyšší kvalitu výsledného produktu, zejména díky lepšímu a propracovanějšímu testování.

Na závěr lze tedy konstatovat, že tímto projektem bylo ověřeno nasazení metodiky ETWA v reálných podmínkách a v rámci konkrétní implementace v jazyce PHP.

3.4 Porovnání obou projektů

Jak již bylo uvedeno na začátku této kapitoly není možné porovnávat dva různé projekty na základě absolutních ukazatelů, neboť každý projekt je od své podstaty neopakovatelnou,

unikátní činností, která je ovlivňována různými jevy a okolnostmi. V tomto případě byly vybrány dva projekty, které vytvářel velmi podobný tým a oba byly z hlediska spotřebovaných MD stejné. Tyto dva projekty byly vybrány také proto, že jejich cílem byly podobné internetové obchody, s podobným počtem funkčních modulů i externích rozhraní. Absolutní ukazatele, které již byly zmíněny ve zhodnocení jednotlivých projektů jsou pro porovnání shrnuty ještě do tabulky 3.1.

Ukazatel	Projekt ALFA	Projekt BETA
Cena	797 000	856 000
Odhadované náklady	610 000	655 000
Skutečné náklady	756 000	727 000
Odpracované MD	105	101
Začátek projektu	12. 2. 2007	28. 5. 2007
Konec projektu	3. 5. 2007	27. 7. 2007
Plánované ukončení	10. 4. 2007	15. 8. 2007
Přečerpání nákladů	+ 14 %	+ 2 %

Tabulka 3.1: Hodnotové porovnání projektů Alfa a Beta

Na uvedených příkladech z praxe byly demonstrovány a ověřeny následující předpoklady a závěry disertační práce:

- Problémy jednotlivých rolí, identifikované v oddílu 2.1se projevily i během těchto realizace těchto případových studií. Znatelné byly především u projektu Alfa, který byl realizovaný standardním způsobem. Nicméně projektové týmy jsou již na obdobné překážky zvyklé a neměly přímý dopad do projektu jako takového a proto jim nebyla věnována zvláštní pozornost ani v tomto zhodnocení.
- Nebyl ověřen vliv zásadních změn zadání, protože v případě obou projektů byli zadavatelé poměrně konzistentní ve svých požadavcích a proto nebylo nutné řešit změnové požadavky, či prodlužovat projekt.
- Pro použití metodiky ETWA v praxi je nutné na tuto změnu připravit jak technologie (zejména používané postupy, funkce a frameworky), tak členy projektových týmů.
- Na úrovni technologií je nutné dobře promyslet a správně navrhnout XML kolektor, který je základní součástí celého konceptu metodiky ETWA.
- Projektové týmy je nutné předem vyškolit nejen na novou metodiku a odlišný způsob vývoje, ale také v oblasti používaných jazyků (XML, XSL, RelaxNG, apod.).
- Zavedení a očekávaný efekt zavedení metodiky ETWA vyžaduje určitý čas a realizace alespoň několika projektů novým způsobem.
- Nasazení metodiky se projevuje na všech projektových úrovních - tedy v projektovém řízení, programování, kódování i testování. Metodika zásadně neovlivňuje analytickou část a přípravu designu.

- ▷ Díky nezávislosti jednotlivých činností je možné počítat i s outsourcingem vývoje nebo kódování šablon. Výhodnou je nezávislost těchto částí, jednoduchá akceptační kritéria a snadné ověření.

Větší vypovídají hodnoty absolutních ukazatelů případových studií by bylo možné dosáhnout ověřením na větším množství projektů a porovnáváním s celým projektovým portfoliem společnosti. Zejména pak délky realizace, spotřebované jednotky práce a počtu alokovaných zdrojů. Realizace stejného projektu dvěma nezávislými týmy konvenčním způsobem a dle metodiky ETWA, bylo z ekonomických důvodů zavrženo. I tak by porovnání nemělo potřebnou vypovídající hodnotu, neboť ukazatele projektu ovlivňuje také kvalita projektového týmu, jeho zkušenosti a specializace.

Použití metodiky bylo ověřeno jako reálné. Větší změny bylo nutné provést především na úrovni technologií. Jinak s jejím zavedením středně kvalitní projektový tým neměl závažnější problémy.

Závěr

Cíle předložené disertační práce byly stanoveny v úvodu a poté byly postupně zpřesňovány v jednotlivých kapitolách, kde byly také naplňovány. Obsah práce pokrývá všechny stanovené cíle, v části vlastního řešení jsou uváděny také dílčí závěry, které vyplynuly z návrhu jako vedlejší přínos.

