Zpracoval: houzvjir@fel.cvut.cz

**01. Životní cyklus programového díla, analýza, návrh, implementace, provoz a metodiky vývoje SW. (A7B36SIN)**

# Obsah

# Životní cyklus programového díla

Životním cyklem programového díla se rozumí proces, kterým software projde od zahájení svého vývoje, až po vyřazení z provozu. Tento proces závisí na použité metodice vývoje.

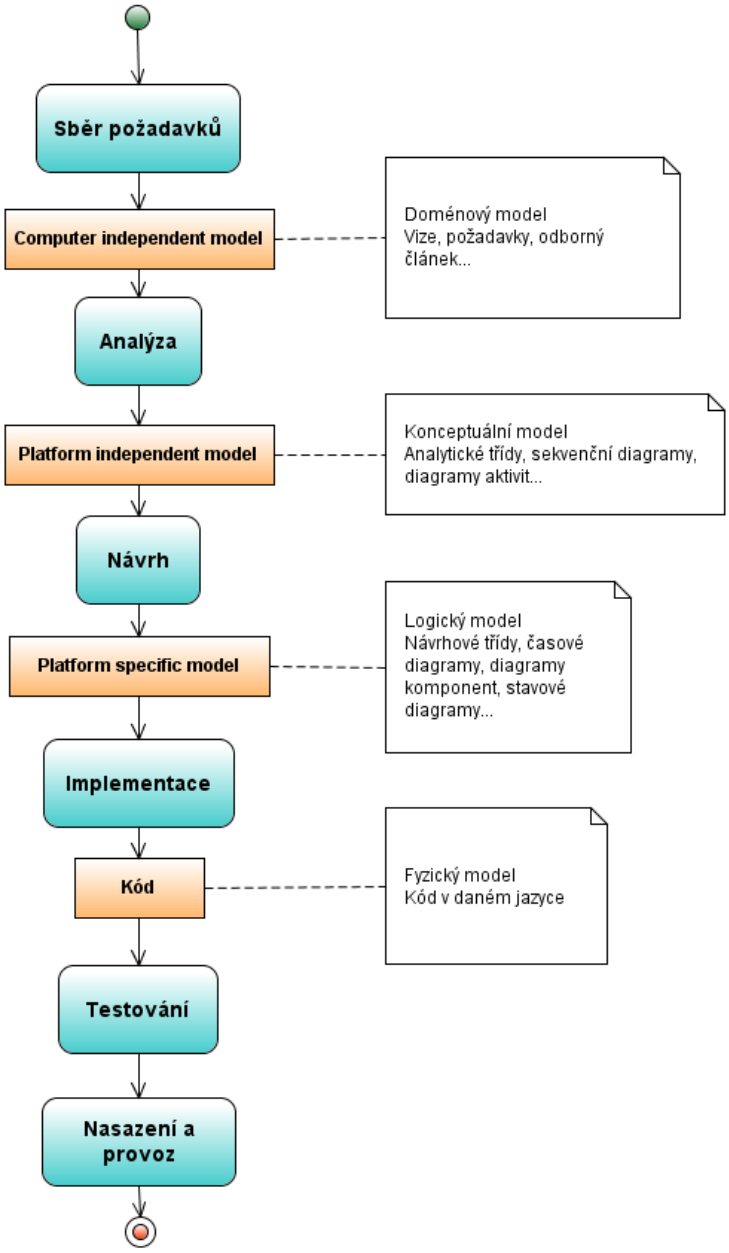
**Příklady modelů životního cyklu softwarového díla:**

- **Model vodopád** – jednotlivé aktivity jsou zpracovány jako nezávislé procesy, které na sebe navazují a vzájemně se neprolínají.
- **Evoluční model** - Tento přístup prokládá jednotlivé etapy realizace kontrolou se zákazníkem a jejich paralelním zpracováním tak, aby se postupné verze realizovaného systému co nejrychleji blížily požadavkům zákazníka.
- **Formální návrh** – tento přístup je založen na vytvoření formálního matematického modelu specifikace systému, který je převeden do programové podoby. Verifikace systému je odvozena z matematického dokazování specifikací.
- **Znovupoužití vývoje –** tento přístup je založen na faktu, že existuje značné množství komponent do nově vytvářeného systému.

Všechny modely mají některé společné základní aktivity, kterými jsou:

- Specifikace
  - Sběr požadavků
  - Analýza
- Vývoj
  - Návrh
  - Implementace
- Testování
  - Validace (SW splňuje požadavky)
  - Verifikace (SW správně implementuje požadované funkce)
- Nasazení a provoz
  - Údržba

Znázornění procesu v diagramu aktivity:




**Sběr požadavků** je klíčovou částí v procesu vývoje software, protože přímo determinuje úspěch projektu. Mottem je "Do the right thing" - aby software dělal skutečně to, co dělat má. Tato aktivita je klíčová zejména proto, že chyby a nepřesnosti objevené již v této fázi se dají poměrně levně opravit. Pokud by se na chyby (odlišnosti od představ zákazníka) přišlo v pozdějších fázích (např. až při implementaci, testování nebo předvádění zákazníkovi) škoda by byla nesrovnatelně vyšší.

Požadavky jsou nejčastěji rozděleny na **funkční** (formulace toho, co by měl systém dělat - funkce systému) a **nefunkční** (omezující podmínky na systém) požadavky, lze je ale kategorizovat i podrobněji.

