Úvod

Motivace pro započetí projektu

Motivace pro vznik projektu:

- **vydělat peníze**
  - zadání od zákazníka
  - mnoho projektů velkých zaběhlých firem
    - Microsoft
- **vyřešit nějaký problém**
  - pokud se podaří vytipovat praktický problém, je to velmi dobrý základ úspěchu
    - Google, GNU, ...
- **pro zábavu**
  - mnoho úspěšných projektů začalo takto
    - Linux, Facebook, ...

Jasný cíl

Základem úspěšného (softwarového) projektu je jasně definovaný, stručný a jasný cíl.
Mělo by být možné ho vyjádřit jen několika slovy.
Příklady:

- Google - Organizovat všechny známé informace.
- Apple - Vyrábět jednoduché počítače pro každého.
- GNU - Vytvořit volně šiřitelnou alternativu systému Unix.
- IBM/Mainframe - Vyrábět maximálně spolehlivé systémy.
- Nvidia - Vyrábět výkonné výpočetní čipy.
- Microsoft - ???.

Jasně definovaný cíl později umožní plně se soustředit na jednu věc a lépe využít dostupných prostředků.

Postup řešení projektu

- rešerše = průzkum "trhu", zjistit state-of-the-art
- ujasnit si cíle projektu
  - "Zorganizovat všechny informace.-směrm filozofie projektu
  - "Implementovat indexovací algoritmus PageRank.-konkrétní proveditělný krok
- priority projektu
  - maximální výkon
  - maximální flexibilita
  - maximální spolehlivost
- mít jasno v tom, co projekt řešit **nebude**
- prezentace projektu pro získání zdrojů
  - (předložení návrhu smlouvy zákazníkovi)
  - zveřejnění hlavního dokumentu na webu (open source nebo vědecké projekty)
  - vypracování žádosti o grant
  - prezentace manažerům firmy

Pragmatické řešení projektu

Nyní je nutné co nejefektivněji plnit dílčí cíle projektu.
Držíme se dvou pravidel:

- **využíváme všech dostupných prostředků**
- **neděláme zbytečnou práci navíc**

Vodopádový model

Dříve se používal následující postup vývoje softwarových projektů:

- definice problému
- specifikace požadavků
- návrh
- implementace
- integrace a testování
- údržba

Ten ale příliš nevyhovuje realitě rychle se měnících požadavků.

Agilní programování

Zejména v případě webových projektů je nutné co nejrychleji vytvořit první funkční verzi.

- první verze Facebooku vznikla za pár týdnů (2-6)
- než sepíšete specifikaci, konkurence může provozovat primitivní implementaci vámi vyvíjené služby a získávat klienty, které už nepřetáhnete ani s mnohem kvalitnějšími službami

Metodiky jako extrémní programování specifikaci vůbec nepoužívají.
Václav Kadlec, Agilní programování, Computer Press 2004

Funkcionalita

Není umění sepsat všechny funkce, které nás napadnou, ale vybrat ty podstatné!!!

Pro první verzi produktu chceme najít nejmenší množinu těch nejdůležitejších funkcí.

- méně znamená více
- design je dokonalý, když už nelze nic odebrat
- "Ideální vůz Formule 1 se rozbije hned za cílovou čarou."
  - konstruktéři se soustředí na výkon, ne na spolehlivost

Funkcionalita

Nepřemýšlíme jen o tom, jaké funkce přidat, ale také o tom, jaké odebrat.

- má-li být systém maximálně bezpečný, odebereme všechny funkce, které mohou bezpečnost narušit, a které nejsou nezbytně nutné
- dobré příklady
  - iPhone bez multitaskingu
  - jednoduché malé a jednoúčelové unixové programy
- špatné příklady
  - linuxové distribuce se spoustou ne úplně funkčních programů
  - spotřební zboží se spoustou zbytečných funkcí
- pokud nedokážu danou funkcionalitu implementovat téměř perfektně, raději ji nepřidávám
- kvalitní implemetace i jednoduché funkce může být velice náročná

Architektura a vývojové nástroje

Nyní začínáme **přemýšlet** nad implementací.
Podle pravidla "neděláme zbytečnou práci"a podle priorit projektu vybereme:

- operační systém
- programovací jazyk
- dostupné knihovny
- použité algoritmy
- vývojové prostředí a nástroje
- u každého bodu zjistíme všechny možnosti a řádně vysvětlíme naši volbu
- to vše zdokumentujeme pro případ, že bude později nutné něco měnit
  - pak se stačí jen podívat na zbylé možnosti
- volíme osvědčěné produkty
- pokud to není nutné, nepoužíváme absolutní novinky
- ty mohou výrazně ohrozit úspěšné vypracování projektu
- není pravda, že co je nové, je vždy lepší, než to staré

Návrh softwarového projektu

Celý systém rozdělíme na menší moduly (např. funkce) a popíšeme vztahy mezi nimi.