V první části práce byly zpracovány různé pohledy na zvláštnosti internetových projektů, zahrnující nezanedbatelný význam uživatelského rozhraní, účast nestandardních rolí, častý přechod z vývoje v plynulý rozvoj, apod. Vlastní řízení takovýchto projektů bylo identifikováno jako jedno z hlavních úskalí, zejména pak nepřesné plánování těchto projektů, komplikace způsobené zapojením zadavatele a s tím související opakované úpravy zadání. Protože se obsahem práce prolínají odkazy na různé technologie a návrhy nové metodiky se odráží také v technologické rovině, bylo do práce zpracováno krátké představení technologií, které jsou v práci používány pro důkazy a ověření. Popsány byly technologie XML, XSL a PHP, které jsou klíčové pro praktické příklady.

Byly shromážděny a popsány způsoby, kterými je možné vytvářet webové aplikace, způsoby generování výstupů pro webové prohlížeče, včetně uvedení výhod a úskalí každého z těchto řešení. Popsány byly zejména současné šablonové systémy. Na základě studia různých literárních pramenů a internetových zdrojů byly charakterizovány webové aplikace a jejich specifika, odlišující je od vývoje běžných aplikací. Do procesu vývoje jsou zahrnovány i ty profese, které se zpravidla vývoje, dle běžného chápání obsahu této činnosti, neúčastní. Jedná se zejména o role grafika a informačního architekta.

Webové aplikace byly rozděleny podle způsobu jejich využití. V současnosti nejvýraznějšími a nejčastějšími typy těchto aplikací jsou elektronické obchody, veřejné a firemní portály, webové informační systémy a aplikace nabízené formou pronájmu služby (ASP). Význam webových aplikací je spatřován v hardwarové a softwarové nezávislosti, rychlé a bezproblémové aktualizaci, ochraně autorských práv výrobce a obecně dobré dostupnosti. Zároveň byly také identifikovány hlavní slabiny ve vývojovém procesu webových aplikací.

Předkládaná práce se především zaměřuje na návrh metodiky vývojového procesu webových aplikací. Byly shromážděny a popsány různé přístupy a používané procesy, související s touto problematikou. V závěru první kapitoly byly popsány a vymezeny projektové role, které se účastní vývoje webových aplikací, včetně náplně jejich práce, způsobu zapojení do projektu, hlavních zodpovědností, vzájemných vztahů a očekávaných výstupů. Na základě tohoto rozboru byly dále v práci definovány požadavky na novou metodiku z pohledu uváděných projektových rolí.

Metodika ETWA

Cílem disertační práce je návrh vlastní metodiky označené jako ETWA (Efektivní Tvorba Webových Aplikací). Na základě provedeného průzkumu mezi reprezentanty jednotlivých projektových rolí, spolupráce s internetovými agenturami a teoretických podkladů byly nejprve určeny překážky, které obecně snižují efektivnost práce na projektu webové aplikace, zpožďují jeho dodávku nebo snižují kvalitu výsledného výstupu. Na základě těchto poznatků byly definovány požadavky na navrhovanou metodiku.

Základem navržené metodiky ETWA je zavedení nové transportní mezivrstvy mezi aplikační a prezentační vrstvu webové aplikace. Dále pak také zavedení XML kolektoru, zajišťujícího realizaci transportní vrstvy, striktní validaci dat a zcela odlišný způsob tvorby šablon uživatelského rozhraní. Zavedením těchto nových prvků nové metodiky se podařilo dosáhnout všech definovaných cílů a eliminace zjištěných překážek.

Metodika ETWA technologicky nezávislá a dostatečně univerzální pro použití v různých podmínkách. Na základě příkladů konkrétních implementací, bylo v práci dokázáno, že je možné navržené postupy realizovat také v praxi za použití běžně dostupných prostředků a konstrukcí.

V oddíle 2.2byly stanoveny obecné požadavky na navrhovanou metodiku. Dále jsou zhodnoceny způsoby, jak metodika ETWA umožňuje tyto požadavky naplňovat:

1. *Zefektivnit celý vývojový proces webové aplikace*

   - *Zrychlit doručení produktu* – postup dle metodiky umožňuje odstranit závislosti klíčových rolí, resp. fází designu, tvorby uživatelského rozhraní a samotného vývoje a díky němu je možné provádět některé činnosti paralelně bez toho, aby se vzájemně negativně ovlivňovaly, především z hlediska posunu dokončení jednotlivých aktivit, což umožňuje zkrátit dobu dodávky.
   - *Umožnit flexibilní alokaci zdrojů pracujících na projektu* – tento požadavek byl splněn částečně, návrh metodiky pokrývá možnost flexibilní alokace grafiků, kodérů a programátorů. U rolí analytiků a testerů jsou možnosti flexibilní alokace omezeny. Flexibility alokace zmíněných rolí bylo dosaženo odstraněním závislostí fází projektu, kterých se dané role účastní.

