Vývoj výzkumu softwarových procesů za posledních 15 let

Semestrální práce ke kurzu 4IT421 Zlepšování procesů budování IS	
Semestr	ZS 2016/17
Autoři – jméno, příjmení , xname	Katarína Bartošová (xbark39) Jiří Pangrác (xpanj21) Markéta Nováková (novm29) Ondřej Bína (bino00)
Téma	Software Process research and practice in the past 15 years.
Datum odevzdání	18.12.2016

Abstrakt

Cílem práce je shrnout vývoj SW procesů za posledních 15 let, shrnutí zprávy z konference FOSE 2000, jejích připomínek a zjištění a také načrtnout možné budoucí trendy. Práce vychází hlavně ze závěrů práce nazvané Software Process od A. Fuggetty a E. Di Nitto. První kapitola se věnuje softwarovým procesům do roku 2000. Druhá kapitola popisuje jejich vývoj za posledních 10 let. Třetí kapitola je věnována hlavním budoucím trendům a výzvám. Čtvrtá kapitola rozebírá problémy a možné směry výzkumu. Práce tedy poskytuje přehled o současném stavu vývoje výzkumu softwarových procesů, problémy a výzvy budoucnosti.

Klíčová slova

Vývoj výzkumu softwarových procesů, trendy softwarových procesů

Obsah

Úvod

V dnešním světě je software velmi důležitý. Zasahuje do všech oblastí našeho života, od práce, přes osobní aktivity, zábavu až po vzdělání a politiku. Veškeré občanské a průmyslové infrastruktury jsou taktéž dnes postaveny na softwaru. Jeho vývoj je proto velmi kritická aktivita – musí se pochopit, vylepšovat, podporovat. Během posledních 50 let bylo cílem výzkumu softwarových procesů studovat tyto aktivity a vypořádat se s problémy, které při vývoji softwaru mohou nastat.

Výzkum softwarových procesů se tedy zaměřil právě na pochopení, popis, hodnocení, automatizaci a vylepšení činností, postupů a technik požívaných ke zvládnutí tohoto složitého úkolu. Výzkumy přinesly mnoho zajímavých výsledků. Problémem ale bylo jejich pochopení a využití – některé byly špatně formulovány a přeceňovány, jiné naopak přehlíženy. A mnoho jich je stále nevyřešeno.

Samotný termín softwarové procesy získal pozornost v 80. letech. V té době ovšem patřilo pod softwarové procesy mnoho projektů v mnoha oblastech – modelování, automatizace, vylepšení. V roce 2000, s příchodem nového tisíciletí, bylo jasné, že je potřeba se zamyslet nad stavem v komunitě výzkumu softwarových procesů. Bylo třeba shrnout dosavadní úspěchy, chyby a výzvy.

Na tuto potřebu reagovala konference ICSE (International konference on Software Engineering) v roce 2000. Na ní začala vznikat práce, která byla později téhož roku prezentována na další z konferencí, konkrétně a konferenci FOSE (Future of Software Engineering).

Nicméně, od té doby došlo opět k mnoha změnám – stále se rozšiřující internet, mobilní zařízení, vznikly nové trhy a oblasti, jako například sociální sítě. Je potřeba změnit přístup k vývoji softwaru, který byl výrazně poznamenán příchodem open source softwaru, agilních metodik a mobilních aplikací… Je tedy potřeba znovu zvážit stav komunity okolo výzkumu softwarových procesů, změnit způsob, jak se vypořádat s problémy a příležitostmi, které byly identifikovány v roce 2000 a zároveň opět zjistit a analyzovat, jak se vyrovnává s problémy v posledních 15 letech.

Cílem této práce je jednak shrnutí zprávy z konference FOSE 2000, jejích připomínek a zjištění. Dále se práce bude zabývat přehledem hlavních směrů ve výzkumu softwarových procesů za posledních 15 let. Na závěr pak práce adresuje problémy, výzvy výzkumu softwarových procesů a nastíní jeho další směřování.

Celá práce vychází z práce nazvané Software Process od A. Fuggetty a E. Di Nitto (FUGGETTA, a iní, 2014).

Vývoj výzkumu softwarových procesů do roku 2000

Rok 2000 byl důležitým milníkem ve výzkumu softwarových procesů, neustále rostl pocit, že se výzkum dostal do slepé uličky. Výzkumníci si kladli otázku, zda jsou výsledky výzkumů schopny pojmenovat a ovlivnit problémy, které byly zjištěny. Zdálo se totiž, že jsou to často je jakási „cvičení", neschopná konkrétně ovlivnit způsob, jakým je software vyvíjen, nasazován a používán v reálném světě.

Proto byla na konferenci FOSE v roce 2000 prezentována práce, která měla zhodnotit dosavadní dopad výzkumu softwarových procesů. V této práci se objevily následující 4 hlavní oblasti zkoumání:

1) **Modelování procesů a podpora**
   Softwarový proces je složitý proces zahrnující profesionály, organizace, firemní politiku, nástroje a podpůrné prostředí. Hlavní oblastí výzkumu bylo vytvoření jazyků a prostředí pro popis a modelování tak komplexních procesů. Tyto jazyky musely být hodně přesné, aby dokázaly postihnout složitost procesu a poskytovaly automatickou podporu developerům a manažerům. Vzniklo tak hodně procesních modelovacích jazyků (PML), které byly navzájem nekonzistentní a pro manažery špatně uchopitelné.

