Softwarové inženýrství

2. přednáška

Radek Janoštík

Univerzita Palackého v Olomouci

2. 10. 2025

Outline

- Softwarový proces
- (tradiční) Modely softwarového procesu
- Agilní metody
- Doporučená četba

Softwarový proces

- = soubor aktivit a procesů potřebných k dokončení softwarového produktu.
  Složen z 4 částí:

1. **Specifikace** – zákazník a sw. inženýři definují, jak má výsledný produkt za jakých podmínek fungovat
2. **Návrh a vývoj** – návrh (architektury), plánování a programování
3. **Validace** – ověření, zda vyvinutý SW splňuje specifikaci
4. **Evoluce** – úpravy SW měnícím se potřebám zákazníka/trhu

- Různé projekty vyžadují výše zmíněné body v různé kvalitě
  - Příklady (řízení letového provozu, eshop, osobní projekt, . . .)

Specifikace software

- = proces zjištění a definování, jak má SW fungovat
- určení omezení (a podmínek) fungování a vývoje SW
- Nejdůležitější část SW vývoje (lavinový efekt)
- Softwarový inženýr by měl zanalyzovat hlavní doménu (obor) softwaru
  - Znát terminologii
  - Porozumět základním principům a motivacím
- Po zjištění požadavků a cílů následuje jejich formální definice
  - Výstup: Dokument specifikace požadavků (DSP)

Specifikace software – základní fáze (1 / 2)

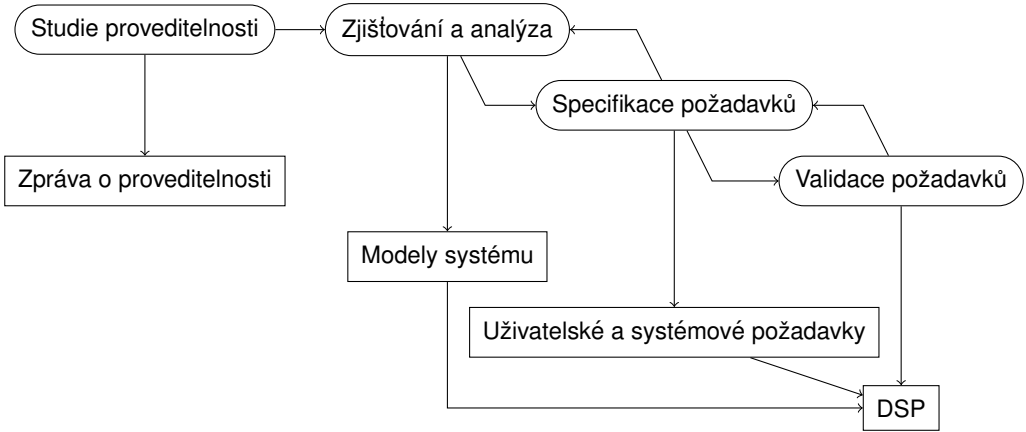
1. Studie proveditelnosti
   - Jsou zákazníkovy potřeby uskutečnitelné s HW a SW možnostmi?
   - Je možné SW vyvinout s daným rozpočtem?
   - Neměla by dlouho trvat, neměla by být drahá $\Rightarrow$ bude se vůbec pokračovat?
2. Zjišťování a analýza požadavků
   - Diskuze s (budoucími) uživateli, vedoucími, manažery
   - Pozorování existujících systémů
   - Vývoj systémových modelů/prototypů SW (snadnější porozumění)

Specifikace software – základní fáze (2 / 2)

3. Specifikace požadavků – sumarizace předchozích fází
   - Výstupem je dokument s definovanými požadavky
   - *Uživatelské požadavky* – abstraktní tvrzení s požadavky na systém z pohledu koncových uživatelů
   - *Systémové požadavky* – detailní popis funkcionality poskytované softwarem
4. Ověření požadavků – kontrola předchozích fází
   - Jsou požadavky realizovatelné?
   - Je specifikace kompletní?
   - Zjištěné chyby je potřeba opravit