2. *Eliminovat závislosti rolí na sobě navzájem, umožnit paralelní alokaci* – částečně vyřešeno v rámci požadavku na flexibilní alokaci zdrojů; předpokladem úspěšnosti paralelní práce výše uvedených rolí je vytvoření nové fáze *definice datové základny* před začátkem vlastního vývoje a fáze *integrace*, včetně integračních testů na konci vývoje.

3. *Striktně rozdělit a definovat odpovědnosti rolí za své výstupy* – hlavní splnění tohoto požadavku bylo směřováno ke kodérům a programátorům, kde každá role vytváří svoji vlastní část projektu, která musí být nezávisle na sobě funkční a lze ji ověřovat systémovými testy. Oddělení jak po stránce odpovědností tak paralelní práce je možné díky zavedení prvku XML kolektoru, který striktně odděluje aplikační a prezentační část aplikace.

4. *Vytvořit podmínky pro flexibilní stanovování priorit činností v rámci celého portfolia projektů i projektů samotných* – oddělení závislosti klíčových rolí a aktivit v projektu umožňuje zdroje převádět mezi jednotlivými projekty či aktivitami i během realizace dle nastavených priorit a časování.

5. *Zefektivnit proces údržby a rozvoje aplikací* – jasné oddělení prezentační a aplikační vrstvy, včetně zodpovědností za ně, umožňuje lépe adresovat změnový/rozvojový požadavek na konkrétní roli. V případě, že je požadavek směřován pouze do roviny aplikační nebo prezentační postačuje alokace pouze jedné role.

6. *Metodika musí vycházet ze standardizovaných, podporovaných a dlouhodobě rozvíjených formátů, aby byl zajištěn i její vlastní rozvoj* navržená metodika, pokud doporučuje konkrétní implementační vlastnosti, vychází výhradně ze standardizovaných formátů zaštiťovaných konsorciem W3C, případně formátů, které jsou převoditelné do některého ze standardizovaných formátů. Důraz byl kladen především na snadnost zaškolení, dostupnost materiálů, podpory a možnosti kombinace jednotlivých formátů v rámci jedné metodiky. Konkrétně se jedná o formáty XML, XSL a RelaxNG.

7. *Striktně oddělit prezentační a aplikační vrstvu*

   - ▷ *Oddělit posloupnosti zpracování v aplikačním kódu a uživatelském rozhraní* – protože data generovaná z aplikačního kódu jsou na výstupu jasně identifikována a zařazena do hierarchie XML dokumentu, je možné je v rámci aplikačního kódu generovat naprosto nezávisle na pořadí jejich zobrazení v uživatelském rozhraní a grafické reprezentaci. Naopak uživateli je možné zobrazit pouze některá data, a to nezávisle na pořadí, ve kterém se v XML dokumentu nachází.
   - ▷ *Odstranit prolínání použitých programovacích jazyků mezi jednotlivými vrstvami* – jak je zmíněno v předchozím bodě, jsou obě vrstvy odděleny aplikačně nezávislou transportní vrstvou a šablona je generována výlučně s použitím standardizovaného formátu XSL. Tyto dva předpoklady znemožňují nesystematické prolínání programovacích a značkovacích jazyků mezi sebou.
   - ▷ *Navrhnout metodu, kterou bude eliminováno vzájemné nesystémové prolínání vrstev* – jelikož v navržené metodice není pro zobrazení uživatelského rozhraní třeba aplikačního kódu, a protože je aplikační kód od prezentační vrstvy oddělen nově zavedenou transportní vrstvou, není možné přesouvat jakékoliv části aplikace na úroveň definice uživatelského rozhraní a naopak. V tomto případě se nejedná pouze o omezení na úrovni definice postupu práce, ale o omezení, které je nutné implementovat na úrovni technologie.

