Česká společnost uživatelů otevřených systémů EurOpen.CZ
Czech Open System Users' Group
**www.europen.cz**

**37. konference**

# Sborník příspěvků

**Hotel Podlesí**
**Svratka**
**10.–13. října 2010**

## Programový výbor

Petr Fershmann
Jan Kynčl
Jan Valdman
Vladimír Rudolf

Sborník příspěvků z 37. konference EurOpen.CZ, 10.–13. října 2010

Plzeň 2010. První vydání.
Editor: Vladimír Rudolf
Sazba a grafická úprava: Ing. Miloš Brejcha – Vydavatelský servis, Plzeň
e-mail: servis@vydavatelskyservis.cz

Tisk: Typos, tiskařské závody, s. r. o.
Podnikatelská 1160/14, Plzeň



Příspěvky neprošly redakční ani jazykovou úpravou.

ISBN 978-80-86583-20-4

# Kompatibilita Androidů a lidí

**Tomáš Zvěřina**

E-mail: zverina@m-atelier.cz

Buďme rádi, zatím ještě nejde o zdířky, zásuvky a konektory ... zaměříme se na kompatibilitu na mnohem neškodnějším rozhraní. Na tom uživatelském. Jak s Vámi bude Váš Android komunikovat, až si ho pořídíte? Budete mu rozumnět? A on Vám?

Uživatelské rozhraní operačního systému Android je na své životní cestě determinováno několika zásadními elementy. Některými se konkurenci podobá, některými se odlišuje a některé jsou v dnešní době vlastně již samozřejmostí. Podívejme se na ně konkrétně.

## Touchscreen

Ano, to je jedna z těch banálních vlastností. Ale touchscreen dělá smartphone smartphonem a nejinak je tomu i u Androidu. U drtivé většiny zařízení se obejdete bez stylusu a poslouží Vám vlastní prsty. Případně mražený párek, pokud je počasí nevlídné. Ovládání pomocí prstů se přizpůsobuje vše. Velikost tlačítek, menu aplikací, skrolování, „kliknutí pravým tlačítkem“ – v tomto případě tedy „LongTouch“.

Od verze 2.0 se v API objevil také MultiTouch, tj. aplikace lze (pokud to podporují) ovládat více prsty současně. Užitečné především při zoomování, otáčení apod. Známe od konkurence.

## Hardwarová tlačítka

Když pomineme „power“ a „volume up/down“, vystačí si někteří odvážnější výrobci moštu a smartphonů jen s jedním tlačítkem. Ne tak Android! Mezi hardwarovými tlačítky Android telefonů neleznete:

*Home* – vrátí Vás z kterékoliv aplikace zpět na desktop. LongTouch Home tlačítka vyvolá nabídku šesti naposledy spuštěných aplikací.

*Back* – skočí zpět na předcházející obrazovku spuštěné aplikace, případně až na desktop. Koneckonců – tlačítko zpět, cancel nebo storno je prakticky na každé obrazovce, tak proč ho nevyčlenit a neunifikovat?

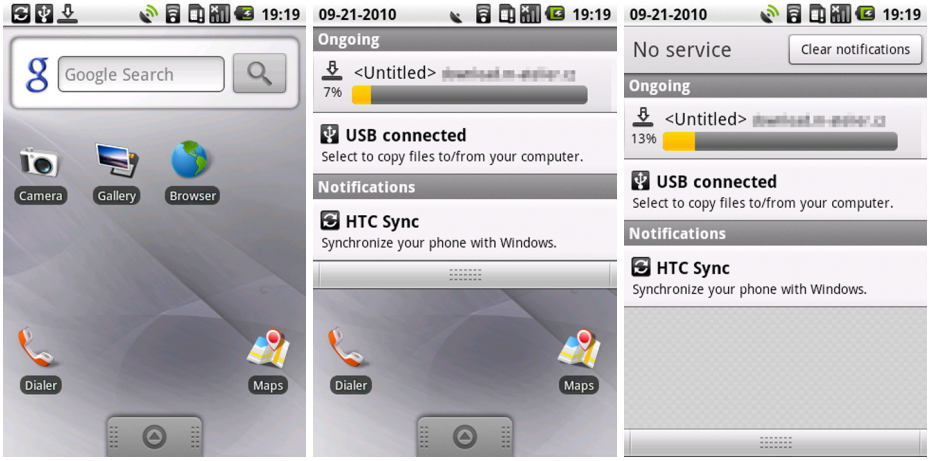
*Menu* – i to je jedna z jistot Android aplikací. Když zmáčnu tlačítko Menu, velice pravděpodobně se objeví . . . menu. No není to pěkné? Každá obrazovka má nějaké funkce, nabídky, možnosti, . . . zabírat na malém displayi místo ovládacími prvky pro Menu je přeci tak nehospodárné. A proto tu máme zvláštní tlačítko.

*Search* – mnoho aplikací obsahuje vyhledávání – emailový klient, mapy, poznámky, dokumenty, web. Stisknu tlačítko Search a formulář pro hledání (pochopitelně vybavený příslušně chytrým našeptavačem) je tu. Je ale pravdou, že se autoři Androidu tlačítkem pro hledání dostávají na temné území brandbuildingu a není proto divu, že se výrobci mobilních telefonů nerozpakují hodit toto tlačítko přes palubu. A mnoho Android aplikací tuto funkci ani nepodporuje.

## Notifikační lišta

Nebo chcete-li – Status Bar. Kterákoliv aplikace Vás může pomocí notifikací informovat o probíhajících událostech nebo dokončených operacích. A že toho Váš telefon může mít na srdci opravdu hodně: přijaté SMS, nové emaily, probíhající nebo dokončené stahování, nepřijaté hovory, dostupné aktualizace, . . . co jen vývojáře napadne.

V levém rohu lišty se zobrazují ikonky notifikací. Stažením lišty dolů si pak můžete notifikace prohlédnout.




Obr. 1

API pro ovládání notifikací je dotažené do puntíku. Aplikace mohou libovolně ovlivňovat vzhled notifikace, mohou notifikaci stáhnout, jakmile se stane zastaralou, kliknutí na notifikaci Vás přenese na tu správnou obrazovku, s ob-

jevením notifikace může v liště proběhnout nějaký text (např. SMS zpráva), lze všemožně vibrovat, vyhrávat, rozsvěcet diody na telefonu apod.

Nezapomínejme prosím na to, že vrozenou vlastností Androidu je multitasking. Z toho plyne, že Vás aplikace mohou o důležitých událostech notifikovat, i když právě viditelně neběží.

## Trackball

Ne každé Android zařízení trackballem disponuje, ale pokud ano, stojí to za to. Speciálně na webových stránkách, kde se to hemží odkazy, se Vám trackball bude hodit pro precizní přesouvání focusu mezi elementy obrazovky a stisknutí trackballu slouží coby touch (klik). Je utrpením uchopit zařízení, které trackball nemá. Třešničkou na dortu je pak podsvícení trackballu, které zajistí, že na Vás může Váš telefon přátelsky pomrkávat při došlé SMS (viz notifikace).

## Widgety na Home Screen

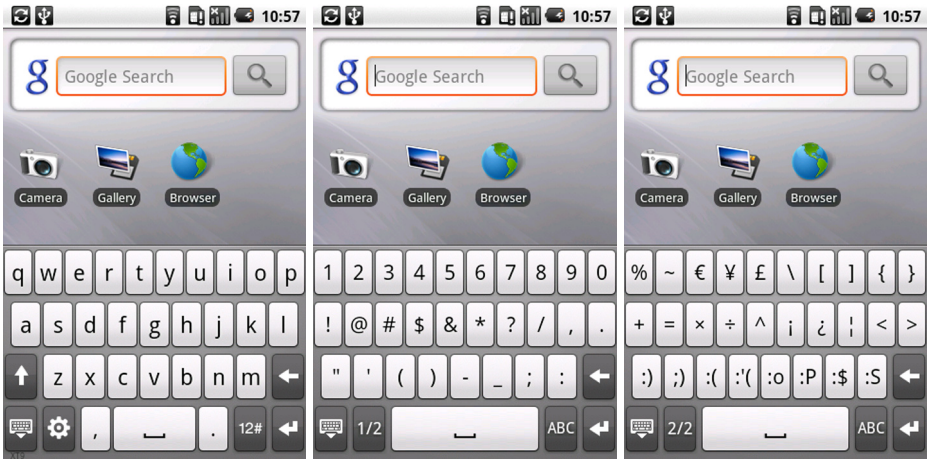
Příjemným zpestřením uživatelského rozhraní jsou miniaplikace (tzv. widgety) umísťované přímo na desktop telefonu. Nejrůznější přepínače a vypínače, RSS feed, Twitter feed, Facebook, Foursquare, kalendář, hodiny, počasí, stav baterie, ... Výhodou je, že nejste odkázáni na výrobce telefonu, ale můžete si widgety doinstalovávat dle libosti.