2) **Vylepšení procesů**
   Vylepšení procesů je druhá nejvýznamnější oblast výzkumu. Vývoj softwaru je totiž dynamický, nemůže stát na místě. Je tedy důležité identifikovat metody a přístupy pro zjištění zralosti procesů a najít způsob, jak procesy vylepšit.

3) **Metriky a empirická měření**
   Jak již bylo řečeno, vývoj softwaru je složitá aktivita, do které vstupuje mnoho aspektů. Během 90. let se výzkumníci proto zaměřili na empirické studie a identifikaci metrik pro posouzení výkonu softwarových procesů.

4) **„Skutečné" procesy**
   Na základě zkušeností a výsledků výzkumů bylo vytipováno několik rámců pro vývoj softwaru – vznikl takzvaný Unified Process a Personal Software Process.

V těchto čtyřech oblastech proběhlo mnoho výzkumů, proběhly pokusy posoudit kvalitu, vývoj. Práce z konference FOSE prozkoumala mnoho prací a vypracovala několik úvah a kritik dosud provedených výzkumů.

1) **Softwarové procesy jsou také procesy**
   Tato námitka vychází z myšlenky, že všechny procesy jsou si podobné. Samozřejmě existuje několik odlišností softwarových procesů od ostatních, ale společných rysů je mnohem více.

2) **Účel jazyků a technologií procesů se musí změnit**
   Řada jazyků z 90. let byla složitá a těžkopádná a bylo nemožné je použít v reálném nasazení. Zároveň je potřeba poskytnout prostředky pro automatizaci.

3) **Empirické studie jsou prostředkem, ne cílem**
   Jak už bylo výše řečeno, některé studie se staly pouhým cvičením, bez reálného dopadu. To bylo potřeba do budoucna změnit.

4) **Vylepšení softwarových procesů je také vylepšení**
   Analogicky k první námitce, není účinné, aby se ke zlepšování softwarových procesů přistupovalo zcela od nuly, když jsou zde určité podobnosti s ostatními procesy.

Práce z konference FOSE měla velký dopad na komunitu výzkumu softwarových procesů. Byla totiž založena na poctivé analýze výsledků z předchozích let. Výzkumníci došli k závěru, že výzkum softwarových procesů potřebuje změnu.

Softwarové procesy za posledních 10 let

Na základě práce z FOSE 2000 se tedy výzkum softwarových procesů, a obecně softwarové procesy, změnily. Problémy, které byly v práci adresovány, byly částečně vyřešeny. Hlavně se začalo k softwarovým procesům přistupovat jako k multidisciplinární oblasti a jako taková by měla zahrnovat výsledky z dalších oblastí a disciplín. Došlo také k přepracování modelu CMM z roku 1990, který se změnil na CMMI. Model zralosti CMMI obsahuje jemnější rozdělení pro měření výkonnosti organizace a zahrnuje specifické procesní oblasti). V další části budou přiblíženy směry, kterými se softwarové procesy a jejich výzkum vydaly.

2.1 Společenský aspekt

Společnost má při vývoji softwaru čím dál větší význam. V současné době je vývoj softwaru považován za socio-technický systém – lidské aspekty jsou také důležité a musí být podporovány technologií.

Výraz socio-technický systém poprvé použili pánové Trist, Bumforth a Emery v 50. letech, když zkoumali práci horníků v dolech (TRIST, a iní, 1951). Zjistili, že nasazení nových technologií nutně neznamená zlepšení výkonu. To je výsledkem až správné souhry společenských hledisek a technologie. Jinými slovy – nelze tato dvě hlediska sledovat odděleně, ale ve vzájemné synergii.

Do oblasti softwaru potom tento výraz převedl pan Cataldo, kdy definoval „Socio-technickou kongruenci“ jako „způsob, jak definovat shodu mezi formální a skutečnou spoluprací.“ Kongruence má podle něj dvě hlediska (CATALDO, a iní, 2008):

a) požadavky spolupráce určené technickým rozměrem socio-technického systému
b) spolupráce vykonávána (vyžadována) na základě organizace ve firmě

Proti tomuto vnímání stojí Blake, který říká, že je důležitá včasná identifikace toho, jak má být práce řízena a koordinována.

2.2 Agilní procesy

Význam společenských aspektů, který byl diskutován v předchozí kapitole, stál za vnikem takzvaného Agilního manifestu. Ten má 4 základní myšlenky:

1) Vývoj se soustředí na produkt
2) Důležitým hlediskem úspěchu vývoje je lidské chování
3) Základem moderního vývoje softwaru je přírůstkový a spirálovitý přístup
4) Je důležité rychle reagovat na změnu požadavků, v jakémkoliv stadiu vývoje

Z agilního manifestu se vyvinulo několik přístupů a metod vývoje softwaru. Nejvýznamnějším je Scrum, který je založen na sprintech. Sprinty trvají maximálně 4 týdny, během kterých se řeší jen určitá část funkcí. V rámci každého sprintu probíhají každý den porady – takzvané stand-up meetingy. Během těchto krátkých porad se zjistí, co kdo udělal, co bude dělat, prodiskutují se případné problémy. Změny ve vývoji se řeší až po konci každého sprintu. Během sprintu dochází nejen k vývoji funkcí, ale i k jejich kontrole a zdokumentování.