8. *Připravit podmínky pro outsourcing jednotlivých projektových rolí* – metodika nepřichází v tomto ohledu s jakoukoliv úpravou vztahů spolupráce na bázi outsourcingu. Externí dodávky jsou zpravidla zatíženy složitou komunikací a častým nedodržováním termínů dokončení. Díky oddělení závislostí jednotlivých etap vývoje, je možné v rozumné míře outsourcovat činnosti grafiků, kodérů a programátorů, přičemž případný dopad pozdější dodávky nemusí být ve výsledném čase kritický jako při konvenčním

způsobu vývoje. V ideálním případě by měly být outsourcingem pokryty pouze ty činnosti, které neleží na kritické cestě, která je v případě metodiky ETWA výrazně kratší.

9. *Podpořit specializaci týmových rolí* – každá role v projektu vytvářeného podle metodiky ETWA pracuje pouze s prostředky a znalostmi, které jsou primárním obsahem její specializace, protože se vzájemně neprolínají jazyky jednotlivých vrstev aplikace. Tento požadavek je splněn u rolí programátora a kodéra, kdy se programátor může více soustředit na vývoj aplikačního kódu a díky tomu jej efektivně rozvíjet, stejně tak znalosti kodéra jsou rozšířeny o další jazyky z rodiny SGML.

10. *Aplikací metodiky podpořit vlastnosti vyvíjeného produktu (např. podporou různých výstupních formátů, vícejazyčných aplikací, rozložením zátěže a zpracování)* – generování nejen uživatelského rozhraní, ale i jakéhokoliv aplikačního výstupu z XML dat umožňuje produkovat výstupy v libovolných formátech a to jen na základě výměny šablony na úrovni prezentační vrstvy – tj. bez nutnosti zásahu do aplikačního kódu. Skládání výstupů na základě vícejazyčných XML elementů, či konfiguračních prvků podporuje navrhovaný XML kolektor, který na základě těchto elementů připravuje data pro prezentační vrstvu, v níž je pak možné realizovat uživatelské rozhraní v různých jazycích na základě jedné konzistentní šablony. U aplikací či systémů, kde je rozložení zátěže kritickým faktorem, umožňuje metodika přesunout část úloh až na úroveň klienta, což má za výsledek nejen rozložení zátěže související s obsluhou uživatelského rozhraní, ale také jeho rychlejší odezvy a efektivnější práci s ním. Tento postup využívá ke své práci kodér při vývoji uživatelského rozhraní dle metodiky ETWA.

Metodika byla využita při implementaci konkrétního projektu internetového obchodu, kdy bylo nutné tomuto úkolu v souladu s metodikou přizpůsobit nejen vývojový proces, ale i používané prostředky vývoje. Vývoj na základě navrhované metodiky byl porovnán s vývojem jiného projektu internetového obchodu, který byl, co do rozsahu, ceny i časování, podobný. Výsledky obou projektů byly zhodnoceny a vzájemně porovnány. U konvenčně vyvíjeného projektu byly identifikovány možné příležitosti pro nasazení metodiky ETWA a také shrnuty výhody konvenčního přístupu. U projektu vyvíjeného dle metodiky ETWA byly zhodnoceny přínosy a rizika jejího použití. Tímto způsobem bylo ověřeno, že nová metodika je v praxi použitelná, a že její nasazení přináší pozitivní výsledky ve vývojovém procesu webové aplikace a odbourává překážky, které byly identifikovány v úvodu této práce jako výchozí stav pro návrh nové metodiky.

Literatura

[1] BATES, C. XML in Theory and Practice. West Sussex: Wiley, 2003. 469 s. ISBN 0-470-84344-6.

[2] BERNERS-LEE, T.; FISCHETTI, M. Weaving the Web: Origins and Future of the World Wide Web. Britain: Orion Business, 1999. ISBN 0-7528-2090-7.

[3] BRÁZA, J. Objekty v PHP5. *Interval* [online]. Publ. 2003-03-12. ISSN 1212-8651 [cit. 2008-10-30] <http://interval.cz/clanky/objekty-v-php5/>.

[4] CEJPEK, J. Informace, komunikace a myšlení: Úvod do informační vědy. 1 vyd. Praha: Karolinum, 1998. 179 s.

[5] DARIE, C.; BRINZAREA, B.; CHERECHES-TOSA, F.; BUCIA, M. AJAX a PHP - tvoříme interaktivní webové aplikace profesionálně. Brno: Zoner Software, 2006. 320 s. ISBN 80-86815-47-1.

[6] DRLÍK, Z. ViewState v ASP.NET aplikacích - implementace a použití. *Interval* [online]. Publ. 2004-05-13. ISSN 1212-8651 [cit. 2008-11-30] <http://interval.cz/clanky/viewstate-v-asp-net-aplikacich-implementace-a-pouziti/>.