## Virtuální klávesnice

A jistě zde musíme zmínit také způsob zadávání textových informací. Pokud má Váš telefon hardwarovou klávesnici, není co řešit. Dobrá polovina dostupných zařízení ji ale nemá a pak přichází ke slovu klávesnice virtuální. Na displayi ovšem není moc místa a je třeba se psychicky připravit na množství překlepů. Rozložení klávesnice se mění v závislosti na tom, jakou hodnotu pole očekává (text, číslo, url, email, ...). Také je možné výchozí klávesnici nahradit jinou, specializovanou, nebo s jiným ovládáním, např. Swype (http://swypeinc.com/). Klávesnice na HTC Magic vypadáku příkladu takto (viz obr. 2).

## Android se mi líbí, co dál?

Výrobců je již hodně (HTC, Motorola, Samsung, Sony Ericsson, LG, ...) a telefonů také, stačí si jen vybrat. Určitě doporučím sledovat, o jakou verzi OS se




Obr. 2

jedná, pod 2.1 se nepouštějte. Také dbejte na to, aby byl výrobce podporován výchozí Android Market, mnozí výrobci se snaží vnutit Vám market vlastní.

A pokud ovládáte něco té Javy, navštivte stránky developer.android.com, kde si stáhnete SDK a naučíte se základy u zákoutí programování pro Android. Subjektivně je vývoj pro Android daleko příjemnější než pro JME, zvlášť pokud pro vývoj používáte Eclipse.

# MySQL, řešení problémů, málo známé vlastnosti

**Jakub Vrána**

E-mail: jakub@vrana.cz

## 1 Co nefunguje v MySQL a jak to obejít

Databázový server **MySQL** udělal za posledních několik let významný pokrok a k jednoduchému rychlému úložišti přidal i pokročilejší funkce. Některé věci ale nefungují nebo je potřeba si na ně dát pozor. Článek uvádí některé z nich.

### 1.1 U sloupce typu date nelze nastavit výchozí čas na aktuální

**Problém:** Hodnota DEFAULT CURRENT_TIMESTAMP z historických důvodů funguje pouze u sloupce typu `timestamp` a navíc ji lze použít jen u jednoho takovéhoto sloupce v tabulce.

**Řešení:** Častý požadavek na existenci dvou sloupců vytvoreno a zmeneno se tedy řeší poněkud krkolomně. Dá se zajistit vytvořením BEFORE INSERT triggeru:

```
CREATE TRIGGER tabulka_bi BEFORE INSERT ON tabulka FOR EACH ROW
SET NEW.vytvoreno = NOW()
```

Sloupec zmeneno může být klasický `timestamp`.

### 1.2 Sestupné indexy

**Problém:** MySQL při definici indexů ignoruje požadavek na sestupné třídění položek – (skupina DESC, poradi) vytvoří stejný index jako (skupina, poradi).

**Řešení:** MySQL dokáže takovýto index použít i pro sestupné třídění, důležité ale je, aby pořadí všech částí indexu bylo při třídění stejné: ORDER BY skupina DESC, poradi DESC index využije, dotaz ORDER BY skupina DESC, poradi ne. Pokud to nedokážeme zaručit, můžeme do tabulky vložit opačnou hodnotu sloupce a řadit podle něj, obvykle to ale potřeba není.

## 1.3 Indexy nad výsledkem funkce

**Problém:** MySQL na rozdíl třeba od **PostgreSQL** nedovoluje vytvářet indexy nad výsledkem funkce. Pokud na sloupec v dotazu aplikujeme nějakou funkci, tak se index až na výjimky nepoužije.

**Řešení:** Při porovnávání je tedy vhodné indexované sloupce uvádět samotné:

```
-- index se nepoužije
SELECT * FROM tabulka WHERE zmeneno + INTERVAL 1 DAY >= NOW();

-- použije se index nad sloupcem (zmeneno)
SELECT * FROM tabulka WHERE zmeneno >= NOW() - INTERVAL 1 DAY;
```

## 1.4 Využití indexů v poddotazu

**Problém:** MySQL nedokáže využít indexy pro setřídění výsledků poddotazu.

**Řešení:** Pokud z poddotazu vytvoříme pohled, MySQL indexy využít dokáže.

## 1.5 Materializované pohledy

**Problém:** Pohledy se v MySQL vyhodnocují při každém dotazu znovu. Na rozdíl od jiných databázových serverů nedokáže MySQL vytvořit tzv. materializovaný pohled, který by data fyzicky ukládal (a při změně aktualizoval) a nad kterým by třeba šly definovat i indexy.

**Řešení:** Vyřešit se to obvykle dá doplněním dopočítávaných sloupců a jejich automatickou aktualizací pomocí triggerů, dá to ale **dost práce**.

## 1.6 Trigger nemůže měnit stejnou tabulku

**Problém:** Trigger nemůže měnit data ve stejné tabulce, pro kterou je definován.

**Řešení:** Žádný *work-around* neznám, ale pokud nám stačí upravit modifikovaný záznam, lze to udělat změnou hodnot v „tabulce“ NEW.

Stejné omezení platí i pro poddotazy při modifikaci záznamu – ty se také nemohou dotazovat do stejné tabulky. Takový příkaz je nutné rozdělit do dvou – nejprve získat data a v druhém kroku provést aktualizaci. Toto omezení lze obejít pomocí druhého poddotazu.

## 1.7 Triggery se nespustí při kaskádovém mazání

**Problém:** Pokud definujeme cizí klíč s příznakem ON DELETE CASCADE a v tabulce je definovaný trigger pro smazání, tak se tento trigger nespustí, pokud se záznam smaže v důsledku kaskády.

**Řešení:** Řešení je pro tabulky s takovýmto triggerem nepoužívat kaskádové mazání a záznamy mazat ručně. Často ale trigger pouze mění záznam v rodičovské tabulce, kdy nám jeho nespuštění nemusí vadit.

## 1.8 Příkazy ukončující transakci

**Problém:** Všechny příkazy pracující se strukturou tabulek vyvolají implicitní COMMIT právě probíhající transakce. Platí to i pro další příkazy, např. ty pro práci s uživateli a donedávna třeba i pro příkaz LOAD DATA.

**Řešení:** V transakcích je tedy vhodné používat jen příkazy manipulující s daty.

## 1.9 Omezující podmínky

**Problém:** MySQL ignoruje omezující podmínky definované klauzulí CHECK při vytváření tabulky.

**Řešení:** Obejít se to dá triggerem, který v případě nesplnění podmínky vyvolá chybu:

```
CREATE TRIGGER uzivatel_bi BEFORE INSERT ON uzivatel FOR EACH ROW
IF CHAR_LENGTH(NEW.login) < 3 THEN
    DO ‘Login musí mít alespoň tři znaky.‘;
END IF
```

Stejný trigger bychom samozřejmě museli definovat i pro změnu záznamu.

Všimněte si také krkolomného způsobu vyvolání chyby, který navíc MySQL obalí hláškou, že sloupec daného jména neexistuje. V MySQL totiž dosud neexistuje příkaz SIGNAL, který by se dal pro vyvolání chyby použít.

## 1.10 Závěr

MySQL je stále mladý databázový server a otázkou je, jestli budou uvedené nedostatky v blízké budoucnosti odstraněny. Lepší je proto o nich vědět a naučit se s nimi žít.

# 2 Srovnání funkcí MySQL a PostgreSQL

Webové aplikace jsem začal vyvíjet s PostgreSQL, pak jsem kvůli výkonnosti přešel na MySQL. Bylo to ale v dřevních dobách a třeba jsem jen PostgreSQL nedokázal správně nakonfigurovat. Z MySQL jsem myslím schopen vytáhnout slušný výkon a expert na PostgreSQL by jistě dokázal totéž i s ním. Na různé úlohy navíc může být vhodná jiná databáze, o rychlosti proto psát nechci.