Mezi společnosti, které úspěšně implementovaly Scrum se řadí například Google, kde se po nasazení zlepšil dohled nad vývojem, jsou stanovovány realističtější deadliny a jsou pečlivě vybírány funkce, které se budou zrovna vyvíjet. Dalším příkladem je firma Systematic Software Engineering, která i vyvrací jednu z kritik agilních přístupů – že nelze agilní metody implementovat ve velkých společnostech. Systematic Software Engineering měla totiž při úspěšném zavedení Scrumu úroveň 5

podle modelu CMMI – byla to tedy organizace s optimalizovaným přístupem k procesům a přesto byla jejich cesta ke Scrumu úspěšná.

2.3 Globální softwarové inženýrství (GSE)

Při rozšiřování vývojářských týmů po celém světě je potřeba koordinace a agility ještě naléhavější. Motivace pro rozdělení vývojářských týmů po celém světě přesahuje samotný vývoj softwaru. Hlavním důvodem je většinou cena, kdy se sníží náklady přesunutím pobočky do levnějších oblastí, nebo do oblastí, kde bude tato pobočka blíže zákazníkům. Výzkumníci samozřejmě pochopili význam GSE praktik a jejich podpory a analyzují vznikající problémy. Vznikají práce na téma použitelnosti Scrumu v globálním softwarovém inženýrství nebo vliv architektury organizace na vývoj softwaru. Nicméně stále není možné udělat jednoznačný závěr ohledně globálního softwarového inženýrství. Obecným předpokladem nyní je, že úspěch záleží na vzájemných vztazích a důvěře mezi developery. Ale to je velmi obecný výsledek, který bude jistě potřeba do budoucna více analyzovat a zpřesnit.

2.4 Empirický výzkum v SW procesech

Softwarové inženýrství není jenom o technických řešeních. Empirický výzkum se v posledním desetiletí posunul v kvalitě používaných metod i externí validity výsledků. Soustředil se hlavně na charakterizaci a metodiky hodnocení použitých v empirických studiích. Autoři Empirical Methods and Studies in Softwate Engeneering motivují k použití empirických metod. Podle nich jsou důležité k plně podloženému rozhodnutí. (WOHLIN, a iní, 2003)

Mnoho probíhajících empirických studií se snaží poukázat na výhody nových přístupů či dokázat validitu zkoumaných hypotéz. Zatímco ty starší byly negativně ovlivněny nemožností replikace, na ty současné vplývá množství dat a pozorovacích uhlů nebo velký růst open sourcových projektů. To umožnilo širší aplikaci empirických analýz a znatelnější výsledky ve zlepšování porovnatelnosti podobně zaměřených studií.

2.5 Modelem řízené inženýrství (MDE)

Modelem řízené inženýrství (Model-Driven Engineering)je specifický přístup k softwarovému vývoji, který se při budování softwaru soustředí na použití modelů jako archetypů, jinak řečeno pravzorů. Zvýšila se tím sledovatelnost od požadavků ke kódu, možnost automatické generace kódu pro různé platformy a analýzou modelů předpovídat schopnosti finálního systému.

Nejznámějším příkladem uplatňujícím tento přístup je Model-Driven Architecture (MDA). První verze (1.0.1) byla vydána v roce 2003, ta další v roce 2005. Žádná novější zatím vydána nebyla, práce na MDA stále pokračuje. O tom svědčí různé konference a workshopy, z nich nejznámější je MODELS 2013.

MDA identifikuje tři různé pohledy na systém:

- Výpočetně nezávislý model (Computation Independent Model – CIM),
- Platformě nezávislý model (Platform Independent Model – PIM),
- Platformě specifický model (Platform Specific Model – PSM).

CIM reprezentuje koncept specifický pro daný business. Na téhle úrovni se popisují požadavky, architektura podniku a business modely. PIM abstrahuje od platformě specifických prvků a řeší, jak systém funguje. Vypracovává se architektonický model popisující obchodní logiku aplikace. PSM se zaobírá aspekty specifických platforem, na kterých by měl být software implementován. Hlavní myšlenkou MDA je oddělit aplikační a business logiku od technický platformy. PIM model

může být pomoci transformačních pravidel automaticky transformovaný do PSM modelu vhodného na přímé generování kódu.