- Zmíněné fáze nejsou striktně odděleny
- Jedna ovlivňuje druhou, zjištěné nedostatky aktualizují předchozí

Specifikace software



Obrázek: Fáze specifikace software

Návrh a vývoj

- = Ze specifikace vytváříme spustitelný systém
- Není čistě o programování
- Zvolení vhodné architektury aplikace
- Určení datových formátů, struktur a modulů
- Abstraktní model:

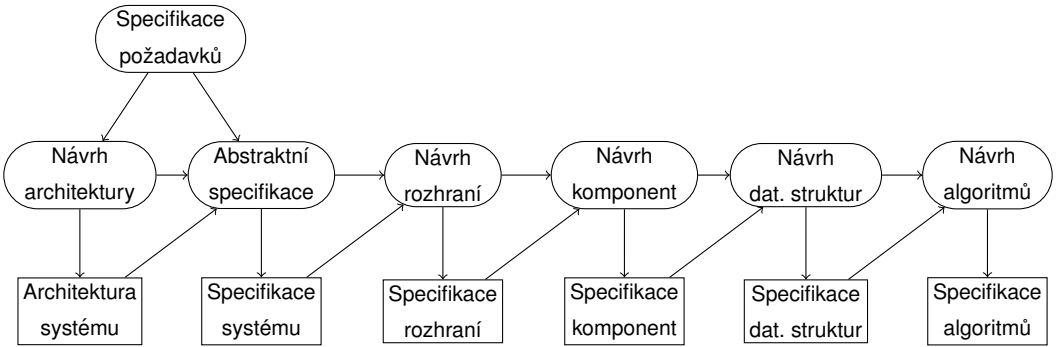
1. **Návrh architektury** – Určení subsystémů a jejich vztahů
2. **Abstraktní specifikace** každého subsystému a jeho služeb a vymezení činnosti
3. **Návrh rozhraní** – mezi každým subsystémem
   - Musí být zřetelné a vyčerpávající
   - Možnost používat subsystém bez jeho vnitřní znalosti (blackbox)
   - Řádně zdokumentované

Návrh a vývoj

4. **Návrh komponent** – určení zodpovědnosti (komponenta ⇔ služba)
5. **Návrh datových struktur** – detailní návrh a určení (+ dokumentace!) používaných datových struktur
6. **Návrh algoritmů** – volba správných algoritmů, postupů metod

- Pouze obecný popis ⇒ v praxi se může lišit
  - Může záležet na konkrétním modelu
  - Poslední dva body často řešeny až při implementaci
- V jakém pořadí programovat komponenty?
  - „Shora“ či „zdola“?
  - Někdy začátek u jasných komponent a postupné propracování ke složitějším (znalost se postupně dostuduje)
- Specifikovat datové struktury a pak se jich (za každou cenu) držet?
  - Nebo finální specifikaci odkládat?

Návrh a vývoj



Obrázek: Fáze návrhu a vývoje

Validace software

- = Ověření, zda se SW chová podle specifikace a splňuje zákazníkova očekávání
- Mělo by probíhat v každé fázi SW procesu
  - Čím dřív se chyba odhalí, tím snadněji se bude opravovat
  - Tím levnější vše bude
- SW by se neměl validovat jako monolit
  - Po částech od nejmenší po větší
- Prvotní testování provádí sami programátoři
  - Asi nikdo neodevzdá naprogramovaný kód aniž by ověřil funkčnost (až na úkoly z C#)
  - = debugování – odhalení (příčiny) chyby, lokalizace, oprava a ověření
  - aktualizování testů $\Rightarrow$ chyba by se neměla opakovat (bude odhalena)