Zajímá mě srovnání funkčních vlastností. MySQL má řada lidí zaškatulkovanou jako primitivní úložiště, to už ale díky verzi 5 neplatí. Takže připomenu, že transakce, cizí klíče (s tabulkami InnoDB, které používám), poddotazy, pohledy, triggery a uložené procedury už MySQL dlouhou dobu podporuje. Některé věci třeba ve srovnání s PostgreSQL s některými omezeními, ta ale reálnému využití nebrání. Co se mi tedy líbí na které databázi?

## 2.1 MySQL

- Vynikající podpora **kódování**. Pro každou databázi, tabulku a sloupec lze určit nezávislé kódování a způsob porovnávání, komunikovat z databází lze v kódování nezávislém na datech.
- Šikovná agregační funkce GROUP_CONCAT.
- Klauzule ON DUPLICATE KEY UPDATE, která dovoluje snadno **pracovat s unikátními číselníky** nebo aktualizovat statistické tabulky.
- Hodit se může i podpora **fulltextového vyhledávání**, ta je ale k dispozici bohužel jen v tabulkách typu MyISAM (v PostgreSQL je od verze 8.3).
- Pro pevně dané výčty je v MySQL k dispozici datový typ **enum**, který je velmi úsporný a efektivní, přitom se s ním pohodlně pracuje. V ostatních databázích se stejného chování dosahuje klauzulí CHECK u řetězcového sloupce, to ale není tak úsporné a efektivní. PostgreSQL podporuje **enum** od verze 8.3.
- Velmi využívaná je replikace, vždyť je v MySQL už od verze 3.23. PostgreSQL zavádí replikace až ve verzi 8.3, MySQL ale replikaci stále zdokonaluje.
- Líbí se mi, jak v InnoDB tabulkách interně fungují transakce. Ty jsou optimistické a data zapisují rovnou na konečné místo a vedle toho si vedou *rollback log*. Když se transakce potvrdí, tak tento log jen smažou, do té doby ho využívají ostatní transakce pro přístup ke staré verzi dat. PostgreSQL to řeší jinak a je potřeba v něm pravidelně spouštět příkaz VACUUM. To se dá v nových verzích řešit i automaticky, ale přístup InnoDB se mi líbí víc.

- Umělé primární klíče se snadno vytváří příznakem AUTO_INCREMENT. V PostgreSQL je nutné používat sekvence, případně zkratku **serial**.
- Modifikátor SQL_CALC_FOUND_ROWS dovoluje zjistit počet všech řádek nezávisle na omezení klauzulí LIMIT.

## 2.2 PostgreSQL

- Na Postgresu mě nadchl sloupec **oid**, pomocí kterého lze snadno pracovat se záznamem v libovolné tabulce. Třeba v Admineru se mi velmi obtížně pracuje s tabulkami s vícesloupcovým primárním klíčem nebo úplně bez něj. Pracovat s takovýmito tabulkami jde, ale se sloupcem **oid** by to bylo jednodušší.
- Mnohem propracovanější jsou v PostgreSQL indexy. Je možné nadefinovat funkční a částečné indexy. Většinou se bez nich dá žít, ale některé úlohy mohou zjednodušit.
- Líbí se mi datový typ pole. Sice jeho existence může někoho svádět ke špatnému návrhu databáze, ale věřím, že bych ho dokázal využít smysluplně.
- Pokud dojde v některém příkazu v rámci transakce k chybě, tak se další příkazy neprovedou. To může být někdy na překážku, většinou to je ale žádoucí chování.
- V uložených procedurách se dají bez hacků používat různé programovací jazyky. V MySQL jen s **hacky**.
- Někdo může ocenit, že tabulky v rámci jedné databáze lze sdružit do schémat. Osobně mi vyhovuje spíš plochá struktura, takže sám vytvářím spíš více menších databází.
- Podpora pravidel, pomocí kterých lze vytvářet virtuální tabulky.

## 2.3 Závěr

Myslím, že co se funkčnosti týče, tak jsou MySQL a PostgreSQL srovnatelné databáze. Mě se víc líbí vychytávky MySQL, ale s PostgreSQL bych pracoval také bez velkého přemáhání. Druhou oblastí je výkonnost a správa, do srovnání toho se ale pouštět nechci.

Přes PostgreSQL nejsem žádný expert, takže budu rád, když další postřehy doplníte do diskuse, já bych je pak případně zohlednil i v článku. Stejně tak jsem mohl zapomenout na nějakou unikátní vlastnost MySQL.

# Co víte a nevíte o PostgreSQL

**Pavel Stěhule**

E-mail: pavel.stehule@gmail.com

## Změny v procesu vývoje RDBMS PostgreSQL

PostgreSQL má za sebou patnáct let vývoje – pokud bereme pouze v potaz dobu, kdy zodpovědnost za vývoj převzala komunita (některé části pg jsou nepochybně starší). Během těchto patnácti let se změnil svět, změnilo se IT a ke změnám došlo i v procesech vývoje této databáze. Historii post akademické éry PostgreSQL lze rozdělit do několika etap:

Raná éra – opravy chyb, čištění kódu, refaktoring kódu – cílem je stabilní kód – automatické reformátování existujícího kódu – relativně malý tým – v porovnání s dneškem minimální nasazení PostgreSQL (cca řada 6.x).

První úspěchy – větší patche – snaha o implementaci ANSI SQL 92, změna správy paměti, první komerční úspěchy – rozšiřuje se povědomí o PostgreSQL – 7.4 je i komerčně úspěšná databáze – RedHat PostgreSQL. Dodnes se používá v produkci. Stále relativně malý tým – relativně malé patche – daří se udržovat 1 release každý rok. Seznam patchu je udržován Bruce Momjanem (cca řada 7.x).

Potíže s růstem – začínají se objevovat relativně velké patche, rozrůstá se počet vývojářů – nyní snaha o použitelnost v Enterprise segmentu a o podporu ANSI SQL 2000. Ztrácí se patche, přihlášené patche nejsou zprocesovány – subjektivní přístup k prioritám – příliš dlouhé feature freeze období – commiters nestíhají (roste seznam čekajících patchů). Velké patche – nativní podpora MS Windows, integrace fulltextu, HOT update, ... (řada 8.x). Diskuze o sw pro řízení vývoje – diskuze o délce cyklu (rok vyhovuje vývojářům, DBA preferují 2–3 roky). (cca 8.0–8.2, problematické verze 8.3 a 8.4)

Nová metodika vývoje – V reakci ná problematický vývoj verzí 8.3 a 8.4 a v očekávání zhoršení situace díky integraci replikace v 8.5 se radikálně mění vývojový model – zavedení kratších cyklů ukončených commitfestem – zavedení release managera, commitfest managera, každý patch před zpracováním commiterem zpracuje (otestuje, zkontroluje) recenzent – reviewer. Stále problémy, komplexní patche nelze zpracovávat v posledním cyklu, nicméně zavedení procesu uklidnilo atmosféru v komunitě – není zapomenutých patchů, nedochází k odsouvání patchů. Vývoj 9.0 proběhl relativně v klidu.

Post CVS období – 9.1 migrace z CVS do GIT – důraz na kvalitu, důraz na dodržování časového plánu, důraz na přátelskou atmosféru v komunitě – pravidla platí pro všechny – několik málo jednoduchých pravidel, intenzivní komunikace – řešíme problémy teď a tady – neodsouváme problémy. Např. oprava chyb – není bugzila – pokud je to možné, tak jsou chyby opraveny do 5 pracovních dnů. Pokud to možné není, tak přidání bodu do ToDo. Pokud je příležitost, dochází k refaktoringu kódu (nic nepotěší Toma Lanea tak jako patch, který redukuje existující kód).

http://www.ohloh.net/p/postgres/analyses/latest

PostgreSQL ... cca 600 tis řádek kódu, cca 200 tis řádek komentářů – počet řádků kódu stoupá lineárně (od roku 2000). Vývojáři PostgreSQL jsou extrémně konzervativní – kód je v C90, teprve nyní se přechází na Git, nepoužívá se žádný komplexní sw, pouze elektronické konference, archivy a v posledních dvou letech mediawiki. Díky tomuto přístupu lze snadno dohledat veškerou komunikaci 13 let zpátky:

http://archives.postgresql.org/pgsql-hackers/

Důraz na efektivní používání jednoduchých vývojových nástrojů – pokud možno nic složitého a nic komplexního. Důraz na kvalitní komentáře jednotlivých commitů. Přechod na Git cca po třech letech diskuzí a roce příprav:

http://git.postgresql.org/gitweb?p=postgresql-migration.git

Za vývojem PostgreSQL nestojí žádná komerční organizace – vývoj je čistě komunitní – nicméně poměrně dobře organizován a udržován – viz core team, major contributors, contributors – nyní více než cca 100 osob – z toho 30 na full time. Vlastní vývoj je stále náročnější na čas – vývoj jedné komplexnější funkce zabere cca 4 člověko měsice a v reálném čase od uvedení prototypu do commitu uběhne rok (dva roky nejsou výjimkou).