### Úvodní studie

Velice důležitým materiálem je **úvodní studie** - studie proveditelnosti (feasibility study). Jejím cílem je zhodnotit přínosy, které daný systém zákazníkovi přinese (vzhledem k současnému stavu, případně již používanému systému). Do úvodní studie patří vize, odborný článek, rozpočet projektu, plán projektu, hrubý nástin funkcí systému a uživatelských rolí. Případný neúspěch úvodní studie nám ušetří značné náklady na implementaci, která by byla odsouzena k nezdaru.

- Měla by odpovědět na otázku PROČ?
- Musí odpovědět na otázku: „Vyplatí se projekt řešit?“
- Musí odpovědět na otázku: „Je projekt uskutečnitelný?“ (feasibility study)
- Musí vymezit hranici projektu
- Musí odpovědět na otázku: „Kdo a co bude k řešení zapotřebí?“

Vstupy úvodní studie:

- Požadavky na systém
  - Zadání projektu, deklarace záměru, vize projektu, odborný článek

Výstupy úvodní studie:

- Definice systému
  - Katalog požadavků, definice hranice systému (diagram kontextu, model , jednání), pojmový slovník, ….
- Projektová dokumentace
  - Řešitelský tým (funkce, zodpovědnosti)
  - Návrh řešení: HW, SW, komponenty
  - Seznam úloh a harmonogram řešení
  - Rozpočet – cena HW, licencí za SW, cena vývoje SW a HW (COCOMO).

## Analýza

Analýza by měla odpovědět na otázku CO?

- Musí proto definovat konceptuální model řešeného systému (PIM – Platform Independent Model).
- Musí definovat představu, s jakými daty bude systém pracovat, jaké služby bude systém poskytovat a jak se bude chování systému měnit – jaká bude dynamika systému.
- Musí stanovit podmínky, za jakých je analytická dokumentace akceptovatelná
- Nezabývá se způsobem realizace

Analytický model

- Konceptuální funkční model
- Konceptuální datový model
- Konceptuální dynamický model

### Postup při analýze

- Vstup:
  - úvodní studie, katalog požadavků  (CIM –Computer Independent Model)
- Výstup:
  - Analytická dokumentace – PIM
- Postup:
  - Paralelně zpracuj koncept a projekt

Zpracování konceptu:

- Vstup: CIM
  - Deklarace záměru, odborný článek, katalog požadavků, seznam aktérů, seznam událostí, model jednání, kontext, 1. Verze datového slovníku z úvodní studie)
- Výstup: PIM
  - Konceptuální analytický model (datový, funkční a dynamický model, 2. Verze datového slovníku)

Zpracování projektu:

- Vstup:
  - Seznam úloh, harmonogram (z úvodní studie)
- Výstup:
  - Projektová dokumentace (projektový deník, seznam zdrojů, matice zodpovědnosti, harmonogram, plán testů, akceptační test)

## Datový model

Datově orientovaná analýza

- Seznam událostí, kontext, datový slovník,
- Identifikace dat, která s událostmi souvisí (identifikace základních objektů)
- Identifikace vztahů mezi objekty
- Scénáře jednání (původce, událost, akce, participanti, výstupy – reakce)
- Modelování životních cyklů objektů
- Popis akcí (minispecifikace základních akcí)
- Vychází z představy, že základem IS jsou data. Služby IS slouží pro pořízení a exploraci dat.
- Doporučuje proto nejprve analyzovat požadavky a definovat konceptuální datový model řešeného systému.
- Konceptuální datový model musí postihovat data přicházející přes hranici systému jako vstupní data související s událostmi, dále data, která se v systému ukládají a nakonec rovněž data, která systém produkuje na výstupu.
- Teprve později doplníme model o další části.

Postup datově orientované analýzy:

1. Seznam událostí, kontext, datový slovník
2. Identifikace dat, která s událostí souvisí (základních objektů)
3. Identifikace vztahů mezi objekty
4. Scénáře jednání (původcem událost, akce, participanti, výstupy – reakce)
5. Modelování životních cyklů objektů
6. Popis akcí (minispecifikace základních akcí)

Jak hledat data?

- Analyzujeme odborný článek, vybereme všechna podstatná jména.
- Roztřídíme je do skupin:
  - Kandidáti na typy objektů (entity),
  - Kandidáti na vlastnosti objektů (atributy)
  - Ostatní (kandidáti na aktéry, smetí).
- Analyzujeme seznam událostí, rozpoznáváme data, která s událostmi souvisí.
- Roztřídíme je do skupin:
  - Kandidáti na typy objektů (entity)
  - Kandidáti na vlastnosti (atributy)

### Datový konceptuální model

(zachycení analýzy dat)

Komponenty:

- Typy objektů (entity) – entita = rozlišitelný identifikovatelný objekt
- Vztahy – množiny instancí reprezentujících vztahy mezi 2 a více objekty
- Indikace přidružených objektů – pro vztahy, o nichž si potřebujeme něco pamatovat
- Indikace vztahů nadtyp-podtyp, celek-část, vyjádření vztahu společný – speciální (dědičnost)

Další postup:

- Z datového modelu se snažíme odvodit funkce:
  - Vytvoříme matici CRUD (Create, Read, Update, Delete)
  - Zkoumáme, zda pro každý typ dat existuje odpovídající funkce
- Z datového modelu se snažíme odvodit dynamiku:
  - Pro každý typ dat zkoumáme, zda objekty nevykazují změny stavu

Matice CRUD:

- Řádky odpovídají typům objektů.
- Sloupce odpovídají funkcím.
- V průsečíku je zapsáno, zda funkce C,R,U a /nebo D odpovídající data.
- V každém řádku by mělo někde být vše (některá funkce musí objekt vytvářet, jiná využívat, či rušit).