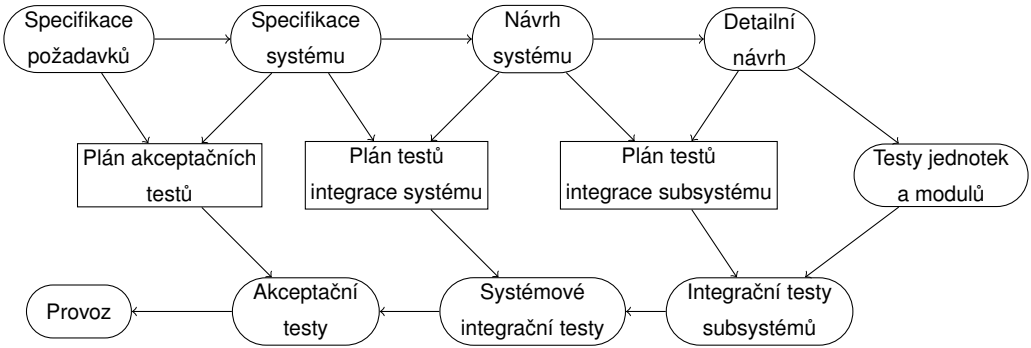
Validace software

1. **Testy jednotek** – testování nejmenších úseků kódu, krátkých komponent
   - Nezávislost testů na ostatních komponentách (fake, mock)
   - Metrika code (test) coverage
2. **Systémové testy (integrační)** – odhalení chyb ve spolupráci mezi komponentami
   - Chyby v rozhraních a datech
   - Kontrola funčních, nefunkcionálních vlastností
   - Kontrola emergentních vlastností
   - Více úrovní (sub-subsystémy, . . .)
3. **Akceptační testy**(*Alfa testy*) – poslední krok před doručením
   - Testování na reálných datech, reálné velikosti (dodány zákazníkem)
   - Odhalení chyb ve specifikaci
   - Testování výkonu
4. **Beta testování** – schválený software nasazen do reálného, testovacího provozu
   - Menší množina uživatelů
   - Vyladění drobných detailů

Absence integračních testů

Obrázek: 2 unit testy, 0 integračních

Validace SW – schéma



Obrázek: Fáze validace software

Evoluce, údržba

- Systémy jsou v provozu většinou delší dobu (až dekády)
- Údržba $\neq$ správa systému, ale programátorské úpravy
- Vždy bude potřeba (v různém rozsahu) nasazený SW upravovat
  - Větší flexibilita než HW
  - Hranice „už nejde upravit“ – těžko se obhajuje, vysvětluje
- Je méně atraktivní než vývoj?
  - Návrh a vývoj je kreativní činnost – většinou baví, jde vidět pokrok
  - Údržba je často chápána jako nezábavné nutné zlo
- Velmi často delší a finančně nákladnější než samotný vývoj
  - $\Rightarrow$ Lukrativnější pro dodavatele
  - Hůře se odhaduje náročnost úprav
- Význam údržby se postupně zvětšuje
  - Málokdy se dnes systémy vyvíjí od nuly
  - Často splývá hranice vývoje a údržby

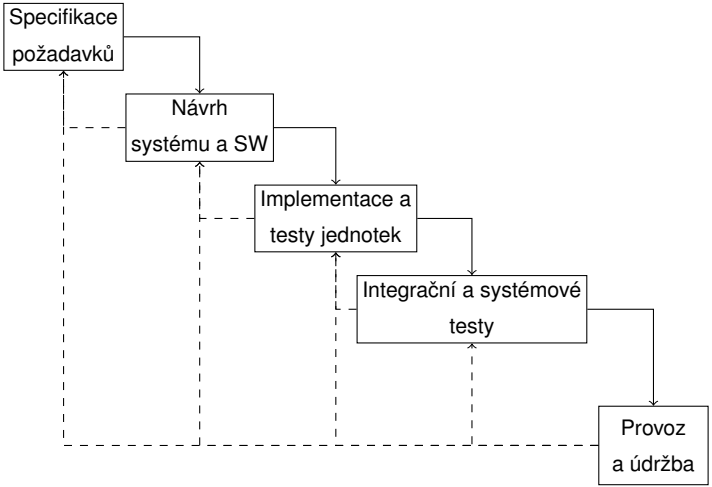
Modely softwarového procesu

- Výše zmíněné fáze by měly být přítomné u každého vývoje
- Pro různé projekty různé váhy daných fází
- Opakování: Softwarové inženýrství = aplikování známých inženýrských postupů do vývoje SW
- Vznik základních (osvědčených) postupů, jak vyvíjet SW
- $\Rightarrow$ Modely (paradigmata) softwarového procesu
- Žádný není „ten nejlepší“
- Je potřeba volit dle specifik projektu
- Je možné kombinovat – jak časově, tak „místně“