# PostgreSQL 9.0

Tato verze zahajuje novou řadu, během které by mělo dojít k implementaci různých modelů replikace databáze, k dořešení implementace partitioningu a snadnější integraci databází (SQL/MED). Z toho se ve verzi 9.0 bylo implementováno:

- Hot standby režim – implementace slave uzlu (databáze je v ro režimu, zpracovává transakční log master uzlu, bez restartu se může přepnout do standardního režimu.

- Streaming replication – implementace distribuce transakčního logu od master uzlu k slave uzlu.
- In-place upgrades – v principu migrace pouze systémových tabulek – datové tabulky zůstávají beze změn – čímž se radikálně snižuje čas pro upgrade.
- Verze pro 64bitové systémy fy. Microsoft
- Hromadné nastavení práv – nastavení práv pro všechny tabulky ve schématu, případně možnost určit výchozí nastavení práv pro db objekty.
- Anonymní bloky a pojmenované parametry – anonymním blokem se míní jednorázově vykonaná kód v některém z podporovaných PL jazyků. Pojmenované parametry – programátor může určit parametry funkce nikoliv pouze pořadím nýbrž i jménem.
- Agregační funkce s určeným pořadím – libovolné agregační funkci lze předepsat pořadí vstupních hodnot.

Vývoj této verze trval cca 13 měsíců – některé nově zavedené koncepty jako je např. eliminace nevyužitých relací v SQL dotazech mohou razantně zvýšit rychlost řady aplikací.

## PostgreSQL 9.1

Verze 9.1 je aktuální vývojová verze – zatím nelze přesně vyjmenovat, které nové funkce bude obsahovat – vývoj PostgreSQL není řízen podle *ToDo* nebo dokumentem typu *Roadmap*. To, co je v které verzi záleží na zájmu vývojářů, případně zájmu sponzorujících firem a potřeb uživatelů. Nikdo nikomu neurčuje, co bude kdo dělat. Nicméně již nyní lze odhadnout, že v 9.1 bude implementováno:

- SQL/MED – podpora externích tabulek
- COLLATION na úrovni sloupců
- podpora synchronní replikace
- integrace s SE-Linuxem
- podpora klasické úrovně izolace SERIALIZABLE
- podpora datového typu JSON
- SQL příkaz MERGE

- zobrazení zdrojového kódu funkce, funkce LEFT, RIGHT, REVERSE, FORMAT, MEDIAN, CONCAT, sprintf (contrib), možnost vložení všech identifikátorů do uvozovek v dumpu databáze, atd

Vývoj by měl trvat zhruba rok a 9.1 je očekávaná na podzim roku 2011.

# Agility at scale aneb RUP v agilním světě

**Jan Valdman**

E-mail: jvaldman@dns.cz

**Abstrakt**

*Softwarový průmysl se změnil, dnes je běžná značná adopce agilních metodik. Tím došlo ale zároveň i k posunu významu „agile“, ze kterého se stal populární buzzword. S adopcí agilních principů a jejich nepopiratelných přínosů však vznikají i nové typy problémů a potíží. V tomto příspěvku se podíváme, co znamená škálování agilních metodik a jaké problémy je potřeba řešit. Stručně bude zmíněn Agility Scaling Model (AGS). S odkazem na populární RUP se stručně zamyslíme na jeho vztah k agile. Systematické nasazení agilních metodik se neobejde bez vhodných nástrojů, proto se krátce seznámíme s nástroji IBM Rational Team Concert, Rational Insight a metodiku Rational MCIF. Článek je převážně kompilací několika nedávno uveřejněných textů doplněný o vlastní postřehy autora.*

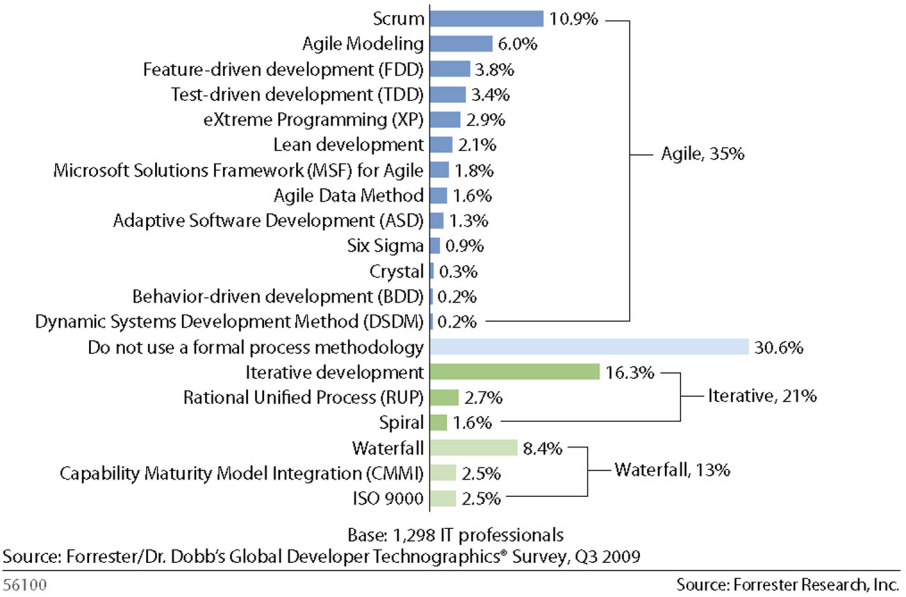
**Keywords:** Softwarový proces, metodiky vývoje software, agilní metody, škálování

## 1 Úvod

Agilní metodiky se staly v posledních letech široce rozšířenou praktikou, neboť se ukázalo, že opravdu fungují, že agilní týmy dosahují vyšší úspěšnosti, doručují kvalitněji, v kratších termínech a většímu uspokojení svých stakeholderů. Asi nejčerstvějším důkazem je nedávná zpráva Forrester [1], ze které vyplývá, že 35 % respondentů používá „agile“.

Softwarové metodiky se v posledních letech nejčastěji dělí na dvě skupiny:

- **Rigorózní metodiky** typu RUP, CMMI považované za formální, velké a těžké. Tyto metodiky věří, že softwarový proces lze plánovat, měřit a řídit, a že kvalita výstupu je úměrná kvalitě procesu. Důraz je kladen na předvídatelnost a opakovatelnost procesu a jeho cizelování.




Obr. 1: Agiles Organizations' Primary Development Approach

- **Agilní metodiky**, které naopak tvrdí, že vývoj software je empirický proces (tj. prakticky experiment) neboť pokaždé probíhá v odlišných podmínkách (jiný úkol, jiný zákazník, jiný tým, jiná technologie, ...) a tedy snaha o opakovatelnost je iluzorní.

Realita je však taková, že i agilní týmy musí plánovat, měřit a řídit, byť o tom agilní puristé neradi mluví.