## Dynamický model

Stavové diagramy

- Slouží k popisu dynamiky systému
- Stavový diagram definuje možné stavy možné přechody mezi stavy, události, které přechody iniciují, podmínky přechodů a akce, které s přechody souvisí
- Stavový diagram lze použít pro popis dynamiky objektu (pokud má rozpoznatelné stavy), pro popis metody (pokud známe algoritmus), či pro popis protokolu (včetně protokolu o styku uživatele se systémem)

# Funkční model

## Funkčně orientovaná analýza

- Začínáme seznamem funkcí – modelem jednání
- Pro každý případ užití navrhneme scénář jednání (průvodce, událost, akce, participanti, výstupy – reakce), diagramy aktivit, příp. diagramy datových toků (návaznosti funkcí)
- Popis akcí (minispecifikace základních akcí)
- Identifikace objektů
- Identifikace vztahů mezi objekty
- Modelování životních cyklů objektů

## Scénáře

Jde o zachycení sledu událostí.

Prvky:

- **Objekty** – znázorněné obvykle jako sloupce
- **Interakce** mezi objekty – orientované šipky mezi objekty
- **Události** – události, které vyvolaly interakci
- **Reakce** – odezvy na události (výstupy)
- **Časová osa** – pro vyznačení sledu událostí

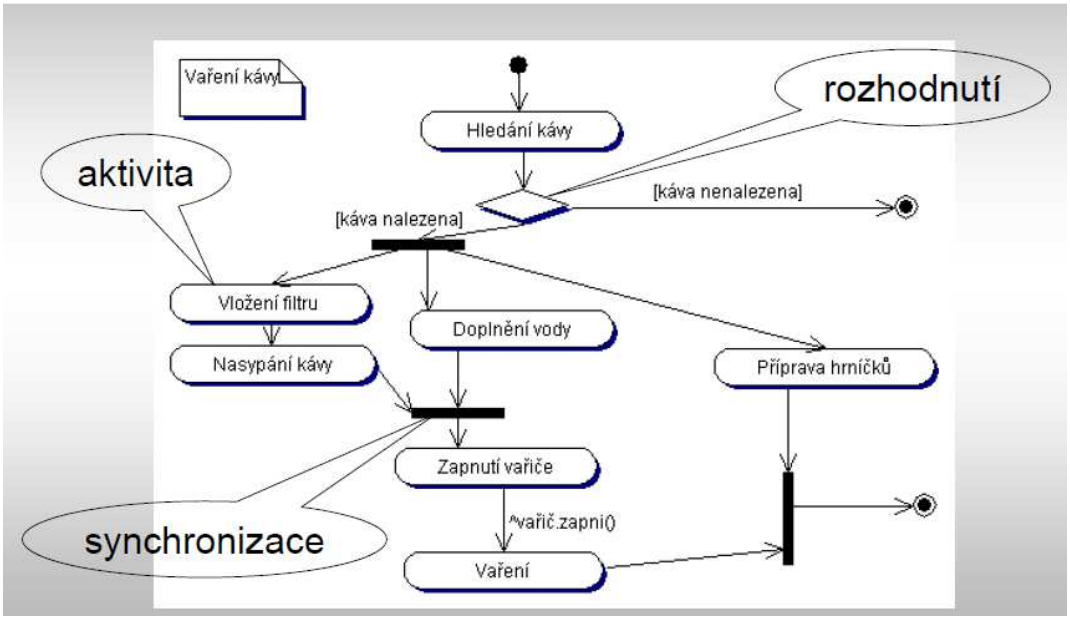

## Diagramy aktivit

- Diagramy aktivity (procesy) a jejich návaznosti.
- Přechody mezi aktivitami jsou vyvolány dokončením akce (jsou synchronní).
- Používají se pro dokumentaci tříd, metod nebo případů použití
- Obecně se mohou použít pro procesní modelování
- Nahrazují v UML neexistující diagramy datových toků.

Diagramy aktivit obsahují tyto prvky:

- Aktivity – činnosti, které modelujeme
- Přechody – po ukončení činnosti se přejde k činnosti jiné
- Objekty – s činností může souviset vytváření nebo konzumace objektů
- Začátek, Konec
- Synchronizační značky (rozvětvení a synchronizace)
- Plavecké dráhy – okruhy zodpovědnosti