Plné přijetí tohoto přístupu v průmyslu je stále limitováno, existují však některé zajímavé studie. Vyniká např. práce autorů Briand a další z MODELS 2012, která poskytuje užitečné rady. Říká, jak zvýšit popularitu modelem řízeného přístupu v reálném průmyslovém kontextu. Popisuje tři specifické případy, které však nebyly dosažené v rámci IT firem, ale organizací zaměřených na námořnictví a energetický sektor. Jedna ukazuje adaptaci tohoto přístupu k podpoře softwarové bezpečnosti certifikačních procesů s pomocí zajištění sledovatelnosti bezpečnostních požadavků. Její cílem je poukázat na to, jak může spolupráce výzkumníku zlepšit výsledky výzkumů. (BRIAND, 2012) Zajímavým je taky další případ podle Whittle a další. Podle ní zvýšení přijetí přístupu souvisí s důležitostí zavedení procesů uvnitř firem. Velký význam má použití odpovídajících nástrojů pro lidi, kteří je budou používat.

2.6 Správa životního cyklu aplikace (ALM)

Správa životního cyklu aplikace (Application Lifecycle Management) je třída produktů částečně z konvenčního softwarového průmyslu i z open source komunity. I když spojení s výzkumem softwarových procesů není na první pohled vidět, nabízí mechanismy pro automatizaci některých úkolů (typicky vývoj a testování) a taky pro propojení manažerských úkolů s aktivitami vývoje softwaru. Tady jsou užitečné konektory na externí nástroje pro specifickou funkcionalitu jako řízení konfigurace, sledování chyb, řízení požadavků atd. Příkladem AML nástrojů jsou Microsoft Visual Studio Application Lifecycle Management, IBM Rational solutions pro CLM (Collaborative Lifecycle Management) a open sourcový MyLyn integrován v Eclipse, který byl postupně vyvinut do komerčního Tasktop.

2.7 Automatizace a její vliv

V současnosti se nejdůležitější aplikace pro automatizaci softwarových procesů soustředí na podporu finálních fází vývoje softwaru.

2.7.1 Správa verzí

Správa verzí byla oživena novými nástroji, přičemž nejvýraznějším je git a „software forges“. Git je open sourcový systém zvládající malé i velké projekty s důrazem na rychlost a efektivitu. [1] „Softwarové výhně“ jsou platformy pro spolupráci, typicky dostupné jako SaaS. Tyto platformy jsou užitečné při organizaci a zpřístupnění informací o vyvíjeném softwaru pro vývojáře i externí pozorovatele. Díky tímhle nástrojům se správa verzí jeví lehčí na instalaci a provoz. Projektové týmy tyhle služby denně používají pro zprávu software, diskuzi a sdílení informací v týmu.

2.7.2 Zajištění kvality

Nástroje a procesy pro podporu automatizace v oblasti zajištění kvality dosáhly vysoký stupeň použitelnosti a jsou více používány vývojovými týmy i organizacemi.

1. Vývojáři používají nástroje pro analýzu zdrojového kódu, které poskytují metriky a indikátory kvality vyvíjeného softwaru. Příkladem je SonarQube.
2. Podpora jednotkového a integračního testování je v současné době z velké části automatizována díky XUnit a Selenium. Selenium podporuje tvorbu automatizovaných testovacích případů pro webové aplikace. Testovací skripty se nahrávají z akcí uživatele na daném webovém rozhraní.
3. Pro automatizaci akceptačního testování můžou být použity Easyb či Fitnesse. Umožňují přesné vyjádření požadavků a automatizaci jejich vyřízení. Easyb používa doménově specifický jazyk

[1] Viz. www.git-scm.com (12.12.2016)

(Domain Specific Language (DSL)) umožňující asociaci scénářů s provedením specifických kousků kódu. Požadavky ve Fitnesse jsou vyjádřeny v tabulkách, které rozsáhle definují korespondenci mezi vstupy a souvisejícími výstupy.

4. Testovací proces jako celek je taky možné automatizovat. STAF framework na definici testů nabízí kompletní textový (založený na XML) skriptovací jazyk. Umožňuje namodelovat kdy a které testy mají byt spuštěny a za jakých podmínek.

2.7.3 Budování SW

Automatizace procesů a aktivit je možná na různých úrovních vývoje softwaru – při budování IS, nasazování či provozu systému.

Použitím skriptu Maven může každý uživatel jednoduše ze správy verzí zjistit verzi softwaru, automaticky stáhnout potřebné externí komponenty a knihovny nebo se připravit na nasazení finálního produktu.

Různé frameworky, jako například Jenkins, podporují kontinuální iteraci. Ten s pomocí integrace výše zmíněných nástrojů uzavírá cyklus vývoje, integrace, kontroly kvality a budování. Poskytuje tak prostředí, které je možné nakonfigurovat na automatickou generaci produktu připraveného na nasazení, jakmile je ve správě verzí dostupná nová verze zdrojového kódu. Zvyšuje tak povědomí vývojového týmu o dopadech vývojářské aktivity.

Automatizace má roli i v provozu systému při správě aplikace a souvisejícího software stack[2]. Tento druh automatizace je možný díky virtualizaci, vzdálenému ovládání či rostoucí popularitě cloudu. Puppet podporuje správu aplikací a Nagios je používán na monitorování stavu aplikace.