Vodopádový model

- Fáze procesu chápany jako samostatné jednotky
  - Každá fáze po dokončení „odepsána“, formálně ukončena
  - V modelu se nepřekrývají, avšak v praxi ano
- Během návrhu odhaleny chyby ve specifikaci, během implementace v návrhu, ...
  - Mírné změny předchozích fází se mohou provádět
  - Je ale drahé udržovat aktuální dokumentace
  - $\Rightarrow$ Uzamknutí (*zmražení*) některých částí, problémy „se vyřeší“ potom
- Výhody:
  - Každá fáze ukončena řádnou dokumentací
  - Měřitelnost pokroku (pro vedení, zákazníka)
- Nevýhody:
  - Brzké *zmražení* $\Rightarrow$ zákazník nedostane, co chtěl
  - Možná špatná struktura – *zmražené* chyby v architektuře obcházeny *workaroundy* v implementaci (často nedokumentováno $\Rightarrow$ zdroj chyb)
  - Zákazník často dostává první verzi až po ukončení vodopádu

Vodopádový model



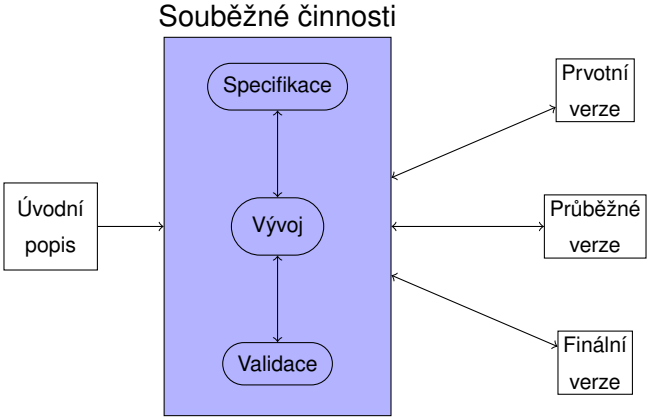
Obrázek: Fáze vodopádového modelu

Evoluční modely

- Rychlé vyvinutí prvotní implementace $\Rightarrow$ ukázána uživatelům/zákazníkům
  - Postupné upřesňování požadavků/cílů
  - Až k vypracování finální verze
- Fáze specifikace, vývoje a validace probíhají souběžně – důležitá zpětná vazba
  - Dva typy evolučních modelů

1. Průzkumný vývoj – zákazník aktivně zapojen do procesu vývoje (zjištění požadavků)
   - Vývoj od srozumitelných celků (kde zákazník „ví co chce“)
   - Postupně se přidává nová funkcionalita, navrhovaná zákazníkem
2. Prototypový(*throwaway*) vývoj – pro lepší pochopení zákazníkových potřeb
   - Úvodní prototyp se zaměřuje na nesrozumitelné části
   - Prototyp se zanalyzuje, vytvoří se z něj specifikace
   - Většinou se pak zahodí a výsledný produkt vznikne reimplementací podle zjištěných požadavků

Evoluční model



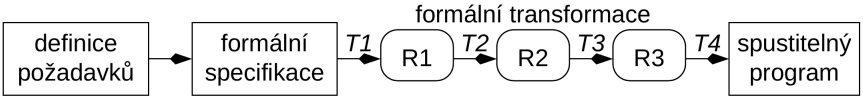
Obrázek: Schéma evolučního vývoje

Evoluční model – shrnutí

- Vhodnější pro menší a střední projekty (do 500 `kloc`)
- Dobrý pro vývoj UI, kombinace s vodopádovým modelem
- Výhody
  - Vyvinutý systém (většinou) více splňuje požadavky zákazníka
  - Možnost vytvářet specifikaci postupně (delší časový úsek, „uležení“, naučení zákazníka)
  - Zákazník průběžně vidí průběžné výsledky – informovanost
- Nevýhody
  - Horší měřitelnost pokroku – v kolika % vývoje již jsme? Kolik ještě bude meziverzí?
  - Architektura systémů může být méně strukturovaná – postupné lepení funkcionality
  - Jednotlivé verze se (většinou) zdražují
  - Nesoulady dokumentace