[7] DUYNE, D., K.; LANDAY, J., A.; HONG, J., I. Návrh a tvorba webů. 1. vyd. Brno: CP Books, 2005. 672 s. ISBN 80-251-0508-3.

[8] EERNISSE, M. Build Your Own AJAX Web Applications. Melbourne: SitePoint Pty. Ltd., 2006. 319 s. ISBN 0–9758419–4–7.

[9] FEUERLICHT J., Enterprise Computing: Standards and Architectures. Praha: Oeconomica, 2008, 148 s. ISBN 978 80 245 1367 6.

[10] GÁLA, L.; POUR, J.; TOMAN, P. Podniková informatika. Praha: Grada Publishing, 2006. 484 s. ISBN 80-247-1278-4.

[11] GUCKENHEIMER, S. Software Engineering with Microsoft Visual Studio Team System. Boston: Addison Wesley Professional, 2006. 304 s. ISBN 0-321-27872-0.

[12] HAROLD, E. R.; MEANS, W. S. XML in a Nutshell. 2. vyd. Sebastopol : O'Reilly, 2002. 634 s. ISBN 0-596-00292-0.

[13] HLAVENKA, J. Využití internetu v Česku stagnuje? Naopak! *Živě* [online]. Publ. 2005-06-09. ISSN 1214-1887 [cit. 2008-12-15] <http://www.zive.cz/Clanky/Vyuziti-internetu-v-Cesku-stagnuje-Naopak/sc-3-a-125142/default.aspx>.

[14] HOMAN, D.; SANCHEZ, E.; KLIMA C. Building A Portal? *Vive La Différence* [online]. [cit. 2008-12-10] <http://www.informationweek.com/story/IWK20011101S0014>.

[15] HRDINA, L. Deset hlavních slabin webových aplikací. *SystemOnLine* [online]. Publ. 2005-02-01 [cit. 2008-11-15] <http://www.systemonline.cz/index.php?sec=casopis&id_clanek=1767>.

[16] CHATHAM, B. Integrated User Profiles Boost Web Analytics [online]. Publ. 2005-02-01 [cit. 2007-06-16] <http://www.forrester.com/Research/Document/Excerpt/0,7211,36329,00.html>.

[17] JONÁK, Z. Pojem „informace“ ve světě sdíleného pojetí skutečnosti. *Ikaros* [online]. 2000, č. 2. ISSN 1212-5075. [cit. 2008-11-10] <http://www.ikaros.cz/node/524>.

[18] KADLEC, V. Liší se vývoj pro web od vývoje neinternetových aplikací? [online]. Publ. 2003-05-06 [cit. 2008-11-30] <http://www.zive.cz/Clanky/Lisi-se-vyvoj-pro-web-od-vyvoje-neinternetovych-aplikaci/sc-3-a-111641/default.aspx>.

[19] KADLEC, V. Rational Unified Process: základní pojmy [online]. Publ. 2003-08-05 [cit. 2008-12-20] <http://www.zive.cz/Clanky/Rational-Unified-Process-zakladni-pojmy/sc-3-a-113011/default.aspx>.

[20] KAISER, S. Deliver First Class Web Sites: 101 Essential Checklists. Melbourne: SitePoint Pty. Ltd., 2006. 353 s. ISBN 0-9758419-0-4.

[21] KALUŽA, R. Vzor Model-View-Controller [online]. Publ. 2006-04-21 [cit. 2008-02-03] <http://radovan.bloger.cz/it/objektove-aplikace/vzor-Model-View-Controller>

[22] KARAPUDA, L. Standard Problems in Web Project Management [online]. Publ. 2005-05-04 [cit. 2008-03-10] <http://www.webassemblyline.com/blog/2005/05/web-project-management-is-your-business-not-development>.

[23] KOSEK, J. XML - šance pro třetí tisíciletí [online]. Publ. 1999 [cit. 2008-09-20] <http://www.kosek.cz/clanky/cw/xml.html>.

[24] KOSEK, J. XSLT v příkladech [online]. Publ. 2004 [cit. 2008-03-10] <http://www.kosek.cz/xml/xslt/>.

[25] KOTEK, J. Podnikové portály - náklady a přínosy. *SystemOnLine* [online]. Publ. 2004-09-01 [cit. 2008-10-11]

<http://www.systemonline.cz/clanky/podnikove-portaly-naklady-a-prinosy.htm>.