2.7.4 DevOps

Zvyšování automatizace umožňuje vývojářům naplnit požadavky na podnikové aplikace týkající se dostupnosti, spolehlivosti a doby odezvy. Také vytváří spojení mezi vývojem a provozem, jinak nazývaný DevOps. DevOps výrazně mění organizaci IT firem. Například Amazon volí „per service“ organizaci, při které za vývoj a provoz množiny služeb jsou zodpovědné mezi-funkční (cross-functional) týmy. Pro posouzení výkonnosti těchto týmů jsou používány metriky soustředící se na měření kvality a stability konečných služeb, spíše než na tradiční měření produktivity navrhového výzkumem v softwarovém inženýrství.

[2] Software stack – skupina programů, či sada aplikací, společně pracujících na výsledku. Zde mohou být zahrnuty instalované programy a software definice produktu či patche. Např. Linuxový software stack je LAMP (Linux, Apache, MYSQL, Perl/PHP/Python), u Windows je to WINS (Windows Server, Internet Explorer, .NET, SQL Server). https://www.techopedia.com/definition/27268/software-stack (12.12.2016)

Hlavní trendy a výzvy

3.1 Internet jako „vývojové prostředí“

Veškerá aktivita během vývoje SW je v dnešní době vykonávána v rámci Internetu. To platí i pro menší vývojové projekty.

Pracujeme a spolupracujeme výhradně skrze Internet napříč jednou společností, mezi několika různými organizacemi nebo i s jednotlivci.

Pojmy jako jsou „vývojové prostředí“ nebo „vývojový tým“ se radikálně a dramaticky změnily. Tím, že vývoj je čím dál více závislý na interakci, integraci a spolupráci mezi samotnými vývojáři nebo mezi vývojáři a koncovými uživateli pracujícími v kontextu Networked Software Development, je jen zřídka software vyvíjen „izolovaně“. To vede ke změnám v metodách a technikách používaných v designu, vývoji, testování, nasazování a rozvoji softwaru. Jednou z takových změn je například tzv. „Crowdsourcing“.

Software je průběžné měněn a znovunasazován na základě zákazníkových požadavků. To klade velké nároky na klasické procesy jako jsou konfigurační management, nasazování softwaru a jeho provoz. Prostředí a podpůrné technologie musí pracovat na úrovni celého Internetu.

Dnes je pro nějaké druhy softwaru důležitější dostat se co nejdříve na trh než jejich vysoká kvalita a spolehlivost, jako jsou aplikace (z největší části mobilní aplikace) používané v rámci několika málo týdnů nebo dokonce jen pro určitou událost. Pro jiné je zase spolehlivost zásadní, protože musí, kvůli svému určení, splňovat kritické bezpečnostní nároky. Životnost takových aplikací může být desítky let a musí být průběžné rozvíjeny a integrovány, aby držely krok s novými technologiemi a službami dostupnými v rámci Internetu. Kvůli takovým rozdílným použitím softwaru je potřeba rozšiřovat a přizpůsobovat existující standardy a modely na různé situace a použití.

3.2 Internet jako architektonická a výpočetní infrastruktura

Většina moderních systému je vytvořena rozdělením a kombinováním distribuovaných komponent, které spolu komunikují napříč Internetem. Toto tvrzení jistě platí pro webové a mobilní aplikace, kde Internet umožňuje mixování aplikací a vytváření široce distribuovaných klient-server nebo p2p systémů. Avšak čím dál více platí i pro ostatní aplikace, že přímo či nepřímo operují nebo jsou integrované skrze Internet. Dokonce kontrolní a průmyslové systémy více či méně komunikují s běžnými informačními systémy a jsou spravovány a ovládány pomocí Internetu. To znamená, že klasické rozdíly mezi různými druhy softwaru mají tendenci mizet nebo se stávají složitějšími na formulaci a pochopení.

V posledních 10 letech se objevilo několik pojmů, které tento trend zdůrazňují a podporují:

1. **IoT (Internet Of Things)**

   Jakýkoliv produkt (věc) se stává svým způsobem „inteligentní“, tím, že má výpočetní a komunikační schopnosti. Trendem jsou například inteligentní domácí spotřebiče ovladatelné skrze Internet.

2. **Chytré služby**

   „Chytré“ služby jsou vlastně běžné služby interagující s inteligentními objekty. Příkladem jsou automobily jejichž propojení umožňuje nové formy pojištění a asistenčních služeb založených na pay-as-you-go přístupu (průběžných platbách).

3. **Mizející počítače**

Výpočetní zařízení mají tendenci se vytrácet, protože dnes již nejsou spojovány jen s konvenčními stolními PC nebo notebooky.

„Cloud computing“ je jedním z dalších přístupů, který přiživuje fakt, že Internet je infrastruktura pro vývoj a provoz moderních softwarových systémů. Je založen na virtualizaci různých druhů služeb a infrastruktur v Internetu:

1. **Infrastructure as a Service (Infrastruktura jako služba)**

   Poskytování virtuálních zařízení (serverů) a jiných prvků infrastruktury.

2. **Platform as a Service (Platforma jako služba)**

   Poskytování databázových úložišť, webových serverů a dalších prostředí pro běh aplikací.

3. **Software as a Service (Software jako služba)**

   Poskytování aplikací jako vzdálených služeb, např. Cloud Desktop.