Formální modely

- Podobné vodopádovému modelu – vychází se z *definitivní* specifikace
- Vytvoří se formální(=matematická) specifikace systému
  - Upřesňování specifikace formálními transformacemi ⇒ spustitelný program



- Postupné přidávání podrobností s formální dokazatelností správnosti
  - Bezchybná formální specifikace ⇒ (dokazatelně) bezchybná implementace
- Problém: V důkazu můžeme udělat chybu – podpůrný SW (model checkers, theorem provers)
- Použití: Kritické systémy, kde potřeba dokázat zákazníkovi bezpečnost

Formální modely – příklad

module *PříkladČítače*

EXTENDS *Naturals*
VARIABLES *cnt*, Čítač.
retval Návratová hodnota.

$TypeInvariant_C \triangleq cnt \in Nat$

$Init_C \triangleq cnt = 0$ Nový čítač se nastaví na nulu.

INTERFACE

cntget $\triangleq$ Zvětší čítač a vrací jeho původní hodnotu.
$\wedge\ cnt' = cnt + 1$ Zvětšíme čítač
$\wedge\ retval' = cnt$ a nastavíme návratovou hodnotu.

THEOREM $Spec_C \Rightarrow \Box TypeInvariant_C$ Typová správnost specifikace.

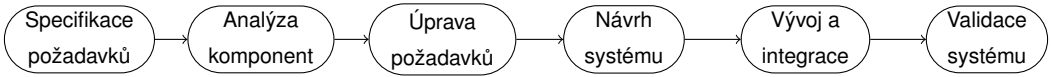
- Nevýhody:
  - Potřeba specializovaných (neprogramátorských) znalostí – potřeba zkušenost
  - Často využívání speciálních programovacích jazyků (nedostatek lidí)
  - Moc se nepoužívá

Znovupoužití existujících komponent

- Často přirozenou součástí předchozích modelů
  - Vývojáři kopírují ověřená řešení z předchozích projektů
  - Pozor na copy&paste chyby, práva
- Ale i ustanovený model softwarového procesu (má svá specifika)
- Obdobná problematika jako s COTS ze systémového inženýrství

1. **Analýza komponent** – zjištění, které komponenty nejlépe vyhoví specifikaci
   - Většinou nenalezneme „Tu pravou“, která 100% splňuje specifikaci
2. **Úprava požadavků** – analýza specifikace na základě dostupných komponent
   - Modifikace požadavků dle komponent
   - Není-li to možné $\Rightarrow$ opakování procesu a zohlednění skutečnosti
3. **Návrh systému** – s ohledem na zvolené komponenty
   - Návrh software, který neumí žádná komponenta
4. **Vývoj a integrace** – vývoj nedostupných částí, „slepení systému“

Znovupoužití existujících komponent



Obrázek: Schéma znovupoužití komponent

Znovupoužití existujících komponent – shrnutí

- Velmi oblíbený model (různé velikosti komponent)
- Výhody:
  - Snižuje množství nového kódu $\Rightarrow$ menší programátorská práce $\Rightarrow$ cena
  - Komponenty by měly být otestované – „zaručeně“ funkční $\Rightarrow$ menší chybovost
  - Rychlejší finální verze
  - Můžeme komponenty prodat vícekrát
- Nevýhody:
  - Nevyhneme se kompromisům $\Rightarrow$ úplné nesplnění původních požadavků
  - Nové verze komponent nemusíme být schopní ovlivnit (externí vývoj)
  - Nejistá životnost a kompatibilita nových verzí komponent $\Rightarrow$ riziko dražší údržby
  - Nízká učící křivka znalosti komponent (rozsáhlé systémy)