[26] KUČEROVÁ, H. Definice informace. Data - informace - znalosti [online]. Publ. 2002-03-20, posl. revize 2005-09-17 [cit. 2008-09-18]. <http://info.sks.cz/users/ku/UIS/inform1.htm>.

[27] KYSELICA, A.; MANA, M. ČSÚ: Výsledky šetření o využívání informačních a komunikačních technologií v domácnostech a mezi jednotlivci v roce 2004 [online]. Publ. 2005-07-18 [cit. 2008-08-04] <http://www.czso.cz/csu/2005edicniplan.nsf/p/9608-05>.

[28] MANA, M.; NOSÁL, P. ČSÚ: Výsledky šetření o využívání informačních a komunikačních technologií v domácnostech a mezi jednotlivci v roce 2007 [online]. Publ. 2007-10-23 [cit. 2008-08-04] <http://www.czso.cz/csu/2007edicniplan.nsf/p/9701-07>.

[29] MANA, M.; KOSINA, V. ČSÚ: Výsledky šetření o využívání informačních a komunikačních technologií v podnikatelském sektoru za rok 2004 [online]. Publ. 2006-02-10 [cit. 2008-08-05] <http://www.czso.cz/csu/2005edicniplan.nsf/p/9602-05>.

[30] MANHARTSBERGER, F. Portály se prosazují zkuste to s nimi. *PCWorld* [online]. [cit. 2007-04-12] <http://www.pcworld.cz/pcw.nsf/069b11ec6da3c04cc1256b04004e3f3c/27bbf48d03a7ad24c12569fa0000ba82?OpenDocument>.

[31] Microsoft: Bankovnictví online [online]. [cit. 2008-09-12] <http://www.microsoft.com/cze/athome/ security/privacy/banking.mspx>.

[32] MYER, T. No Nonsense XML Web Development With PHP. Melbourne: SitePoint Pty. Ltd., 2005. 374 s. ISBN 0-9752402-0-X.

[33] NetCraft: PHP Usage Stats [online]. [cit. 2008-09-27] <http://www.php.net/usage.php>.

[34] NetCraft: September 2005 Web Server Survey [online]. [cit. 2008-12-27] <http://news.netcraft.com/archives/2005/09/05/september_2005_web_server_survey.html>.

[35] NetCraft: November 2008 Web Server Survey [online]. [cit. 2008-12-27] <http://news.netcraft.com/archives/2008/11/19/november_2008_web_server_survey.html>.

[36] NGHIEM, A. IT Web Services - A Roadmap for the Entreprise. New Jersey: Prentice Hall, 2003. 307 s. ISBN 0-13-009719-5.

[37] NIELSEN, J. Web.Design. 1. vyd. Praha: SoftPress, 2002. 382 s. ISBN 80-86497-27-5.

[38] POLÁK, J.; MERUNKA, V.; CARDA, A. Umění systémového návrhu. Praha: Grada Publishing, 2003. 196 s. ISBN 80-247-0424-2.

|39| PROCHÁZKA, J. Nástroje CASE? Co? Proč? Jak? *Databázový svět* |online| Publ. 2004-05-27. ISSN 1213-5933 [cit. 2008-07-12] <http://www.dbsvet.cz/view.php?cisloclanku=2004052702>.

[40] PŮČEK M.; MATOCHOVÁ J. Řízení rizik a financí. Praha: Mepco, 2007. 85 s. ISBN 978-80-903960-0-5.

|41| SOUKENÍK, J.; PETRÁŇ, M. Jak pomoci projektu. *Business World* [online]. Publ. 2005-06-01. ISSN 1213-1709 [cit. 2008-03-10] <http://www.trask.cz/DeliverLive/digitalAssets/458_Jak%20pomoci%20projektu_BW_0606.pdf>.

|42| ŠLAPÁK, O. Data, informace, znalosti. *E-Logos - Electronic Journal For Philosophy* [online]. 2003, č. 2003. ISSN 1211-0442. [cit. 2008-06-12] <http://nb.vse.cz/kfil/elogos/miscellany/slapa103.pdf>.

[43] ŠORM, M. Seminář o Univerzitním informačním systému. 1. vyd. Brno: Mendelova zemědělská a lesnická univerzita v Brně, 2008. 111 s. ISBN 978-80-7375-171-5.

|44| ŠORM, M. Webové informační systémy. *Sborník konference MendelNet 2003.* Brno: MZLU v Brně, 2003. ISBN 80-7302-048-3.

|45| TALICH, M. Webové služby a aplikace XML. *INFORUM 2004: 10. konference o profesionálních informačních zdrojích* [online]. Publ. 2004-05-27 [cit. 2008-07-09] <http://www.inforum.cz/inforum2004/pdf/Talich_Milan1.pdf>.