Tento přístup podporuje organizování systémů jako kombinaci vzdálených, distribuovaných a lokálních komponentů.

3.3 Uživatelé jsou mobilní, často cestují a jsou pořád připojeni

Pokroky technologií v posledním desetiletí radikálně změnily trh s IT a také chování a přístup uživatelů. Zejména příchod chytrých telefonů, tabletů, senzorů a jiných inteligentních zařízení způsobil oproštění se od soustředění na klasické stolní a podnikové počítače a zaměření se na mobilní zařízení. Tato změna s sebou přináší množství kritických a specifických výzev pro vývojáře softwaru.

Práce s mobilními zařízeními je odlišná kvůli jejich rozměrům a možnostem jejich ovládání. Proto je třeba přemýšlet úplně jinak nad uživatelským rozhraním. Mobilní software musí fungovat s ohledem na lišící se spolehlivost připojení k Internetu, na kterém může být závislá.

Jedním z největších problémů u mobilních zařízení je spotřeba energie. Ta není spojena pouze s hardwarovými komponenty nebo bateriemi. Designéři softwaru si jsou čím dál více vědomi, že je třeba software vyvíjet tak, aby minimalizovali používání hardwarových a komunikačních prostředků zařízení, a tím snížili spotřebu energie. Právě tuto oblast je v dnešní době třeba zkoumat z hlediska softwarového inženýrství a softwarových procesů.

Design softwaru pro mobilní zařízení není jen variací klasických procesů vývoje, je třeba nových a specifických technik, zásad a metod k efektivnímu pokrytí všech novinek, které tato zařízení nabízejí.

3.4 Internet jako základní infrastruktura pro obchod

V posledních 10 letech se také radikálně změnil způsob distribuce a komercializace softwaru. To je zapříčiněno dvěma základními důvody: 1) zařízení jsou neustále připojená k Internetu 2) software lze jednoduše distribuovat, instalovat a konfigurovat skrze Internet.

Aktualizace softwaru mohou být prováděny častěji. Vývojáři mohou, hned jak objeví chybu, aktualizovat aplikaci pomocí Internetu. Uživatelé a zákazníci proto očekávají včasné aktualizace, které řeší závady nebo problémy aplikace.

Vývoj softwaru není už omezen národními či místními trhy. Proto je třeba aby software byl schopný pracovat ve více jazycích a při tom dodržovat specifické požadavky a omezení každého regionu.

Na distribuci softwaru byly aplikovány prvky e-komerce. Protože je software, už ze své podstaty, „virtuální věcí“, bylo velmi jednoduché nahradit jeho obvyklé distribuční kanály a obchody těmi virtuálními.

Tyto trendy byly důvodem vzniku jiné důležité inovace, a to takzvaných „app stores“. Zpočátku byly navrženy tak, aby zjednodušily nákup aplikací na mobilních zařízeních. Prvními příklady byly virtuální obchody vyvinuty pro Windows Mobile a OS Symbian v první polovině minulého desetiletí.

Nicméně, příchod iPhone App Store přinesl radikální změny, je to jediný způsob jak legálně prodávat a instalovat software na mobilních zařízení od firmy Apple. Touto cestou App Store představuje nový pohled na distribuci softwaru, vytváří ekosystém ve kterém výrobce hardwaru/softwaru kontroluje platformu/operační systém a procesy používané k distribuci a prodeji jakéhokoliv jiného softwaru.

V posledních letech vznikly další obchody a ekosystémy vytvořené podle stejného schématu jako Apple App Store. Mezi nejpopulárnější patří Mac Store (OSX), Google Play (Android) a Windows Store (Windows). Ve směs tyto obchody definují úplně nový trh se softwarem, nejen proto, že mění procesy, skrz které je distribuován a instalován, ale i proto, že díky nim vznikly nové business modely (např. „in-app“ nákupy) a narušily stávající obchodní přístupy.

Celkovým efektem je příchod mnoha nových (často malých) vývojářů, neuvěřitelný nárůst počtu prodaných aplikací a vytvoření zcela odlišných ekonomik a profilů příjmu.

Problémy a směrování výzkumu

Problémy ve vývoji softwaru jsou mimořádně složité a rychle se rozvíjející. Z tohoto důvodu je nezbytné, aby byl výzkum (vývoj) neustále přehodnocován a směrován. To samozřejmě není vůbec jednoduché. Rizikem je, že ta skutečně důležitá témata, ve kterých nejčastěji dochází k problémům, nebudou rozpoznána a budou přehlédnuta. V této kapitole shrneme některá z těchto témat, která jsou stále důležitější a chybí na ně přesvědčivé odpovědi.

4.1 Obecný rámec je lepší než přesný návod

Vývoj softwaru je proces zaměřený na lidi, ve kterém tvořivost a samostatnost hrají klíčovou roli. V důsledku toho většina činností vyskytujících se v procesu softwaru nemůže být pevně automatizována. Typicky jsou dány nějaké hranice a cíle, které omezují a řídí práci vývojářů softwaru, ale je nemožné přinutit je dodržovat přísné a předdefinované vzory a postupy. Jistě, existují specifické činnosti, při kterých předpis a automatizace jsou užitečné, například distribuované řízení konfigurace a nasazení softwaru. Jedná se totiž o opakující se a k chybám náchylné činnosti, a tudíž mohou mít užitek z vysoce automatizovaného prostředí. Ale to je jen podmnožina z aktivit, z kterých se vývoj softwaru skládá.