Softwarové inženýrství je v jistém ohledu boj s nejistotou a rizikem. Absence fyzikálních limitů a zákonitostí umožňuje na rozdíl od jiných oborů lidské činnosti (například stavebnictví) měnit prakticky kdykoliv cokoliv. V softwarovém inženýrství chybí staletí zkušeností (viz např. opět stavebnictví) a pokušení o znovuvynalézání kola je příliš velké. Bodů variance je v softwarových projektech příliš mnoho, takže jakékoliv plánování a odhadování je velice obtížné a nepřesné, protože už samotné měření „vhodných" veličin a získávání podkladů pro rozhodování je obtížné. Pokud se budeme na softwarový vývoj dívat s odstupem, tak lze prohlásit, že vodopádový přístup byl historicky nahrazen přístupem iterativním, který snížil riziko neúspěchu, zavedl brzkou integraci a snížil objem neproduktivních předělávek, zavedl používání architektury a middleware komponent... V tomto pohledu je agile pouze dalším vývojovým krokem, který přináší zvýšení efektivity a orientaci na výsledek. Iterativní metody se vyvinuly v agilní metody tím, že se naučily lépe pracovat s nejistotou a přesností.

# 2 Agilní metodiky

Kde iterativní metodiky spoléhají na plánování a definování procesu a obsahu, tam agilní přístupy využívají odhady a zjednodušování. Agilní metody boří paradigmata iterativního vývoje (například že rozsah projektu nelze příliš měnit, klasický trojúhelník scope-resources-schedule atp.) Agilní metody prosazují „dělat správné věci“ před „dělat věci správně“ a snaží se eliminovat všechny neproduktivní nebo neefektivní činnosti, zejména byrokracii, papírování a nerealistické plánování. Hlavním přínosem agilních metod je zvýšená flexibilita, která dovoluje i větší změny v průběhu projektu, resp. neustálé vyjednávání o rozsahu (scope), plánech a řešení.

## Starý a nový agile

Původní pojetí agile bývá vykládáno s odkazem na tzv. agilní manifest [4], který deklaruje 4 hodnoty a 12 principů agilního vývoje. Tyto hodnoty a principy obstály v čase a i po deseti letech stále platí, přičemž žádný z nich nebyl zavržen nebo zásadně pochyben.

V posledních dvou letech ale proběhlo několik průzkumů, např. [7, 8] ze kterých vyplývají rozporuplná pozorování. Nejen, že například úspěšnost agilních a iterativních metodik je prakticky stejná, ale objevují se i zkušenosti o selhání agile nebo zprávy o potížích s jeho implementací. Navíc se ukazuje, že některé týmy si přisvojují atraktivní nálepku „agile“ aniž by ve skutečnosti byly agilními – což situaci ještě více zamlžuje.

Původní pojetí agile bylo vodou na mlýn vývojářů, kteří byli stavěni do středu procesu, Agilní metodika říkala to, co chtěli slyšet (důraz na spustitelný kód „hlavně ať to funguje“, potírání projektové byrokracie apod.) a dávala jim pocit, že jim patří softwarový process.

## Nový agile

Agilní metodiky jsou dnes používány v široké škále situací, nejen malými úzce spolupracujícími týmy, jak je popisováno ve starší literatuře. Agilní přístup se používá během různých fází projektu, nejen v konstrukční fázi. Hledají se tedy nové definice agility, viz například [7]. Hledají se způsoby, jak úspěšně implementovat agile ve velkých organizacích s tradičním způsobem řízení firmy a vztahy mezi byznysem a IT.

## Problémy agilního vývoje

Při pokusech o adopci agile v organizaci problémy vznikají i mimo samotné IT. Byznys, který je zvyklý z iterativních dob dostávat (nesmyslně) přesné odhady

již na začátku projektu. Pro byznys resp. celou organizaci je poměrně náročné opustit zaběhnuté koleje rozpočtování, plánování a řízení projektů a namísto vyjednávání o rozsahu (scope) a termínech se soustředit na spolupráci a dodávanou hodnotu.

Během adopce agile mnoho týmů narazilo na potíže. Ukazuje se, že přijetí agile znamená náročnou filosofickou změnu z „development focus“ na „delivery focus“, tj. místo na aktivity procesu samotného se soustředit na jeho výsledky. Termíny jako požadavek, kvalita nebo změna dostávají jiný význam. Navíc se ukazuje, že v naší „kultuře“ je hluboce zakořeněný postup řešit projekty vodopádem.

Při nasazení agile na větší a komplexnější projekty se ukázaly některé časté nedostatky

- Sdílení statusu je časově náročné
- Mnoho agilních praktik vyžaduje automatizaci
- Retrospektivy vyžadují informace
- Různé situace vyžadují používání odlišných praktik

## Doporučení

Vždy jsem vnímal označení „agilní“ za záminku pro jistou formu nepořádku a vyhýbání se dělání nepopulárních věcí, jako jsou například plány a dokumentace. Scott Amber ve svém článku dokonce píše (volně cituji z [6]), že v mnoha organizacích jsou nedisciplinované týmy, které přečetly jeden nebo dva články o agile, a které „implementovaly agilitu“ ať to již znamená jakoukoliv novátorskou, svobodomyslnou a nedokumentovanou softwarovou kreativitu.

Podle Scotta Ambera musí tým splňovat následujících pět jednoduchých a poměrně snadno vyhodnotitelných kritérií, aby mohl být považován za opravdu agilní:

1. Produkovat funkční software (working software) na pravidelné bázi, v rámci častých, time-boxovaných iterací
2. Aktivně a úzce komunikovat se stakeholdery
3. Regresně testovat minimálně na úrovni vývojáře, lépe na úrovni celého projektu formou např. test-driven development (TDD)
4. Mít rozumnou organizaci, tj. mít rozumné složení sebeorganizujícho týmu, který funguje v rámci přiměřeného principu řízení (governance framework)

5. Neustále se snaží zlepšovat, tj. pravidelně vyhodnocují úroveň své činnost (lepší týmy ji i měří) na časové bázi implementují zlepšení

Na základě stovek pozorování učiněných společností IBM v uplynulém desetiletí byly formulovány následující principy agilního delivery:

1. Snižujte nejistotu v projektu, architektonicky významnými rozhodnutími se zabývejte nejdříve

2. Nastavte adaptivní delivery proces, který je schopen usměrňovat směr dalšího vývoje a zužovat množství variant řešení

3. Omezte množství zakázkového kódu využitím co největšího počtu znovupoužitelných prvků a middleware komponent

4. Instrumentujte proces tak, abyste byli schopni měřit cenu změn, trendy kvality a postup projektu

5. Čestně komunikujte postup projektu se všemi stakeholdery včetně situací, kdy jste se místo dopředu museli vrátit zpět

6. Pravidelně spolupracujte se stakeholdery na přehodnocení priorit, rozsahu, zdrojů a plánů

7. Soustavně testujte a vyhodnocujte jednotlivá vydání produktu (která se zvětšují, takže je to čím dál více práce)

8. Vytvořte prostředí (včetně nástrojů) pro podporu týmové práce

9. Podporujte skrze automatizaci svobodu měnit plány, rozsah a jednotlivá vydání produktu

10. Použijte vhodný model řízení, který dává lidem svobodu kreativně tvořit

Úspěšné agilní projekty mají následující charakteristiky:

1. Potřeby a požadavky stakeholderů se vyvíjejí (dotvářejí) na základě již existujících prvků řešení. Požadavky (requirements) tedy nejsou shromážděny na začátku, ale jsou definovány jako jakýsi aktuální rozpor mezi potřebou a skutečnosti a jsou naplňovány postupně v rámci jednotlivých iterací. Vize se postupně přetváří na dílčí kritéria vyhodnocení iterací a nakonec až v akceptační kritéria. Rozsah (scope) se vyvíjí z abstraktní podoby do přesné specifikace, tak je roste porozumění potřebám stakeholderů a klesá nejistota v projektu.

2. Delivery proces a nástroje se vyvíjí od volnějších a flexibilních forem k více svázaným procesům tak, jak postupuje realizace projektu a blíží se termín produkčního nasazení. Toto postupné „utahování šroubů“ (např. v pozdějších iteracích platí přísnější pravidla) funguje jako kompromis mezi svobodou a pořádkem.

3. Evaluace projektu je čestná. Reálné projekty vykazují nárůsty i poklesy v kvalitě, hodnotách i trendech dokončení projektu, což odpovídá postupnému snižování nejistoty, refaktoringu, nečekaným potížím apod.

4. Testování je hlavní řídící mechanismus. Objektivní testování spustitelného kódu je jednou ze základních činností v projektu a je jednou z klíčových činností, které podávají informace o jeho zdraví a postupu prací.

