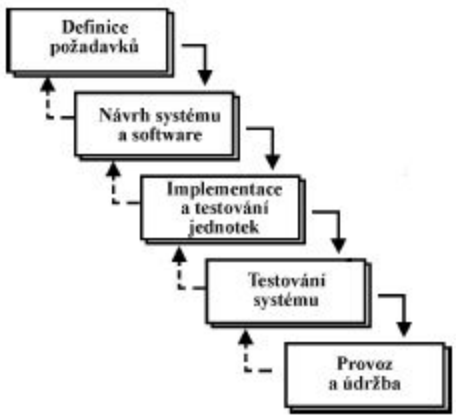
ÚVOD, VÝVOJOVÉ METODIKY

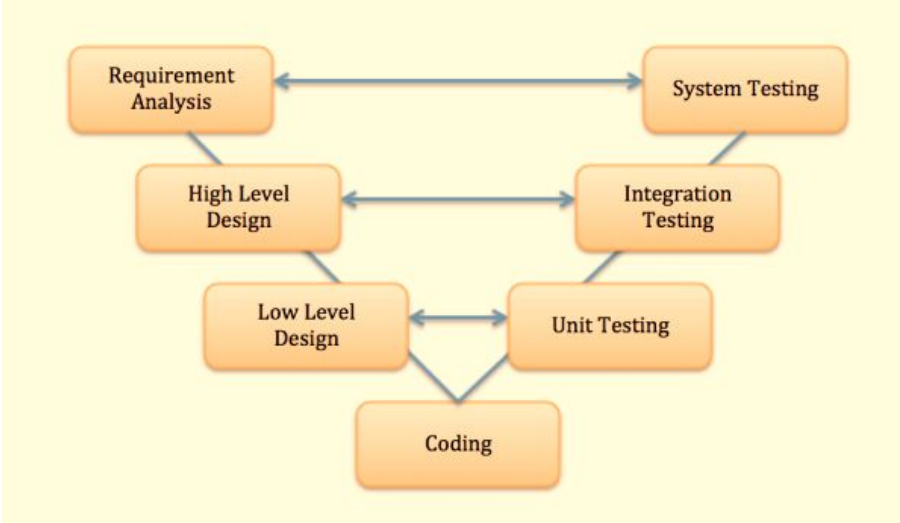
- Definujte co je to software a z čeho se skládá:
    - Software je tvořen celkovým souhrnem počítačových programů, procedur, pravidel a průvodní dokumentace a dat, která náleží k provozu počítačového systému.
    - **Dokumentace:** vývojová, administrátorská, uživatelská.
    - **Softwarový produkt** je výrobek určený k předání uživateli. Charakteristiky:
        - Je vyvíjen a řešen inženýrskými pracemi, není vyráběn v klasickém slova smyslu.
        - Fyzicky se neopotřebuje.
        - Prvotní investice se v budoucnu vrátí (znovupoužití kódu).
        - Problémy s údržbou: helpdesk, odstranění chyb ⇒ vysoká cena programů.
        - Extrémní individuální odchylky v produktivitě programátorů (1:28).
        - Invariance programovacího jazyka: Složitost aplikace má větší vliv na produktivitu, než volba programovacího jazyka.
        - Obvykle na míru; málo produktů je sestaveno z předem existujících komponent.

- Popis + 2 výhody a 3 nevýhody vodopádu:
    - **Výhody:**
        - Čas strávený na začátku zajišťováním toho, že požadavky i návrh jsou naprosto správné a kompletní, ušetří mnoho úsilí a času později.
        - Klade stejný důraz na dokumentaci jako na zdrojový kód.
        - Poskytuje strukturovaný přístup, model postupuje lineárně, diskrétními, jednoduše pochopitelnými a vysvětlitelnými fázemi.



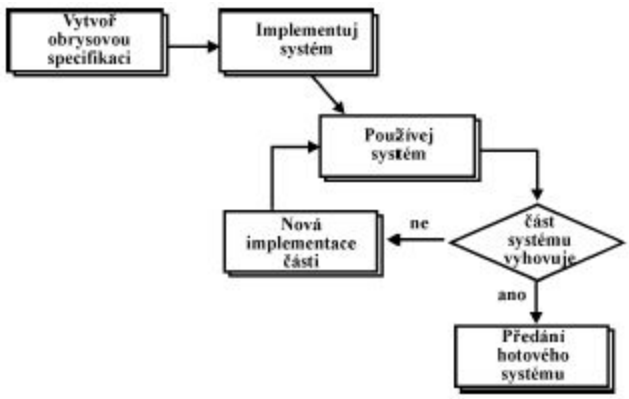
        - Integrace systému zároveň s jeho testy.
        - Dělení na části - ideální pro velké týmy a projekty.
        - Vždy je jasné, jaký další krok následuje - usnadnění pro management.
    - **Nevýhody:**
        - Reálné projekty nedodržují jednotlivé kroky v předepsaném pořadí.
        - Uživatel nedokáže v počátečních etapách formulovat požadavky na systém zřetelně a přesně.
        - Zákazník musí být trpělivý.
        - Pozdní odhalení nedostatků může vážně ohrozit celý projekt.
        - U velkých projektů je velká šance, že se přehlédne něco důležitého během analýzy a návrhu systému.