Iterativní modely

- *„Změna je jediná životní jistota“* – očekávání změn přímo v modelu
- Hybridní model, časté iterace celého SW procesu

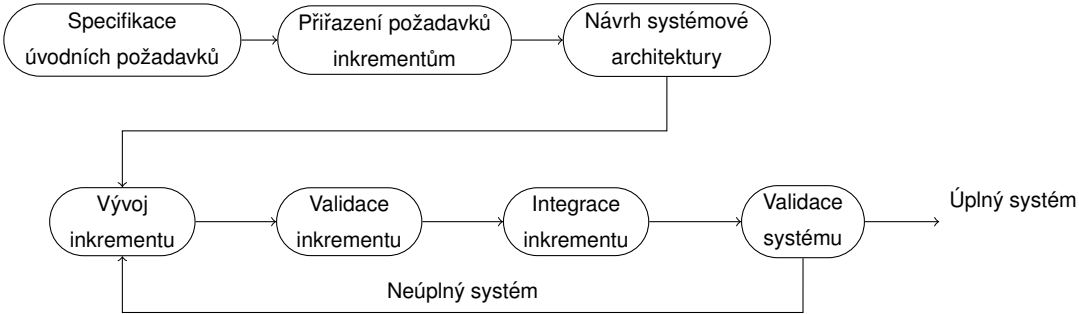
1. **Inkrementální vývoj** – přírůstkový model
   - Specifikace, vývoj a implementace rozdělena do více menších *inkrementů*, oddělený vývoj
2. **Spirálový vývoj** – každá otočka spirály = celý SW proces

- Konečná specifikace je hotová, až je hotov poslední inkrement

Inkrementální vývoj

- Snaha omezit přepracovávání části kvůli změnám požadavků
- Inkrement = malý přídavek
- Možnost odložit některá rozhodnutí – po zkušenosti s částí systému upřesněno
- Určení priority dodávaných inkrementů
  - Začíná se od nejdůležitějších
  - Poslední inkrementy – dodatková funkcionalita

Inkrementální vývoj



Obrázek: Schéma inkrementálního vývoje

Inkrementální vývoj – shrnutí

- Výhody:
  - Zákazník může provozovat systém už od prvního inkrementu
  - Trénování zákazníka ve specifikování požadavků do dalších inkrementů
  - Kritické části nejdéle v provozu $\Rightarrow$ nejvíce otestovány
  - Snížení rizika neúspěchu celého systému (ale přítomnost problematických inkrementů)
- Nevýhody:
  - Určení velikosti inkrementu – dostatečně malé ale ucelená funkcionalita
  - Určení základních služeb používaných všemi inkrementy
- Uplatňování agilních metod

Spirálový vývoj

- Proces není lineární, kde bychom se k něčemu museli vracet
- Každá otočka spirály = 1 fáze SW procesu
  - Uprostřed – všeobecný záměr, studie proveditelnosti
  - Druhá – specifikace požadavků
  - Třetí – návrh systému
  - vnější – implementace
- Celá spirála rozdělena na sektory:

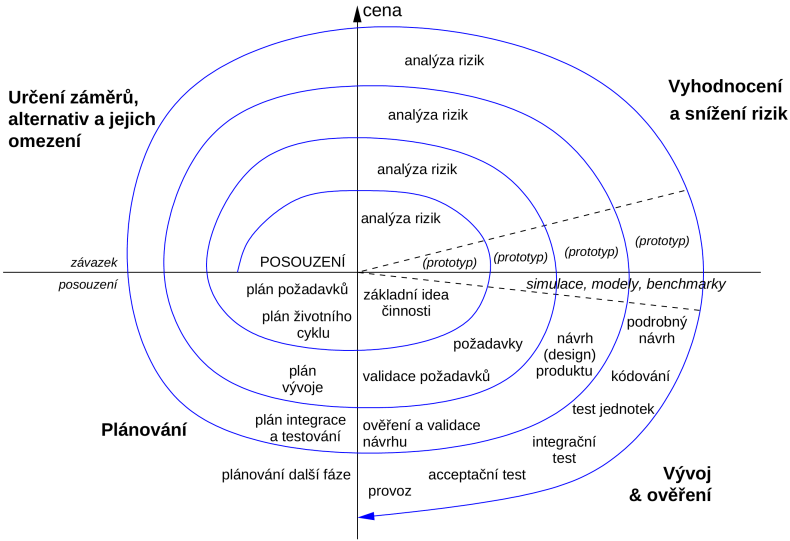
1. **Určení záměrů** dané fáze – funkcionalita, výkon
   - Zvážení alternativních možností realizace
   - Analýza alternativ (cena, časový plán, výhody, nevýhody)
   - Definování rizik alternativ

Spirálový vývoj

2. **Vyhodnocení a snížení rizik** – detailní analýza rizik a možnosti jejich snížení
   - Prototypování, simulace, dotazníky
   - Např.: Je riziko špatné specifikace $\Rightarrow$ vytvoříme prototyp
3. **Vývoj a validace** – samotné vypracování a ověření dané fáze
   - Možné volit rozdílné modely SW procesu (dle rizik a analýzy)
4. **Plánování** – revize předchozích částí
   - Naplánování další fáze (organizace, rozdělení práce, časový plán, nutné zdroje)
   - Konzultace a informování zákazníka

- Jedna z mála metod, která explicitně řeší možná rizika (nutná zkušenost)

Spirálový vývoj



Agilní metody

- Aplikování předchozích modelů na menší projekty může vést na jejich zbytečné zdržování
- Vývoj systému tradičním vodopádem mohl trvat roky
  - Jak moc se změní technologie od začátku projektu po jeho nasazení?
  - *Moorův zákon*
  - Nestrávíme více času formálními záležitostmi než prací na projektu?
- Vývoj software by měl být přece pružnější než vývoj HW
- Tlak na zrychlení dodávek
- $\Rightarrow$ Rozvoj *agilních metod* (90. léta)

Agilní metody – základní principy

- **Zapojení zákazníka** – zákazník by měl být součástí všech fází softwarového procesu
  - Dodává nové požadavky
  - Určuje prioritu a vyhodnocuje kvalitu
  - Problém: je potřeba zákazníka krotit
- **Inkrementální dodávky** – rozdělení funkcionality do menších částí
- **Důraz na lidi, ne proces** – důraz na poznání schopností vývojového týmu
  - Některé věci prostě nechat na členech (vyšší zodpovědnost)
  - I když neznají „big picture“
  - Problém: povahové rysy, sebereflexe, ego
- **Počítání se změnou** – smíření se s možnými změnami
  - Předvídání možných změn v návrhu systému (vs. optimalizace, přehnaná obecnost)
- **V jednoduchosti je síla** – nedělejme SW a celý proces zbytečně komplikovaný
- `https://agilemanifesto.org/`