Příklad diagramu aktivity

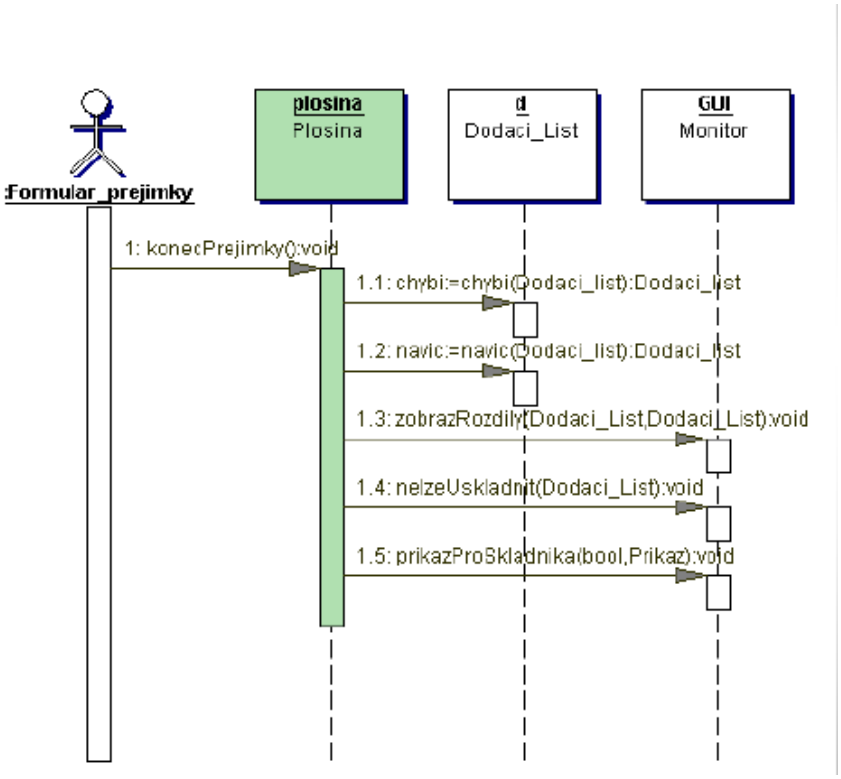


### Diagramy komunikace

Slouží k zachycení komunikace mezi objekty, obsahují tyto prvky:

- Objekty – znázorňují se jako obdélníky
- Interakce mezi objekty - orientovaná spojení mezi objekty
- Zprávy – dokumentace zpráv, které si objekty mezi sebou posílají, včetně parametrů a návratových hodnot – odezvy na události (výstupy)
- Čas - vyznačen číslováním

Příklad:




## Návrh

V této fázi dojde k vytvoření návrhového modelu, který má odpovědět na otázku **jak** se to udělá. Jde tedy o přesnou specifikaci způsobu, jakým budou implementovány funkce, prezentována data atd. Vytváří se **PSM** (Platform Specific Model) - tedy model pro konkrétní platformu. Klíčovým mottem je "Do the thing right", tedy dělat věci správně.

Návrh se skládá z několika kroků, konkrétně je zapotřebí provést návrh architektury systému (lze využít některé z architektonických návrhových vzorů), návrh uživatelského rozhraní, návrh komponent a komunikace mezi nimi, návrh způsobu integrace komponent a testování celku. Součástí návrhu je také volba vývojových prostředí.

Dále je třeba přejít od analytických tříd ke třídám návrhovým, které obsahují všechny důležité podrobnosti (datové typy, návratové funkce, podrobnější relace atd.).

Kroky návrhu:

- Návrh architektury systému
- Návrh uživatelského vzhledu
- Návrh komponent
- Návrh komunikace mezi komponentami
- Návrh způsobu integrace komponent a testování celku

Základní technologická rozhodnutí ve fázi návrhu:

- Architektura systému
- Datové zdroje, přístupové mechanismy k nim
- Distribuce programových modulů, komunikační mechanizmy
- Typy a formy výstupů
- Uživatelská rozhraní
- Vývojové prostředí

Výstupní dokumenty návrhu:

- Architektura systému (HW, SW)
- Popis implementace dat (logický datový model)
- Popis komponent (modulů)
- Projektová dokumentace návrhu

Postup návrhu:

- Úprava (normalizace) konceptuálního modelu
  - Vyloučení multihodnotových a násobných atributů
  - Vyloučení funkčních závislostí (převod modelu do 3-NF, 4-NF nebo 5-NF)
  - Náhrada nebinárních vztahů binárními
  - Náhrada vztahů M:N přidruženými třídami
- Návrh reprezentace typů (entit)
- Návrh reprezentace vztahů
- Návrh reprezentace integritních omezení

Návrh reprezentace:

- Pro každou jednoduchou entitu (typ) navrhneme tabulku, jméno tabulky bude množné číslo jména typu.
- Návrh jmen sloupců pro reprezentaci atributů a odpovídajících domén.
- Doplníme informace o volitelnosti formátu sloupců.

- Z nejčastěji používané unikátní identifikace vytvoříme primární klíč, nebo zavedeme nový identifikační sloupec (OID).
- Pro N-konce vztahů přidáme k tabulce jednoznačné identifikace z tabulky na 1-konci (volitelné vztahy indikují nepovinnost. Současně přidáme odpovídající cizí klíče.
- Pro každý vztah typu nadtyp/podtyp navrhneme reprezentaci (společná tabulka s rozlišovací položkou, samostatné tabulky).
- Pro každý vztah typu celek/část navrhneme reprezentaci (společná tabulka s rozlišovací položkou, samostatné tabulky).
- Pro každý exkluzivní vztah (exkluzivní podtypy) rozhodneme, zda se má řešit společnou doménou, nebo explicitními cizími klíči.
- Doplníme sloupce odpovídající často používaným odvozeným atributům a navrhneme mechanizmus jejich údržby.
- Navrhneme indexy pro často využívané unikátní kombinace, které nejsou realizovány jako primární klíče. Indexy rovněž vytvoříme pro cizí klíče.
- Přidáme definice pohledů (zejména pro nadtypy, podtypy, celky a části).
- Pro generované primární klíče přidáme definice sekvencí pro jejich generování (může být implementačně závislé).
- Navrhneme řešení integritních omezení (použijeme deklarativní relační integritní omezení, nebo navrhneme „triggery“).