Obecně platí, že vývoj softwaru je složitý proces, ve kterém není vhodné vymáhat krok za krokem soulad s tuhým předdefinovaným modelem. Ve vývoji softwaru jsou velice časté výjimky v konzistenci a nejednotnost. Z tohoto důvodu je nemožné lpět na (často přísně) definovaných pravidel a omezeních. Mnohem důležitější je, aby bylo zajištěno, že proces jako celek "konverguje" směrem k žádoucímu výsledku, a že bude tolerovat, kontrolovat a odhalovat nesrovnalosti, které se vyskytnou. Tato "inteligentní" konvergence je mnohem důležitější než přísné dodržování pravidel.

Přístupy Agilního vývoje se pohybují podél této linie, zdůrazňující schopnost řešit neustálé změny v požadavcích a provádět požadované architektonické refaktorování. Bohužel, i když je vodopádový model obvykle prezentován jako historický relikt dávných dob a na trh bylo uvedeno mnoho účinných agilních metod, až příliš často jsou v reálném vývoji stále prováděny aktivity podle nepružných a poměrně sekvenčních procesů. Ve skutečnosti je velice důležité zkoumat nové přístupy, metody řízení, metody návrhu, a také technická řešení, aby se inteligentní konvergence snadno implementovala a byla přijata v tradičních procesech vývoje softwaru.

4.2 Stírající se rozdíl mezi návrhem, vývojem a fungováním

Ke snížení časového rozsahu od vývoje až po provoz jsou stále častěji integrovány vývojové a provozní týmy. Kromě toho, jak bylo zjištěno před několika lety, je zde stále silná vazba a vzájemná závislost mezi softwarovými architekturami a infrastrukturou (operačních systémů, internetových architektur a infrastruktur, cloud computing…).

Tato pozorování mají minimálně tři hlavní důsledky:

1. Konstrukce moderního softwarového systému je striktně závislá na povaze a vlastnostech používané infrastruktury. Není moudré a není ani možné navrhnout systém ignorující infrastrukturu, která bude použita k sestavení, nasazovat a spravovat ji.

2. Systémy jsou vyvíjené jako kombinace klasických softwarových vývojových aktivit, dynamické infrastruktury, "on-the-fly" rekonfigurací Runtime komponent.

3. Nefunkční požadavky týkající se řízení výkonu a odolnosti proti chybám jsou zcela zásadní a musí být brány v úvahu po celou dobu rozvoje procesu, a nejen jako otázky týkající se implementace, které mají být řešeny "jakmile systém přejde do provozu".

Obecně tedy platí, že klasický rozdíl mezi designem, implementací a provozem má tendenci mizet nebo se radikálně předefinovat.

4.3 Zhodnocení a vizualizace procesů

Jednou z nejdůležitějších a zásadních otázek ve zvládnutí jakéhokoliv složitého fenoménu je schopnost pochopit a posoudit jeho stav a chování. V kapitole 2.7 jsme se zmínili, že nástroje jako Jenkins nabízejí metriky obvykle odvozené z výsledků analýzy kódu a testovacích nástrojů. Tyto metriky však nemusí být nutně dostačující, aby plně posoudily a vyhodnotily stav složitého procesu vývoje softwaru.

Obecně platí, že je nezbytné identifikovat a rozvíjet nové metody k posouzení, zastoupení a sdílení stavu procesu vývoje softwaru. Nejde jen o to, identifikovat nové a přesvědčivější metriky a klasické měřící programy, nebo si představit snadno použitelné uživatelské rozhraní. Opravdu potřebujeme nová paradigmata (a účinné podkladové technologie), která jsou schopna intuitivně zprostředkovat klíčové informace popisující stav procesu.

4.4 Bezpečnost, soukromí a důvěra

Software je stále více a více všudypřítomný. V důsledku toho jsou otázky týkající se bezpečnosti, soukromí a důvěry čím dál kritičtější. Bezpečnostní hrozby se mohou objevit kdekoli a jakákoliv fáze hodnotového řetězce může být napadena a vystavit firmy a jednotlivce vážným rizikům. Kromě toho impozantní vývoj mobilních technologií a služeb založených na internetu, spolu s rostoucími debatami o mezinárodních monitorovacích činnostech, se stali jedním z hlavních témat i pro komunitu softwarového inženýrství.

Jsou metody vývoje softwaru a technologie dostatečné k posouzení úrovně bezpečnosti, soukromí a důvěry softwarového systému? Jak můžeme dokázat svým zákazníkům a uživatelům, že softwarový systém vykazuje dostatečnou úroveň bezpečnosti? Je to něco, co lze a mělo by být řešeno nezávisle na klasických softwarových vývojových aktivitách? Nebo hůře, je to něco, co by mělo být považováno "mimo rozsah" a nebýt relevantní pro komunitu softwarového inženýrství?