[46] TIOBE Programming Community Index for November 2008 [online]. [cit. 2008-12-02] <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>.

|47| VODÁČEK, L., ROSICKÝ, A. Informační management, pojetí, poslání a aplikace. Praha: Management Press, 1997. 146 s. ISBN 80-85943-35-2.

[48] VOLDÁN, J. Zvláštnosti internetových projektů [online]. Publ. 2002-01-25 [cit. 2008-10-10] <http://www.park.cz/zvlastnosti_internetovych_projektu/>.

[49] WARD, S.; KROLL, P. Building Web Solutions with the Rational Unified Process |online|. Publ. 2001-01-04 |cit. 2008-07-20| <http://www.dcc.uchile.cl/~luguerre/cc61j/recursos/76.pdf>.

|50| Web Modeling Language |online|. |cit. 2008-07-02| <http://www.webml.org>.

[51] WILLIAMS, M. The Principles of Project Management. Melbourne: SitePoint Pty. Ltd., 2008. 224 s. ISBN 978-0-9802858-6-4.

|52| WOODS, D. Enterprise Services Architecture. Sebastopol: O'Reilly Media, 2003. 205 s. ISBN 0-596-00551-2.

|53| ZELDMAN, J. Designing with Web Standards. Indianapolis: New Riders, 2003. 436 s. ISBN 0-7357-1201-8.

Seznam vlastních publikací

1. ŠIMEK, P.; ČERMÁK, V. Implementace moderních metod při tvorbě internetového portálu. In Sborník prací z mezinárodní vědecké konference Agrární perspektivy XII.: ČZU v Praze, Provozně ekonomická fakulta, 2003, s. 919-924. ISBN 80-213-1056-1.

2. ČERMÁK, V.; ŠIMEK, P. Progressive Approach to Portal Concepts and Software. In Sborník příspěvků z konference Informační systémy v zemědělství a lesnictví: HELP SERVICE Education, Seč u Chrudimi, 2003, s. 56-57. ISBN 80-239-0270-9.

3. ČERMÁK, V. Distribuce a zpřístupňování informací. In Sborník příspěvků z doktorandského semináře: PEF ČZU v Praze, 2003, s. 20-29. ISBN 80-213-1016-2

4. ČERMÁK, V.; ŠIMEK, P. ASP model utilization. In Proceedings of Workshop Internet and Information Systems: PEF ČZU v Praze, 2003, s. 6-9. ISBN 80-213-1118-5

5. ČERMÁK, V. Netscape/Mozilla: Alternativa v prohlížečích. *Computerworld*, 2003, roč. 14, č 25, s. 22. ISSN 1210-9924.

6. ČERMÁK, V. Komunikační a výukový systém. In Sborník příspěvků z doktorandského semináře 2004: PEF ČZU v Praze, 17. 2. 2004, s. 51- 51. ISBN 80-213-1150-9.

7. HAVLÍČEK, Z.; ČERMÁK, V.; DVOŘÁK, M.; BRÁZDA, R.; HALBICH, Č.; SMÍTKA, M.; PEJSAROVÁ, S.; ŠIMEK, P. Internetové technologie I. Praha: Credit, 2004. ISBN 80-213-1109-6.

8. ČERMÁK, V.; ŠIMEK, P. Internet as a Medium for Digital Photography Sharing. In Internet and Informations Systems II.: Katedra informačních technologií, 21.9.2004, s. 56- 61. ISBN 80-213-1282-3.

9. ŠIMEK, P.; ČERMÁK, V. Uložení, distribuce a sdílení dat internetovými portály. In Sborník prací z mezinárodní vědecké konference Agrární perspektivy XIII (trvale udržitelný rozvoj agrárního sektoru - výzva a rizika): Česká zemědělská univerzita v Praze, Provozně ekonomická fakulta, 22. 9. 2004, s. 645- 650. ISBN 80-213-1190-8

10. ČERMÁK, V.; HAVLÍČEK, Z. Optimalizace vývojového procesu webových aplikací. *Systémová integrace*, duben 2009, roč. 16, č 1, s. 66-76. ISSN 1210-9479

Seznam obrázků

Seznam příkladů

Dodatek A

Hodnocení činnosti projektových týmů

Dobrý den,
pro účely sběru reprezentativních dat pro moji disertační práci na téma "Efektivní tvorba webových aplikací", bych vás chtěl požádat o odpovědi na několik, níže uvedených, otázek. Pevně věřím, že vám tato aktivita nezabere více jak 10 minut vašeho času. Výsledky nebudou nikde prezentovány ve spojení s vaším jménem ani nikterak jinak identifkovány. První otázka je proto dobrovolná a je pouze pro účely evidence vyplnění dozníků na mé straně.