## Implementace

V této fázi se vytváří **fyzický model** systému - spustitelný kód systému pro zvolenou platformu. Kostru kódu lze automaticky vygenerovat přímo z některého z **CASE** (Computer-Aided Software Engineering) nástrojů

## Nasazení, provoz a údržba

V momentě nasazení software (rozuměno při jeho dodání zákazníkovi) životní cyklus díla nekončí. Je potřeba zajistit uživatelskou podporu po celou dobu provozu systému a opravovat chyby, které se **vždy objeví** (dokonalý software neexistuje - množství chyb a cena jejich oprav ale závisí na úspěšnosti testování v průběhu všech předchozích vývojových fází). Dále je třeba zajistit případný rozvoj systému - přidání dalších funkcionalit a podobně.

Složitost údržby systému závisí hlavně na tom, zda bylo při jeho tvorbě pamatováno na budoucnost a zda je systém flexibilní. Veškeré detaily by měly být definovány výhradně pomocí konfiguračních souborů, jelikož jejich změna je levnější a pohodlnější než zásah do kódu. U specifických druhů software, kde se požadavky mohou měnit velmi často je potřeba s tímto počítat již v raných fázích vývoje (při analýze a návrhu). Přepisovat celý účetní program s každou novelou zákona by asi nebylo nejlepší řešení.

# Metodiky vývoje SW

Metodika vývoje SW určuje, co se má v které fázi dělat, jaké postupy se mají používat a jaký má být výstup fáze.

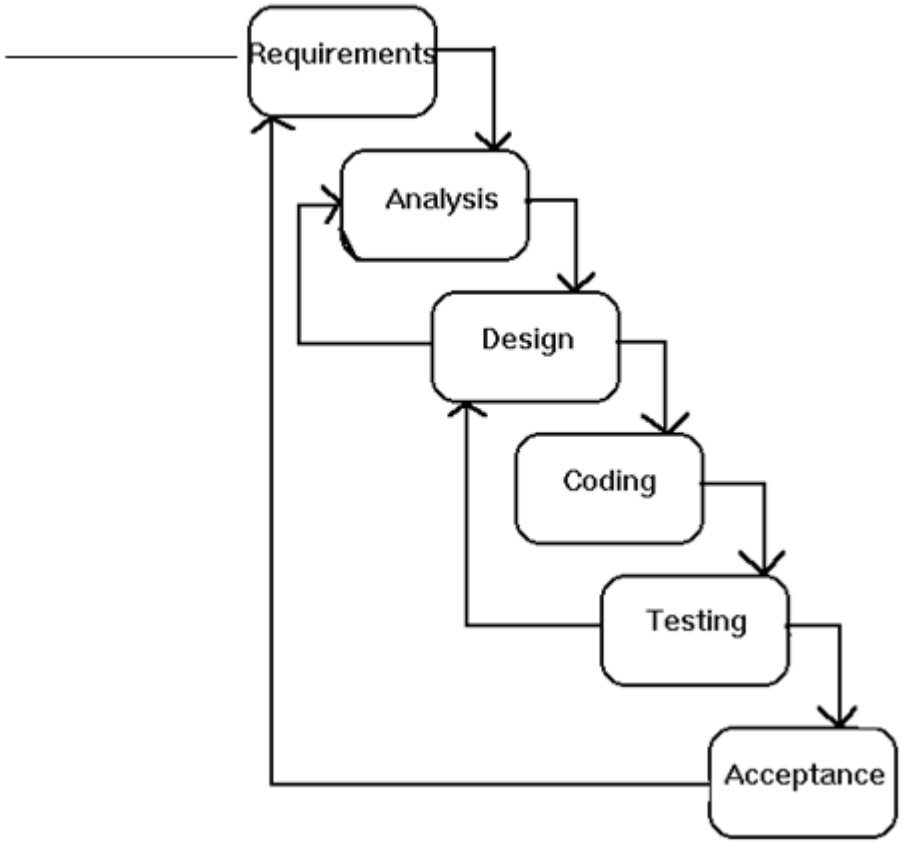
Je třeba dát pozor na důležitý rozdíl mezi modelem životního cyklu softwarového produktu a metodikou vývoje software. Rozdíl je ten, že **model životního cyklu popisuje fáze**, kterými softwarový produkt prochází, kdežto **metodika určuje, co se v které fázi má přesně dělat**, jaké postupy se mají použít a k jakému výsledku se má nakonec dojít.

## Model vodopád

**V literatuře je řazen mezi metodiky vývoje software, ale jedná se o model životního cyklu programového díla.**

Vodopádový model je všeobecně považován za první ucelený model životního cyklu softwarového produktu. Jeho charakteristickým znakem jsou sekvenčně seřazené fáze bez iterací. Mezi jednotlivými fázemi je schvalovací proces, přes který musí všechny dokumenty projít, aby vývoj mohl pokročit do další fáze.