- moduly nemají být příliš velké
  - pak jsou komplikované
- nemají být příliš malé
  - pak jsou komplikované vztahy mezi nimi

Návrh softwarového projektu

Modul je dobře navržený, pokud:

- ho dokážu stručně a jasně pojmenovat.
  - tzn. dokážu stručně a jasně popsat, co má dělat
  - to je podobný princip jako stručný a jasný cíl projektu
- má jen pár vstupních parametrů
  - ideálně ne více, než 7
- funguje nezávisle na ostatních modulech
  - tzv. ORTOGONALITA
  - špatný příklad: "Tuto funkci lze volat jen po volání fcí. A a B. Následně je nutné volat funkce E a F. Byla-li již volána funkce G, nelze použít parametr Z."

Implementace

Vytvoříme co nejjednodušší implementaci, která vyhovuje zadání.
Nejprve implementujeme nejpodstatnější funkce.
VYHÝBÁME SE PŘEDČASNÉ OPTIMALIZACI.

- pro každý algoritmus je potřeba mít téměř bezchybnou a co nejjednodušší implementaci
- ukáže-li se, že není dostatečně výkonná, pak přemýšlíme o optimalizaci
- původní implemetaci nezahazujeme
  - porovnáváme vůčí ní efektivitu optimalizace
  - porovnáváme, zda obě implementace dávají shodné výsledky

Čistota kódu

Čistotou kódu máme na mysli:

- kód je dobře čitelný a srozumitelný
- kód se snadno upravuje a doplňuje o další funkce

Za účelem dobré čitelnosti:

- dodržujeme obecná pravidla a nesnažíme se být originální
- používáme návrhové vzory

Pročišťování kódu - refaktorizace

- ideální implementace vzniká iterativně
  - většinou až zpětně vidíme, jak by šlo algoritmus napsat lépe/čistěji
- pokud kód dobře funguje necháme ho být a implementujeme další důležité funkce
- v jistém okamžiku může "nečistota"kódu ztěžovat implementování nových funkcí
- přistoupíme k očistění = refaktorizace
  - existuji nástroje pro usnadnění tohoto procesu

Dokumentace kódu

Ideální kód je srozumitelný sám o sobě a dokumentaci nepotřebuje.

- v dokumentování kódu znamená, žé méně je více
- dokumentace kódu **nesmí** být zastaralá oproti kódu
  - pak je kontraprodutkivní
- píšeme-li hodně dokumentace, pak musíme vynaložit velké úsilí při každé změně kódu
- proto je dobré komentovat jen větší celky
  - moduly, třídy, funkce metody
- případně vysvětlit použité algoritmy

Dokumentace kódu

Popis modulů:

- k čemu modul slouží
- vstupy (parametry funkce)
- za jakých podmínek může být modul spuštěn (volání funkce)
- výstup modulu (co funkce vrací)

Je-li dokumentace nepřehledná, může to značit špatně navržený modul:

- není přesně jasné, co má modul dělat
- má příliš mnoho vstupů
- složitě interaguje s okolím
- není jasné, co vrací

Testování kódu

- testování kódu je nezbytné pro ověření jeho kvality
- během vývoje píšeme tzv. **unit test**
- ty mohou výrazně urychlit vývoj kódu a zvýšit jeho kvalitu
- platí pravidlo, že 80% je implementováno za 20% času
- s tím, jak kód přibývá, programátor tráví stále více času odstraňováním chyb
- na odstranění chyby zabere nejvíce času její lokalizace
- správné testování kódu značně urychleju nalezení chyb a vede k budování kódu na pevných základech

Psaní unit testů

- ve chvíli, kdy dokončím jeden celek kódu (funkci), napíšu kód, který testuje, zda funkce pro dané vstupní parametry vrací správné výsledky
- pokud to provedu dobře, mám jistotu, že funkce funguje správně
- když budu následně hledat nějakou chybu v kódu, nemusím jí hledat v této funkci, čímž se mi zjednodušuje lokalizace chyby

Psaní unit testů

Nic není tak ideální:

- prakticky žádnou funkci nelze otestovat kompletně v rozumném čase
  - kombinatorická epxloze
- je nutné vytipovat nejdůležitejší vstupní parametry
  - to může být velký problém
- ne vždy je jasné, jaký má být výsledek funkce

Spouštění testů má být automatizované (jednoduché).