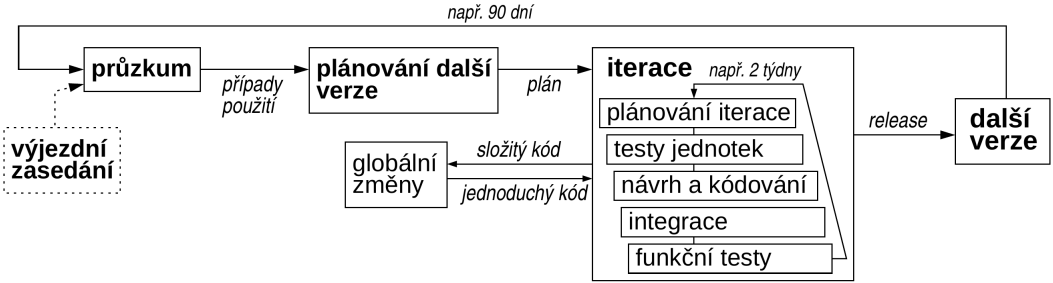
Extrémní programování

- Aplikování dobrých praktik (iterativní vývoj, zapojení zákazníka) na *extrémní* úrovni
- Práce v malých týmech (2-10)
- Často používání techniky párového programování
  - Jeden programuje, druhý kontroluje, komentuje
  - Častá výměna rolí
- *Test-driven development* – pro novou funkcionalitu se nejdříve napíšou testy
  - Až poté se začne programovat funkcionalita
  - Splněné testy == hotovo
- Zrychlené dodávky – nové (vývojové) verze několikrát denně
  - Nasazování nových verzí např. co dva týdny

Extrémní programování – principy

- **Inkrementální plánování** – požadavky zaznamenány v *Story cards*
  - *Stories* nasazeny dle priorit a aktulních možností
  - *Stories* rozděleny vývojáři do úkolů (*tasks*)
- **Malé dodávky** – vydání co nejmenší (a zároveň užitečné) množiny funkcí
- **Jednoduchý návrh** – nedělání zbytečných věcí (obecnost)
- **Refactoring** by měli dělat vývojáři průběžně
  - $\Rightarrow$ nezanášení architektury
- **Kolektivní odpovědnost** – nejsou striktní role (FE vs. BE)
  - Každý může implementovat „cokoliv“
- **Průběžné integrace** – jakmile je *úkol* hotov, je integrován do produktu
- **Zákazník „na place“** – zástupce zákazníka by měl vývojářům dostupný
  - Dodávat zákazníkův pohled, konzultovat

Extrémní programování – schéma



Extrémní programování – shrnutí

- Před zahájením projektu se může uskutečnit „výjezdní zasedání“
  - Často problematické v českém prostředí (víkend, ochota lidí, . . .)
- Párové programování by mělo přinést kvalitní kód (kontrola)
  - Výrazně dražší
  - Těžko měřitelné
- Zákazník i vyšší management nemusí přistoupit na *extrémní* podmínky
- Po dokončení projektu většinou neexistuje kvalitní dokumentace

SCRUM & Kanban

- Nejznámější konkrétní implementace extrémního programování a agilních metod
- Marketingový balast (školení, . . .)
- Ponechávám na samostudium
- Bude zařazeno jako zkoušková otázka

Agilní metody – shrnutí

- V ČR velmi populární (2005+)
  - Kdo nevyvíjí agilně, jakoby neexistoval
  - Často negativní konotace v komunitě (podobně jako vodopád „kdysi“)
  - Parodické reakce na žargon
- Být agilní za každou cenu? (Je to potřeba?)
  - Je toho náš tým schopen?
  - Jsou schopni nést část zodpovědnosti?
- Vznik nepřeberného množství metod
  - Která je ta pravá? Je to funkční?

Agilní metody – uzavření kruhu

- SW inženýrství vzniklo z tradičního inženýrství
- Z tradičního $\rightarrow$ nové progresivní metody
- V průběhu let zastarání (a možná nefunčknost) těchto metod $\rightarrow$ agilní metody SW vývoje
- Jaká je jedna z nejtradičnější klasických inženýrských disciplín?
  - Letecké (a vesmírné) inženýrství
- Jak dnes vyvíjí raketoplány společnost SpaceX?
  - Přesun agilních metod z „dětí“ na „rodiče“
  - Je to funkční? Je to finančně výhodné?
- Kam bude směřovat vývoj?

Doporučená literatura

- Ian Sommerville. Software Engineering (10th Edition). Pearson, 2015. ISBN: 978-93-325-8269-9.
  - Chapter 2 – Software processes
  - Chapter 3 – Agile software development
  - Chapter 16 – Component-based Software Engineering
- Joseph Ingeno. Software Architect's Handbook. Packt, 2018. ISBN: 978-1-78862-406-0
  - Chapter 3 – Understanding the Domain
- Alistair Cockburn. Agile Software Development. Pearson. 2002. ISBN:0-201-69969-9