## Škálování agility

V raných dobách byly projekty realizované pomocí agilních metodik relativně malé a přímočaré. Takové projekty šlo velmi dobře realizovat pomocí sady základních agilních postupů. Poté se však situace změnila s tím, jak se organizace pokoušejí používat agile na projekty nejrůznějšího druhu. Ukazuje se, že kontext a velikost softwarového projektu mají podstatný vliv na „úspěšnost“ agilního přístupu. Původní malé sady agilních praktik fungovaly dobře v malých týmech soustředěných na jednom místě a orientovaných zejména na konstrukční etapy projektu. Potřeba resp. pokusy nasadit agile na větší a sofistikovanější projekty ukázala nutnost adoptovat nějaký ucelený agilní delivery proces, od jeho zahájení až po nasazení do produkce. Volba vhodné strategie závisí na konkrétních okolnostech, zejména komplexitě daného projektu, kterou lze vyjádřit různými faktory:

- Velikost týmu (většina agilních týmů má dle průzkumu do 10–20 osob)
- Geografické rozložení týmu (v praxi není možné umístit celý tým do jedné místnosti)
- Regulace a compliance (softwarový proces musí splňovat různé normy typu ISO, CMMI apod.)
- Složitost problému (první pokusy s agile bývají orientované na relativně přímočaré problémy/projekty)
- Organizační bariéry (v reálných projektech jsou členové týmu často z jiných divizí, od partnerů, od dodavatelů, ...) co znemožňuje jejich alokaci do společného prostoru

- Technická složitost (je mnohem jednodušší dosáhnout vysoké kvality, pokud systém děláte od začátku. V reálném světě projekty staví na předchozích systémech, nekvalitním předchozím kódu, nespolehlivých datových zdrojích, nedokonalých API apod.; takoví „kostlivci ve skříni“ dokážou snadno zmařit agilní ideály)

- Firemní kultura (mnoho organizací je na úrovní řízení nebo kultury nastavená na vodopádový model a není schopno vstřebat agile; v některých organizacích existuje více skupin, které mají odlišný názor na agilní postupy; individuální nebo skupinové strategie mohou fungovat dobře, ale nemohou vzdorovat systému a nasměrování celé organizace)

- Podniková architektura (mnoho větších organizací vyznává globální dlouhodobé strategie nebo iniciativy, které směřují ke standardizaci infrastruktury, business flexibility nebo enterprise architecture; agilní týmy orientované na bezprostřední výsledky a okamžité hodnoty se mohou dostat do sporů)

Výše uvedené faktory ilustrují, že často vznikají situace, ve kterých nelze snadno (anebo vůbec) vyhovět některým základním požadavků agilních praktik, protože uvedená omezení jsou v reálném světě prostě stanovené „shora“ a nezbývá, než je přijmout.

Organizace, které používají například RUP, obvykle zajímá, zda RUP může být agilní a jakým způsobem mohou zavést agilitu do svých procesů. Při tom mohou narazit na různé obtíže:

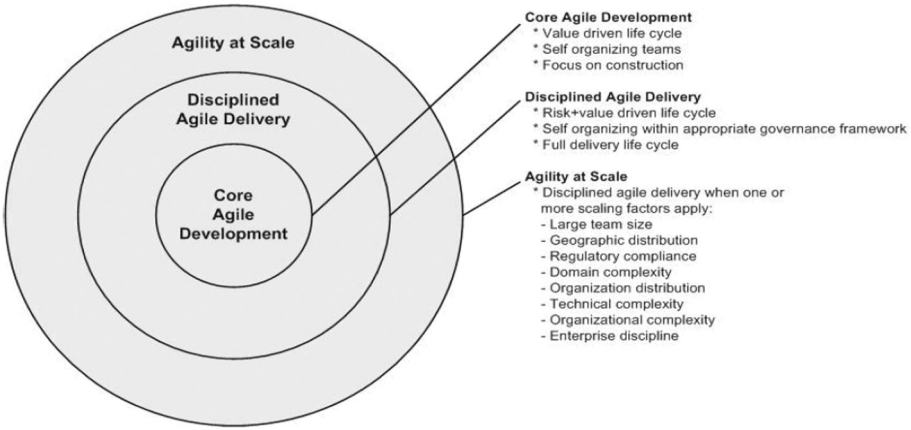
1. Strategie tailorování procesu a zavedení agilních praktik, například denních stand-up schůzek se zdá přímočará, ale brzo se narazí na problém velikosti týmu, neefektivní sdílení status informace. Rozdělení na menší subtýmy (nebo opačně např. „scrum of scrums“) často vede na to, že subtýmy pouze plní wiki nebo databázi výstupy ze svých schůzek.

2. Strategie adoptovat agilní praktiky „za pochodu“ dle potřeby nepřináší větší problémy kromě potřeby takové změny formálně dokumentovat a propagovat do formálních procesních popisů typu ISO 9000.

3. Manuální postupy fungující pro malé agilní týmy neškálují. Konkrétně jsou zmiňovány problémy typu nedostatek bílých tabulí nebo neřešitelnost správy stovek post-it štítků nalepených po zdech. Větší týmy musí nezbytně používat vhodné nástroje (viz například Rational Team Concert popisovaný níže). Nástrojové strategie jsou v malých týmech jednoduché a přímočaré, ale velké týmu řeší problémy integrace existujících (a často i předepsaných) nástrojů.

# 3 Agility Scaling Model (ASM)

U běžných agilních metodik jsem si vždy uvědomoval, že postihují vždy jen některou z disciplín potřebných k realizaci projektu. Z mého pohledu, který je nejvíc ovlivněn metodikou RUP, jsou agilní metodiky velmi malé a nedostačující k „end-to-end“ realizaci projektu. Podobně, ale ovšem mnohem lépe, to zformuloval Scott Ambler ve svém ASM modelu [6].




Obr. 2: Overview of the Agile Scaling Model (ASM)

Model ASM zobrazuje soudobé agilní přístupy jako základní agilní jádro, které je obaleno dvěma dalšími vrstvami sofistikovanějších metod.

## A. Agilní jádro

Agilní jádro zahrnuje základní metody typy Scrum nebo XP, které jsou sebeorganizující, orientované na doručování hodnot a obvykle zaměřené jen některé činnosti nebo etapy v projektu. Tyto metody se opírají o základní hodnoty a principy agilního manifestu. Jsou optimalizované pro menší týmy situované do jedné lokality a problémy, které mají relativně přímočaré řešení.

## B. Disciplinovaný agilní softwarový proces

Disciplinované agilní metody, jako je například OpenUP, jsou dále a snaží se pokrýt celý vývojový proces, všechny jeho fáze od přípravy projektu až po nasazení do produkce. Disciplinované agilní procesy jsou orientované na řízení hodnoty i řízení rizika a stále mají sebeorganizující charakter v rámci daného systému

řízení (governance framework). Obvykle kombinují některé základní agilní praktiky s některými „tradičními“ přístupy, jako je trocha plánování na začátku projektu nebo modelování architektury.

Klíčové atributy této skupiny jsou:

- **Plný rozsah metodiky** (delivery lifecycle), který bývá z globálního pohledu sériový (existují fáze projektu) a v detailu iterativní.
- **Dodávají řešení**, ne pouze sofware, který bývá pouze jednou složkou řešení.
- **Řídí rizika i hodnoty.** Základní agilní princip dodávat viditelné hodnoty je rozšířen o řízení a mitigace rizik.
- **Jsou sebeorganizující v rámci daného systému řízení** (governance framework), tj. je aplikován přiměřený vliv vnější „vlády“, tj. respektují pravidla a omezení organizace (např. normy ISO 9000, společnou infrastrukturu, globální strategie, ...)

## C. Škálovatelná agilita

Tato kategorie se zabývá situacemi, kde dominuje některý z faktorů škálovatelnosti, jako je například velikost týmu, složitost problému apod (viz výše). Takové situace vyžadují dále kultivovat disciplinované agilní procesy, přidat nové praktiky, zavést více disciplíny a tak vytvořit plnohodnotný sofistikovaný proces. Samotné agilní core metodiky (jako například Scrum nebo XP) v takových situacích již samy o sobě nefungují.

V této kategorii se uplatňuje systematický přístup k tailorování agilních praktik a do procesů se dostává více „metodiky“. Na rozdíl od tradičních přístupů zde však nejde o opakovatelné procesy (např. ve smyslu CMMI level 2–3), ale opakovatelné výsledky. Vývojový proces je instrumentován o nástroje podporující týmovou spolupráci, automatizaci a získávání informací pro retrospektivy s co nejmenší pracností a režii.