Výhody unit testů

Mám-li svůj kód dobře pokrytý testy mám následující výhody:

- při každé úpravé kódu snadno zjistím, zda jsem neporušil funkcionalitu
- testy samotné mohou sloužit jako ukázka použití dané funkce/modulu
  - Test driven development

Vložím-li přiměřené množství energie do psaní testů, bohatě se mi to vrátí v dlouhodobě efektivnějším programování.
Ani s testováním to nepřeháníme. Náklady na perfektně otestovaný produkt mohou až 10x překročit náklady na vývoj produktu samotného - např. automobilový průmysl.

Uvolnění první verze

- dotažení projektu do úspěšného konce (tzv. production run) je velice náročné
- ne zcela kvalitní produkt může uživatele odradit, a pak je těžké získat je zpět
- proto do první verze implementujeme jen nezbytné minimum, ale snažíme se o dobrou kvalitu
- můžeme uvolnit beta verzi a použít dobrovolníky pro pomoc při testování
- některé produkty jsou určitou dobu nasazeny soubežně s již funkčním systémem a provnávají se výsledky
- ne zcela doladěné funkce je lepší v první verzi vypnout a vydat je až s updaty

Uživatelské dokumentace a prezentace výsledků

- cílem našeho projektu bylo vyřešit určitý problém tak, aby **náš uživatel vyřešil stejný problém ve zlomku času**, který jsme našemu projektu věnovali my.
  - většina softwarových projektů je navrhována za účelem ušetření času lidí
- to je nutné mít na paměti při psání uživatelské dokumentace a při prezentaci výsledků
- obojí musí být stručné a jasné
  - sebelepší program nebo vědecký výsledek je k ničemu, když ho nikdo nechápe

Uživatelské dokumentace a prezentace výsledků

Dobrý způsob jak psát dokumentaci pro uživatele:

- najít si typické ale JEDNODUCHÉ úlohy, které budou uživatelé řešit
- sepsat tutoriál, kde ukážeme jak, tyto úlohy řešit
- tak si uživatel ozkouší náš projekt a pochopí základní principy na úloze, která ho sama zajímá
- dále sepíšeme podrobnou referenční příručku, kde si uživatel dohledá potřebné detaily
- případně sepíšeme FAQ
- pokud se nás někdo dotazuje na něco, co nepochopil z dokumentace, neodpovídáme osobně, ale snažíme se dokumentaci upravit tak, aby byla lépe pochopitelná

Údržba

Údržba spočívá v:

- opravování chyb a pročišťování kódu
- doplňování funkcí
  - funkce nepřidáváme bezhlavě
  - dáváme pozor, aby nenarušily dobrou funkcionalitu našeho produktu
  - některé firmy přidávají do produktů zbytečné funkce, jen aby mohly prodávat nové výrobky
  - komunikujeme s uživateli
  - samotná komunikace s uživateli ale nestačí - uživatelé nejsou vždy odborníci na to, co je možné implementovat
  - zejména zde hledíme na to, aby se náš produkt dobře a snadno používal
  - je škoda, když se dobře fungující produkt pokazí novými funkcemi

Dobře navržený produkt může dojít ke kompletnímu naplnění - např. TEX.
Někdy se můžeme rozhodnout přepracovat projekt zcela od začátku.

Literatura

Algoritmizace

- skripta ZALG, Virius
- Robert Sedgewick, Algoritmy v C, Softpress, 2003.
- Donald E. Knuth, Umění programování, Computer Press, 2008.

Programování

- Steve McConnell, Dokonalý kód, Computer Press, 2005.
- Andrew Hunt, David Thomas, Programátor pragmatik, Computer Press, 2007.
- Rudolf Pecinovský, Návrhové vzory, Computer Press, 2007.
- Martin Fowler, Refactoring - Zlepšení existujícího kódu, Grada 2003.
- Václav Kadlec, Agilní programování, Computer Press, 2004.
- Eric S. Raymond, Umění programování v Unixu, Computer Press, 2004.