Jistě, toto téma není jen technické, ale stejně jako výzkumní pracovníci můžeme identifikovat a posoudit hlavní problémy, navrhnout způsoby, jak monitorovat a řídit hrozby a zhodnotit dopad, který tyto problémy mohou mít na vývoj softwaru činnosti. Kdy bychom o nich měli uvažovat? Jak? Do jaké míry? Jaké techniky potřebujeme? Zodpovědná komunita vědců nemůže ignorovat nebo přehlížet tyto problémy a otázky, protože jsou velkým problémem pro občany, podniky a instituce po celém světě.

4.5 Kontrolovaná vs. neplánovaná integrace a kooperace

Spolupráce na aktivitách vývoje softwaru značně přesahuje klasické hranice tradičního vývoje softwaru společnosti. A zvláště tam jsou některé důležité trendy ke zvážení:

1. Spolupráce mezi podniky a mezi podniky a jejich zákazníky probíhá především v internetovém měřítku. Ve skutečnosti existuje značný počet nástrojů a prostředí, které se zaměřují na řešení tohoto stále většího sektoru trhu (například Asana a Basecamp).

2. Open source a jiné komunity založené na internetu mění vnitřní dynamiku, podle toho, jaký software je zrovna vyvíjen a distribuován.

3. Impozantní vývoj cloud computingu, internetových a Mashup služeb (např. IFTTT, Zapier, CloudWork a Mashape), a otevřených API ekosystémů (např. E015) radikálně transformuje pojem "spolupráce" mezi vývojáři softwaru.

Vývoj softwaru probíhá v nových, netradičních a nepředvídatelných situacích a scénářích. Je proto nezbytné vyvinout nové přístupy ke spolupráci a součinnosti, které berou v úvahu tyto trendy.

4.6 Model podniku a organizace

Díky velkým inovacím v technických a organizačních přístupech používaných ke vzniku a dodání softwaru, obchodů s aplikacemi a open source softwaru, se radikálně změnila organizace a struktura trhu se softwarem. Na jedné straně, software je často považován za spotřebitelský produkt, jehož náklady by měly odrážet trendy a vnímání tohoto druhu trhu. Na druhé straně, software je nejkritičtější a sofistikovanou součástí každého komplexního produktu nebo služby. Tudíž se stává stále obtížnější určit obchodní a organizační modely, které jsou schopny zaručit inovace, návrat investic a spokojenost zákazníků.

Samozřejmě, že existují určité oblasti (real-time a řídicí software, zakázkový software pro velké a vojenské organizace, vestavěné systémy…), kde jsou tyto změny méně relevantní a viditelné. Nicméně, mnoho klasických softwarových společností je nuceno radikálně měnit postavení na trzích, zaměřit se na strategii svých produktů a klíčových obchodních modelů. Microsoft je klasickým příkladem společnosti, která významně mění svou pozici a obchodní model: od běžných klientských aplikací na cloud computing (IaaS, PaaS a SaaS), od klasických licenčních režimů na jiné modely, včetně open source. Obecně platí, že ekonomika a obchodní modely softwaru se dramaticky mění. Je důležité zkoumat tuto transformaci, aby se zabránilo velkému riziku: obecné a neodůvodněné "komoditizaci" softwaru. Software není komodita, software něco stojí.

Závěr

Cílem této práce bylo jednak shrnutí zprávy z konference FOSE 2000, jejích připomínek a zjištění. Dále se práce zabývala přehledem hlavních směrů ve výzkumu softwarových procesů za posledních 15 let. Práce vychází hlavně ze závěrů práce nazvané Software Process od A. Fuggetty a E. Di Nitto.

První kapitola se věnovala softwarovým procesům do roku 2000. Druhá kapitola popisuje jejich vývoj za posledních 10 let. Třetí kapitola je věnována hlavním budoucím trendům a výzvám. Čtvrtá kapitola rozebírá problémy a možné směry výzkumu. Práce tedy poskytl přehled o současném stavu vývoje výzkumu softwarových procesů, problémy a výzvy budoucnosti.

Zdroje

BRIAND, Lionel, et al. 2012. Research-based innovation: A tale of three projects in model-driven engineering. *Conference on Model Driven Engineering Languages and Systems.* s.l. : Springer Berlin Heidelberg, 2012, s. 793-809.

CATALDO, Marcelo, HERBSLEB, James D. a CARLEY, Kathleen M. 2008. *Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity.* s.l. : ACM, 2008.

FUGGETTA, Alfonso a DI NITTO, Elizabetta. 2014. Software Process. *Proceedings of the on Future of Software Engineering. ACM.* 2014, s. p.1-12.

TRIST, E. L. a BAMFORTH, K. W. 1951. Some social and psychological consequences of the Longwall method. s.l. : Human relations, 1951.

WOHLIN, Claes, HÖST, Martin a HENNINGSSON, Kennet. 2003. Empirical research methods in software engineering. [aut. knihy] Reidar Conradi a Alf Inge Wang. *Empirical Methods and Studues in Software Engeneering.* s.l. : Springer Berlin Heidelberg, 2003, s. 7-23.