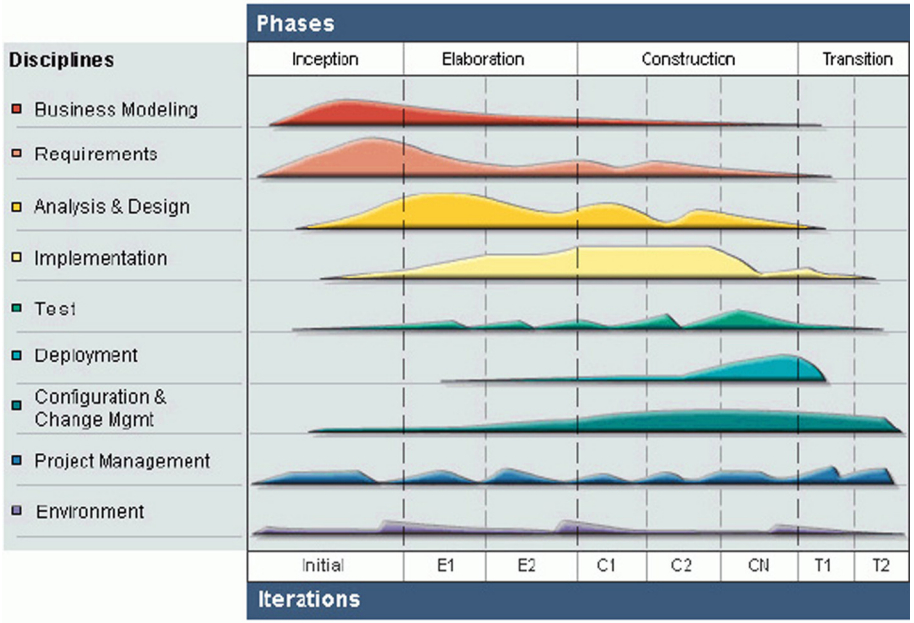
Rozdělení běžných agilních metodik na tři skupiny dle ASM ukazuje tabulka 1:

Tab. 1

| **Agile Core** | **Disciplined Agile Delivery** | **Agility at Scale** |
|---|---|---|
| Agile Data (AD)<br>Agile Modeling (AM)<br>Extreme Programming (XP)<br>Feature Driven Development (FDD)<br>Scrum | Agile Unified Process (AUP)<br>Agile With Discipline (AWD)<br>Eclipse Way<br>Rational Unified Proces (RUP)<br>Open Unified Process (OpenUP) | Enterprise RUP (EUP)<br>Rational Measured Capability Improvement (MCIF) |

# 4 RUP v agilním světě

Rational Unified Process (RUP) byl vlajkovou lodí softwarových metodik, v devadesátých letech vyvinutý firmou Rational, kterou později koupila IBM. RUP je klasifikován jako velká a těžká rigorózní metodika, která je velice těžkopádná a vhodná leda tak na velké projekty o stovkách vývojojářů. RUP bývá často dáván jako protipříklad agility, jako monstrózní metodika orientovaná na vytváření obrovského množství dokumentů.




Obr. 3

## Starý dobrý RUP

Klasický RUP opravdu definuje přes 80 projektových artefaktů, 27 projektových rolí, ... a jeho nasazení pro malé projekty opravdu bylo v této podobě diskutabilní. Na druhou stranu je potřeba si uvědomit, v jaké době RUP vznikal: v době, kdy se drtivá většina projektů řešila tzv. vodopádovým modelem (waterfall development) a 80 % projektů končilo špatně. RUP byl navržen jako továrna na software, jakási výrobní linka, a jeho hlavním cílem je detailně popsat technologický postup tvorby softwarového projektu. Jeho cílem bylo minimalizovat riziko

neúspěchu a je založen zejména na iterativním principu vývoje software (iterative development). V té tobě (vraťme se do poloviny devadesátých let) bylo na pořadu dne vůbec popsat samotný proces spolupráce různých členů týmu a předávání si mezivýsledků jejich práce (workflow) tak. Aby to vůbec fungovalo a bylo aspoň trochu univerzální a efektivní. K tomu RUP sesbíral (ne vymyslel) nejlepší tehdejší znalosti, principy a postupy (best practicies).

## Agilní RUP

RUP jako metodika se vyvíjel, reagoval na změny a vylepšoval se. Dnes mimochodem existuje už ve verzi 7. Jedním z konceptů RUP je tzv. tailoring, tj. přizpůsobení procesu povaze projektu. Tak vznikl RUP pro malé projekty s podstatně menší režií a od něj později odvozený OpenUP. V této souvislosti je dobré si uvědomit několik věcí:

- Většina praktik doporučovaných RUP platí dodnes a hojně se využívá i v agilních metodikách.
- RUP popisuje (takřka) celý životní cyklus projektu, od přípravy zadání až po nasazení, tj. z pohledu modelu AGS se jedná o metodiku typu „disciplined agile delivery“.
- RUP je velmi flexibilní a s jistým zjednodušením lze prohlásit, že OpenUP je tailorovaný RUP, a že například SCRUM je jen hodně upravený a zjednodušený RUP.
- RUP je na tolik rozsáhlý a mnohotvárný, že na něj mají lidé odlišné názory a náhledy podle toho, jakou jeho část nebo v jakém kontextu s ním přišli do styku.

Podstatným rozdílem např. ve srovnání se SCRUP, XP a podobnými metodikami je to, že RUP popisuje několik projektových fází (inception, elaboration, construction, transition), kdežto agilní metodiky se často soustředí pouze na konstrukční fázi (v RUPu přibližně ekvivalentem elaboration + construction). RUP také popisuje devět hlavních disciplín (business modeling, requirements, alaysis&design, implementation, test, deployment, CCM, project management a environment), kdežto mnoho agilních metodik se soustředí na jednu oblíbenou praktiku, okolo které vystaví jednoduchý „process“:

- Architektura funguje, budeme tedy dělat architekturu a máme model-driven architektur (MDA)
- Vizuální modelování funguje, budeme tedy hodně modelovat a máme model-driven development (MDD)

- Testování funguje, budeme tedy hodně testovat a máme test-driven development (TDD)

- Inkrementální vývoj funguje, budeme tedy hodně iterovat a máme feature-driven development (FDD)

Toto jistě trestuhodné zjednodušení tedy vede k úvaze, že v RUP je mnoho agilních principů vlastně „vestavěno“. RUP je tedy tak agilní, jak agilní si jej uděláme, tzn. jak jej budeme tailorovat a jak jej naimplementujeme do svého projektu. Z hlediska použitých praktik je RUP dobře kompatibilní s agile, pouze je potřeba potlačit proces a formální stránku věci. Principy a filosofie zůstávají. Z pohledu RUP stačí tedy jen dostatečně následovat jeden z jeho základních konceptů „tailor the process“. Má to ale háček. RUP se snaží být obecným procesním frameworkem, který lze přizpůsobovat, kdežto ASM propaguje obecné agilní praktiky, které jsou samy o sobě přizpůsobitelné. Pokud tedy nejste procesní expert, bude pro vás druhý způsob snadněji uchopitelný. RUP se snaží současně řešit velké množství věcí, rizik a dalších faktorů složitosti, což jej dělá složitým pro tailoring bez znalosti celé knihovny, tj. hlavní otázkou pro mnohé zůstává, kterou část RUPu mají pro své projekty použít.

# 5 Nástroje IBM Rational

Z kapitoly 2 vyplynulo, že agilní týmy je zobtížné škálovat bez adekvátních nástrojů. Proto v reakci na nástup agilních metod firma IBM v posledních letech uvedla na trh několik nástrojů podporujících agile. Pro tyto nástroje nejprve vytvořila novou platformu Jazz [4], která je primárně orientovaná na změněná paradigmata softwarového procesu a tvorbu nových vývojových nástrojů a jejich vzájemnou integraci.

Z pohledu cílů sdružení Europen je na platformě Jazz zajímavý zejména otevřený standard Open Services for Lifecycle Collaboration [5], který by měl softwarovým týmům pomoci kombinovat nástroje od různých výrobců.