Základní fáze vodopádového modelu jsou následující: - Definice problému, poznání zákazníka a cílové oblasti - Analýza a specifikace požadavků - Návrh - Implementace - Testování a integrace - Provoz a údržba



I přes sekvenční charakteristiku je možné se vracet o jednu fázi zpět. Když například v průběhu implementace zjistíme, že jsme udělali chybu v návrhu, můžeme se vrátit a opravit ho a poté se věnovat opět implementaci. Při přechodu mezi jednotlivými fázemi je důležité, aby všechny dokumenty prošly schválením. **Hlavní nevýhodou** je fakt, že o několik kroků zpět se můžeme vracet až ve fázi údržby, což je z dnešního hlediska naprosto nepřijatelné (většina změn požadavků se objeví daleko dříve).

V 80. letech byl tento model podroben kritice, byly zformulovány základní nedostatky:

- reálné projekty zřídka kdy sledují jednotlivé kroky v předepsaném pořadí
- pro zákazníka je obtížné vyjádřit předem všechny požadavky
- provozuschopná verze bude k disposici až po delší době (může být už zastaralá)
- často dochází k prodlevám

Přesto však i v současné době je pro řešení řady i velkých projektů model vodopádu používán.

## Model prototypování

Vývoj pomocí vytváření prototypů je takový přístup k vývoji software, kde dochází k vývoji neúplných verzí software, tzv. prototypů.

Slouží k pochopení požadavků na systémy, které nejdou dobře specifikovat předem. Prototypy, na kterých mají být modelovány nějaké vlastnosti systému, mohou být dělány s vědomím, že budou zahozeny. Tento model lze použít pro menší systémy.

Základní principy prototypového přístupu:

- Není samostatným a kompletním přístupem metodiky vývoje, ale spíše přístup k jednotlivým částem větších tradičních metodik vývoje software (tj. přírůstková metoda, spirála, nebo RAD - Rapid application development).
- Snaha snížit nebezpečí projektových rizik rozdělením projektu na menší části a zjednodušit tak možnost změn v průběhu procesu vývoje.
- Uživatel je zapojen v celém procesu vývoje, což zvyšuje pravděpodobnost přijetí konečné implementace uživatelem.
- Malé ukázky systému jsou vyvíjeny iterativním procesem, dokud se prototyp nevyvine tak, že splňuje požadavky uživatele.
- Většina prototypů je sice vyvíjena s tím, že budou vyřazeny, ale v některých případech je možné pokročit od prototypu k funkčnímu systému.
- Aby se předešlo vývoji, který řeší jiný problém, než bylo zadáno, je třeba pochopit základní business problematiku.

## Přírůstkový model

Kombinuje lineární model (vodopád) s iterativní filosofií. Produkt se vytváří po částech (přírůstcích),
které jsou funkční (na rozdíl od prototypu). První přírůstek je označován jako jádro. Model je

vhodný pro malý tým a velký úkol, který se dá zvládnout předem po částech, v domluvených termínech.

## Spirálový model

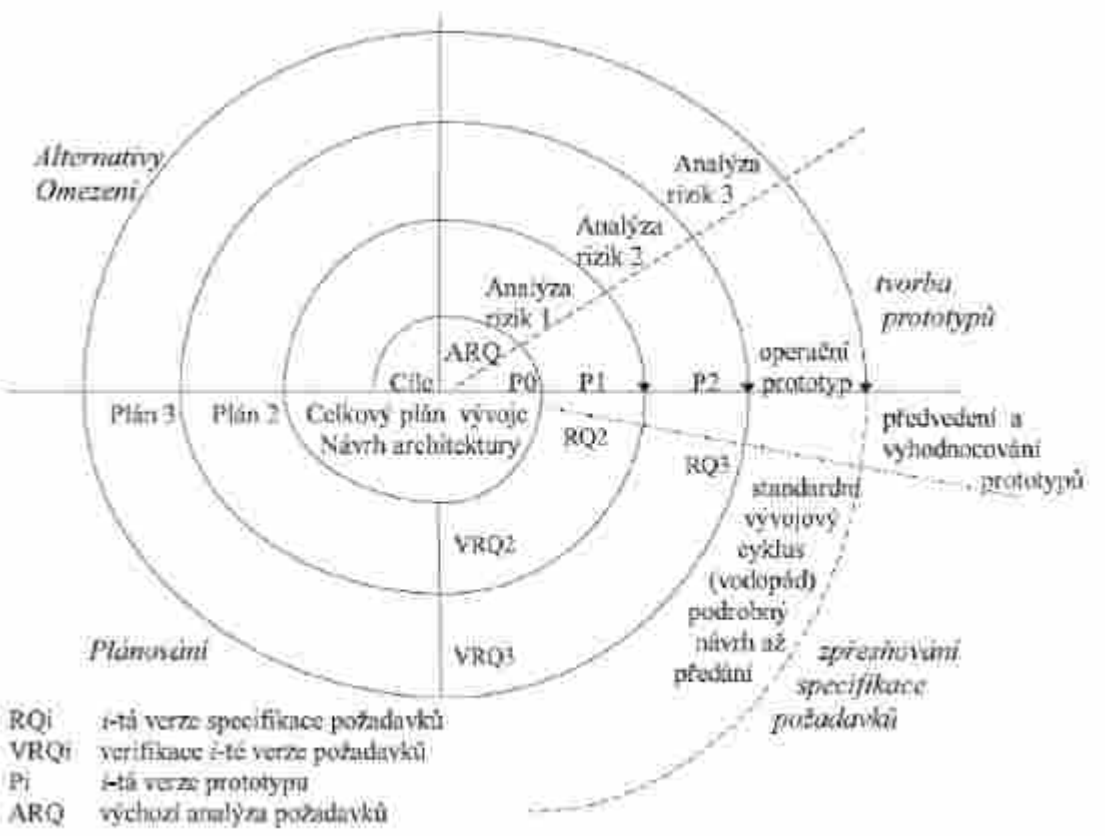
**V literatuře je řazen mezi metodiky vývoje software, ale jedná se o model životního cyklu programového díla.**

Model životního cyklu softwarového produktu, který vychází z vodopádového modelu, ale přináší dvě zcela nové zásadní vlastnosti - **iterativní přístup** a **podrobnou analýzu rizik**. Celý model patří do skupiny *přístupů řízených riziky (risk-driven approach)*.