Za vyplnění vám předem děkuji.
Vráťa Čermák

Osobní údaje

1. Vaše jméno a příjmení: _____
2. Vaše primární role při tvorbě projektu webové aplikace *
   - ▷ Analytik, Architekt
   - ▷ Designér
   - ▷ Kodér (HTML/CSS)
   - ▷ Programátor
   - ▷ Projektový manažer

Typizované problémy

Snažte se vybrat 3 nejdůležitější odpovědi na otázky z pohledu uvedené role. Nemusíte se omezovat pouze na odpovědi u vaší role. Můžete vyplnit i důvody u ostatních rolí, které si myslíte, že jsou důležité.

1. Při návrhu DESIGNU webové aplikace se nejčastěji potýkám s následujícími problémy
   (a) Není dokončena analýza a není se proto o co opřít
   (b) Jedná se o první fázi projektu a požadavky se často mění
   (c) Zákazník nemá grafické cítění
   (d) Musí být vytvořeno více zbytečných návrhů
   (e) Jsem omezen možnostmi webu (velikosti, kvalita obrázků, ...)
   (f) Nevyhovující podklady od zákazníka
   (g) Málo času na tvorbu designu
   (h) Pro svou práci nemám vhodné nástroje
2. Při tvorbě layoutu HTML/CSS webové aplikace se nejčastěji potýkám s následujícími problémy
   (a) Grafika odporuje pravidlům webu
   (b) V grafice je něco jiného než v analýze (co je požadováno zákazníkem)
   (c) Zbytečná tvorba prototypů (klikací šablony)
   (d) Málo času na přípravu šablon
   (e) Pro svou práci nemám vhodné nástroje
   (f) Do kódu HTML jsou vkládány značky programovacího jazyka, které znemožňují jeho následnou úpravu
   (g) Pravidla SEO a prohlížečové optimalizace jsou porušována programátorem
   (h) Moje účast je vyžadována po celou dobu projektu
   (i) Neznám potřeby programátora, abych jim mohl layout uzpůsobit
   (j) Po spuštění webu obsahuje stránka spoustu kódu v jiném jazyce než je HTML
3. Při PROGRAMOVÁNÍ webové aplikace se nejčastěji potýkám s následujícími problémy
   (a) Nedostatečné, neodpovídající zadání
   (b) Špatně připravené výstupy předchozích fází (HTML šablony, příp. design)
   (c) Špatná orientace v HTML, obtížná integrace s výstupy programu
   (d) Obtížné udržet kód čistý od specifik layoutu (HTML)
   (e) Nutná častá interakce s kodérem za účelem relizace požadovaného vzhledu, funkčnosti

(f) Používané pracovní postupy, framework

(g) Pozdní zapojení do projektu, kdy už nemohu vznést své připomínky

(h) Musím realizovat sám nedodělky, chyby z předchozích fází

4. Při ŘÍZENÍ PROJEKTU webové aplikace se nejčastěji potýkám s následujícími problémy

(a) Zdroje nejsou dostupné v dohodnutém čase

(b) Omezená možnost záměny pořadí fází, záměně činností v čase

(c) Neustálé změny zadání prováděné zákazníkem

(d) Obtížné nalezení role odpovědné za opravy zjištěných chyb

(e) Prodloužení jedné fáze automaticky prodlužuje celou dodávku

(f) Obtížný souběžný vývoj více rolí, nebo více součástí projektu najednou

(g) Obtížná dohoda s členy projektového týmu

(h) Obtížná realizace formou iterací (sady menších dodávek)

(i) Omezené možnosti outsourcingu rolí

Netypizované problémy

Jestliže jste v předchozích problémech něco nenašli, co považujete za problém, zadejte ho do této sekce. Co mi brání v tom, abych mohl dělat svoji práci efektivněji, kde mě v mé práci blokují ostatní členové projektového týmu, co mi nevyhovuje na současných pracovních postupech, apod.

Děkuji za váš čas a vyplnění tohoto dotazníku.

Dotazník je možné vyplnit prostřednictvím služby Wufoo.com na adrese:
`http://kremik.wufoo.com/forms/dotaznak-disertaaena-prace/`