## Rational Team Concert

RTC je nástroj pro týmovou spolupráci softwarového týmu, která do něj vnáší novou dimenzi produktivity. Tato oblast bývá dnes označována jako Collaborative Application Lifecycle Management (Collaborative ALM), pod čímž se skrývá běžné Eclipse IDE rozšířené o týmové funkce: přehled o aktivitách kolegů (team awareness), sledování zdrojů, dashboardy, agilní plánování, sledování úkolů, centrální úložiště kódu s podporou sestavování, podpora agilních procesů SCRUM, OpenUP, Eclipse way (process awareness), základní reporting a další.

Tento nástroj je dokonce k dispozici zdarma pro týmy do 10 osob (resp. pro prvních deset členů vždy).

## Rational Insight

Rational Insight je produkt, který vznikl aplikací principů byznys inteligence (BI) na data vznikající v softwarových projektech. Pomocí technologie IBM Cognos BI reporting bylo vytvořeno množství sond do různých nástrojů, ve kterých vznikají nebo jsou uložená projektová data – například. CVS, Subversion, ClearCase, ClearQuest, maven, buidforge, RTC, MS Visual studio... Pomocí těchto sond jsou informace pumpovány do centrálního datawarehouse, nad kterým lze vytvářet různé reporty, OLAP analýzy a dashboardy. Projektový manažer a potažmo celý tým tak získávají aktuální informace o stavu projektu – burndown grafy, přehledy tiketů, chybovost, trendy... s přidanou hodnotou obvyklých interaktivních BI operací typu drill-down apod.

## Rational MCIF

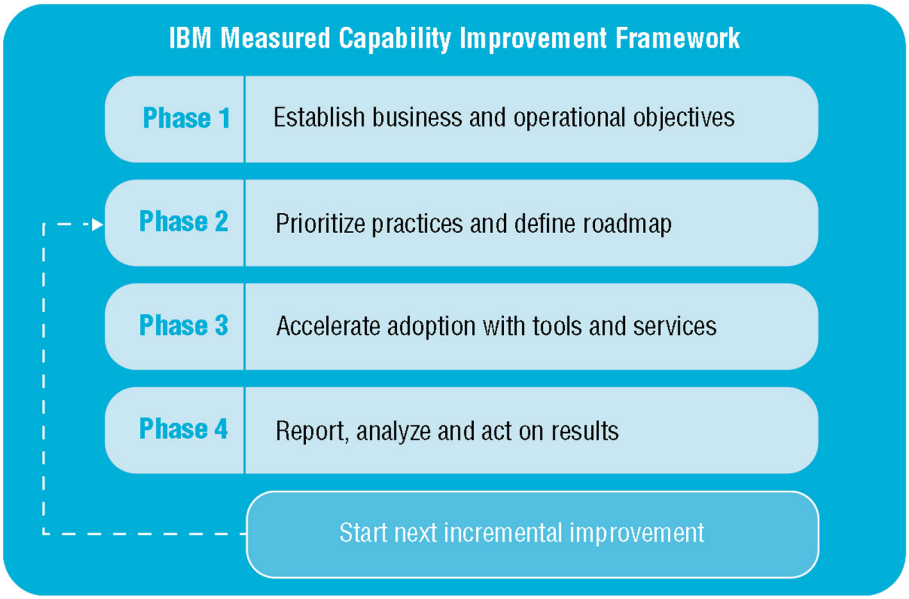
Jak již bylo zmíněno výše, softwarové týmy obvykle nevystačí s malou sadou původních „cool“ agilních praktik a své procesy si doplňují o prvky, které považují za účelné. („Používáme modifikovaný SCRUM“ je jedním z nejčastějších výroků.)

Mnoho společností si vytváří své vlastní disciplinované agilní delivery procesy, kombinují Scrum, praktiky z XP a praktiky z jiných metodik (často i nevědomě) jako je AM, AD nebo FDD, což je plně v souladu s agilními principy retrospekce a neustálého zlepšování týmu. Přístup „pokus–omyl“ při hledání té správné kombinace praktik a nástrojů není efektivní. Velké společnosti řeší související otázku „Kde začít?“ a kam směřovat investice do zlepšování softwarového procesu s ohledem na maximální ROI.

Strategie kombinování různých užitečných praktik funguje, ale může být ekonomicky neefektivní a časově náročná. Firma IBM proto na základě svých mnohaletých zkušeností získaných na stovkách projektů vypracovala metodický rámec, který dokáže pomocí dotazníku (formou oblíbeného pokeru) „změřit“ stav týmu a doporučit mu hlavní oblasti pro zlepšení.

IBM Measured Capability Improvement Framework (MCIF) je strukturovaný přístup pro soustavné a měřitelné zlepšování softwarového procesu zaměřený na agilní metodiky. Jedná se o cyklus trvalého inkrementálního zlepšování postavených filosoficky odpovídající CMMI Level 5 nebo Six Sigma. Základem každého cyklu je assesment, kdy tým ohodnotí dosavadní praktiky a postupy. Metodika MCIF potom „spočítá“ resp. změří různé aspekty vývojového procesu a vyjádří je číselným vektorem, který je obvykle prezentován formou pavučinového grafu. Týmy tak mohou sledovat svoje zlepšování.

Pro agilní tým MCIF poskytuje plán pro trvalé zlepšování. Z praxe IBM Rational vyplývá, že týmy, které explicitně sledují svůj postup osvojování zlepšení, mají větší uspěšnost než týmy, které to nedělají.




Obr. 4

### Rational Quality Manager

RQM je nástroj pro agilní plánování a řízení testů (test management). Je postaven rovněž na platformě Jazz a vhodně doplňuje RTC o funkce potřebné pro testování a QA. RQM je navržen pro agilní metodiky, tj. například defekty jsou vlastně jen jiným typem úkolů (work items), testování je transparentní pro celý tým, je zde podpora pro kontinuální integraci, reporting-retrospekci a další agilní principy.

## 6 Závěr

Dělit metodiky na agilní a „ty druhé“ postupně přestává mít smysl. Význam slova „agile“ se posunul a změnil. Agilita se stala módním filosofickým přístupem, který prolíná napříč organizacemi i mnoha postupy softwarového vývoje. Agilní jsou dnes všichni, což ale přináší i řadu nových problémů, zejména při

nasazení „původních“ malých agilních metodik na velké projekty. Ukazuje se potřeba zavádět disciplinovaný agilní delivery procesy postihující všechny etapy projektu a podpořit adopci agile vhodnými nástroji pro tzv. collaborative application lifecycle management (CALM). U větších týmů je nutné podporovat zmenšování neproduktivní režie sdílení statusu, podporovat retrospekci a klasické malé agilní metodiky rozšířit o standardizované způsoby agilního delivery kompletních projektů.

V kontextu portfolia společnosti IBM, jednoho z vedoucích vendorů agilních instrumentů, je vhodné věnovat pozornost následujícím řešením:

- Rational Team Concert, nástroji na podporu týmového vývojového procesu s orientací na agilní metodiky
- Rational Insight, nástroji na podporu vytěžování informací z projektových dat a podporu rozhodování
- Rational MCIF, metodickému frameworku, který pomáhá softwarovým týmům v sebereflexi (nebo chcete-li retrospekci) a doporučuje jim, jaké oblasti zlepšovat

Nesmíme však zapomínat, že agilita je cesta a ne cíl. Cílem není být agilní, ale být neustále lepší a lepší.

# Literatura

[1] West Dave, Hammond Jeffrey S.: *The Forrester Wave: Agile Development Management Tools, Q2 2010.* May 2010.

[2] Royce Walker: *Improving Software Economics.* Whitepaper, IBM Corp. May 2009.

[3] Agile Manifesto. http://agilemanifesto.org/ February 2001.

[4] Jazz project. http://www.jazz.net

[5] Open Services for Lifecycle Collaboration. http://open-services.net

[6] Amber Scott W.: *The Agile Scaling Model (ASM): Adapting Agile Methods for Complex Environments.* Whitepaper, IBM Corp. December 2009.

[7] Amber Scott W.: Surveys Exploring The Current State of Information Technology Practices. http://www.ambysoft.com/surveys/

[8] State of Agile Development Survey. VersionOne 2009 http://www.versionone.com/pdf/2009_State_of_Agile_Development_Survey_Results.pdf

# Obsah