**Iterativní přístup** byl důsledkem toho, že u spousty projektů se nepodařilo stanovit přesnou a úplnou specifikaci požadavků, což činilo vodopádový model takřka nepoužitelným. Ukázalo se, že bude mnohem lepší stanovit na počátku jen rámec architektury a funkčnosti celého systému a ten postupně rozpracovávat do detailů. Cyklické opakování jednotlivých kroků vývoje, tzv. iterace, znamenalo ve své době přelom v chápání životního cyklu.

**Analýza rizik** je prováděna v každém cyklu a určuje další směr vývoje projektu. Riziko je tedy klíčovým pojmem, pod kterým se rozumí jakákoliv událost nebo situace, která můžetedy klíčovým pojmem, pod kterým se rozumí jakákoliv událost nebo situace, která může ohrozit projekt. Při analýze je nutno každému riziku přiřadit jeho nebezpečnost a pravděpodobnost výskytu. Častá a podrobná analýza rizik má za úkol dostatečně dopředu odhalit nevhodná řešení nebo skryté problémy, které by mohly ohrozit průběh projektu.

Model kombinuje prototypování se systematickým sekvenčním přístupem a opakováním na výším stupni zvládnutí problematiky. Je založen na prioritě tvorby verze s vyšší rizikovostí. Části s větší mírou rizika jsou realizovány dříve. Nevýhodou modelu je, že nelze stanovit přesné termíny (při práci na zakázku), závisí na správnosti rizikové analýzy, náročné na zkušenost pracovníků (je nutné více kontrolních bodů, po každé analýze rizik). Je vhodný pro velké systémy.




Jiný obrázek z Wikipedie:


## Rational Unified Process

Detailně propracovaná rozsáhlá metodika vývoje software Rational Unified Process (**RUP**) je komerčním produktem firmy Rational (později koupena firmou IBM). Celá metodika je objektově orientovaná a patří do skupiny přístupů *řízených případy užití (use-case-driven approach)*. Vývoj podle RUP probíhá v **iterativních cyklech**. První cyklus se nazývá **úvodní vývojový cyklus** a jeho výsledkem je **funkční softwarový produkt** implementující

**nejpodstatnější část funkcionality**. Zde ovšem vývoj nekončí, ale pokračuje v různém množství rozvíjejících cyklů.

Každý cyklus se skládá ze 4 fází (v závorce je uvedeno typické rozdělení doby pro jednotlivé fáze úvodního vývojového cyklu): - Zahájení (10%) - Projektování (30%) - Realizace (50%) - Předání (10%)

RUP je dodávána ve formě internetových stránek, které slouží jako online instruktor, který vývojáře vede celým průběhem vývojového procesu, poskytuje instrukce, rady, metodické pokyny a příklady ke konkrétní fázi vývoje. RUP je těsně provázán s jazykem **UML**. Všechny dokumenty, modely a případy užití jsou modelovány právě v UML.

Největší **výhodou RUP** je jeho **robustnost, obecnost a přizpůsobivost pro celou řadu nejrůznějších projektů**. Jedna z úvodních fází při použití RUP je modifikace samotné metodiky na míru konkrétnímu projektu, který začínáme realizovat. Není problém upravit metodiku ani pro malý tým a jednodušší projekt. Součástí metodiky jsou nejrůznější průvodci, šablony a podpůrné nástroje.

## Agilní metodiky

S rostoucí konkurencí v oblasti vývoje software se postupně místo kvality klade větší důraz na vysokou rychlost a nízkou cenu vývoje. Po roce 2000 začaly dostávat na světlo metodiky, které umožnily vyvíjet software **rychle a přitom byly schopny reagovat na průběžnou změnu zadání**. Začalo se jim říkat agilní. Název vzešel z anglického agile, což znamená hbitý, čilý, bystrý nebo svižný.

**Klasické metodiky** počítají s funkcionalitou jako s fixní veličinou, na počátku vývoje se stanoví pevná specifikace požadavků, která se musí dodržet. Čas a zdroje jsou proměnné, a mění se podle toho, jak vývoj postupuje kupředu. Proto se u projektů vyvíjených podle tradičních metodik nezřídka setkáváme s překročením termínu dodání a zvýšenými náklady oproti původnímu plánu.

U **agilního přístupu** je tomu právě naopak. Jako fixní jsou při startu projektu stanoveny zdroje, tzn. finance, lidské zdroje, vývojová prostředí a jiné, a funkcionalita je proměnná. V konečném důsledku zákazník může dostat produkt, který zatím **implementuje jen část funkcionality**, ovšem díky agilnímu pojetí vývoje se snadno upravuje a vylepšuje i za provozu. Většinou je pro zákazníka výhodnější, když dostane nedokonalou aplikaci v termínu a za dané peníze, než kdyby musel zaplatit další pokračování vývoje a ještě několik týdnů počkat na dokončení aplikace.

### Principy agilního programování

**Následující principy jsou společné pro všechny metodiky:**

**Iterativní a inkrementální vývoj s krátkými iteracemi** Vývoj probíhá v krátkých fázích,

takže celková funkcionalita je dodávána po částech. Zákazník tak má možnost průběžně sledovat vývoj, může se k němu vyjádřit a oponovat změny. Na konci má jistotu, že nedostane něco, co neočekával. **Komunikace mezi zákazníkem a vývojovým týmem** V ideálním případě je zákazník přímo součástí vývojového týmu, má možnost okamžitě vidět průběžné výsledky a reagovat na ně. Účastní se sestavování návrhu, spolurozhoduje o testech a poskytuje zpětnou vazbu pro vývojáře.

**Průběžné automatizované testování** Díky krátkým iteracím se aplikace mění velice rychle, proto je nutné pro zajištění co nejvyšší kvality ověřovat její funkčnost průběžně. Testy by měly být automatizované, předem sestavené a měly by být napsány ještě před samotnou implementací testované části. Při každé změně musí být aplikována kompletní sada testů, aby nebyla porušena integrace jednotlivých částí.

## Přehled agilních metodik

### Extrémní programování (Extreme Programming, XP: Kent Beck, Ward Cunningham, Ron Jeffries)

Nejznámější zástupce agilních metodik. Mnoho lidí se mylně domnívá, že agilní metodika rovná se extrémní programování (XP). XP je sice nejrozšířenější a považováno za jakéhosi zástupce, ale pořád je to jen jedna z mnoha agilních metodik. Hodí se pro menší projekty a malé týmy, vyvíjející software podle zadání, které je nejasné nebo se rychle mění. Vychází z přesvědčení, že **jediným exaktním, jednoznačným, změřitelným, ověřitelným a nezpochybnitelným zdrojem informací je zdrojový kód**. Používá běžné principy a postupy, které dotahuje do extrémů.

### Vývoj řízený vlastnostmi (Feature-Driven Development, FDD: Jeff de Luca, Peter Coad)

Zaměřuje se na **vývoj po malých kouscích** - vlastnostech, rysech, což jsou elementární funkcionality přinášející nějakou hodnotu uživateli. Vývoj probíhá **v pěti fázích**, první tři jsou sekvenční, poslední dvě pak iterativní. Iterace trvají zpravidla 2 týdny. Začíná se vytvořením modelu, ten se převede do seznamu vlastností, které se postupně implementují. Velice dobře se měří pokrok ve vývoji projektu, FDD umožňuje detailně plánovat a kontrolovat vývojový proces. Hlavní výhodou je zaměření na dodávání fungujících přírůstků každé dva týdny.

### SCRUM Development Process: Ken Schwaber, Mike Beedle

Principem je opět iterativní vývoj definující 3-8 fází, tzv. **sprintů**, každý z nich trvá přibližně měsíc. Metodika nedefinuje žádné konkrétní procesy, pouze zavádí pravidelné každodenní schůzky vývojového týmu, tzv. **scrum meetings**. Zde si jednotliví členové sdělují, které položky byly od minulé schůzky dokončeny, které nové úkoly nyní přijdou na řadu a jaké překážky stojí vývoji v cestě a musí se vyřešit. Každý sprint je zakončen předvedením funkční demo-aplikace zákazníkovi, který poskytne zpětnou vazbu.

### Adaptivní vývoj software (Adaptive Software Development, ASD): Jim Highsmith

ASD definuje místo tradičních sekvenčních fází plánování - návrh - realizace iterativní fáze spekulace - spolupráce - učení. Hlavním přínosem této metodiky je, že **odchylky od plánu nechápe jako chyby, ale jako příležitosti k učení**.

### Lean Development: Mary Poppendieck, Tom Poppendieck

Není metodikou v pravém slova smyslu, ale spíše **souhrnem deseti pravidel**, kterých dodržování slibuje efektivnější a rychlejší vývojový proces. Příkladem pravidel může být: odstranit vše zbytečné, minimalizovat zásoby, zavést zpětnou vazbu, odstranit lokální optimalizaci, apod. Dále na základě pravidel definuje **7 principů spolu s nástroji, jak tyto principy realizovat v praxi**.

### Vývoj řízený testy (Test-Driven Development, TDD): Kent Beck

Nezabývá se tvorbou specifikací, plánů a dokumentace, to si každý tým musí zvolit sám podle toho, jak mu to vyhovuje. TDD doporučuje přistoupit k testům jako k hlavní fázi celého vývojového procesu. Základním pravidlem je **psát testy dříve než samotný kód** a **implementovat** jen přesně **takové množství kódu, které projde testem**. Nic méně, ale také nic více.

### Crystal metodiky (Crystal family of methodologies): Alistair Cockburn

Již z původního anglického názvu je patrné, že se nejedná jen o jednu metodiku, nýbrž o celou **rodinu metodik**. Autor vychází z toho, že **sebelepší metodika nemůže vyhovovat každému projektu** a je lepší **přizpůsobit metodiku na míru** danému **projektu**. Dalo by se říct, že první fází vývoje je vlastně vytvoření metodiky. Kritérii pro výběr správné metodiky je například velikost projektu, velikost vývojového týmu nebo kritičnost projektu.