- V-model návrhu software:



    - V-model můžeme považovat za rozšíření modelu vodopád.
    - Vývoj však neprobíhá v lineárním směru - křivka je ohnuta do tvaru V, po fázi implementace nastávají fáze testování, verifikace, validace a údržby.
    - Model demonstruje vztah mezi každou fází návrhu a s ní asociovanou fází testování.
    - Horizontální křivka reprezentuje dokončenost projektu, vertikální zase úroveň abstrakce.

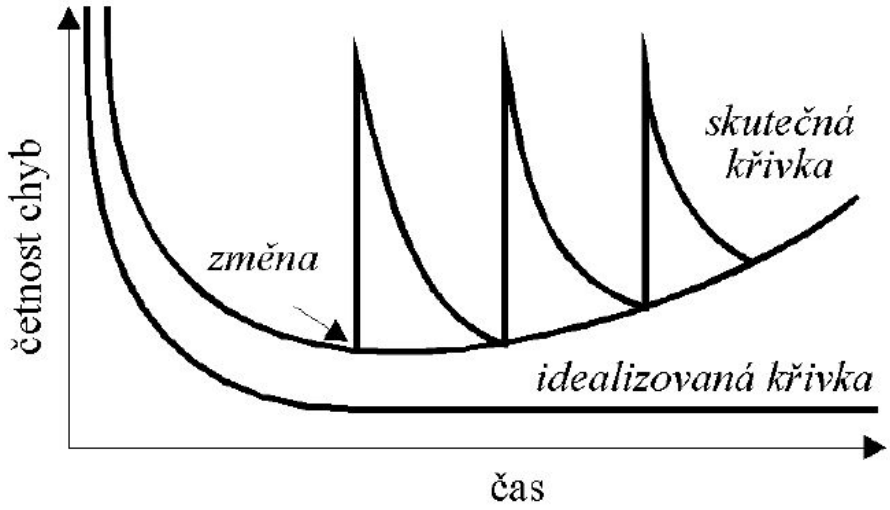
- 3 výhody a 2 nevýhody inkrementálního životního cyklu:



    - **Výhody:**
        - Zákazník může průběžně dávat feedback ke stavu softwaru - vývojáři tak získají cenné informace o používání či chybách.
        - Vzhledem k tomu že inkrementální model implementuje software po částech, jsou pozdější úpravy modulů poměrně jednoduché.
        - Software je použitelný už ve své rannější fázi, ne až po dokončení všech vývojových fází jako je tomu např. u vodopádu.
    - **Nevýhody:**
        - Obrysová specifikace vs. skutečnost
        - Dokumentace programu vs. specifikace
        - Údržba zvyšuje entropii

- Rozdíl mezi iterativním a inkrementalním vývojem:
    - **Inkrementální** vývoj využívá daný počet kroků a vývoj tak jde od začátku do konce lineární cestou. Kroky inkrementálního vývoje mohou být: návrh, implementace, testování, údržba. Ty se dají dále rozdělit. Klasickým příkladem je model vodopád.
    - **Iterativní** přístup nemá pevně daný sled kroků, vývoj probíhá v cyklech. Méně se zabývá sledováním postupu u jednotlivých funkcionalit. Je zaměřen na vytvoření funkčního prototypu a nabalování další funkcionality v jednotlivých cyklech. Příkladem je agilní vývoj.

- Křivka znázorňující množství chyb při vývoji SW + popsat:



    - Zákazník požaduje během provozu nové a nové úpravy, což dělá systém složitějším a zanáší do systému další chyby. Jednou přijde den, kdy bude lepší to celé naprogramovat znovu.
    - Čím později v životním cyklu detekujeme chybu, tím nákladnější bude její oprava.
    - Mnoho chyb zůstane skryto a je odhaleno až po ukončení fáze, v níž byly udělány.
    - V požadavcích je mnoho chyb.
    - Chyby v požadavcích jsou především chybná fakta, opomenutí, rozpory a nejednoznačnosti.
    - Chyby v požadavcích lze detekovat.

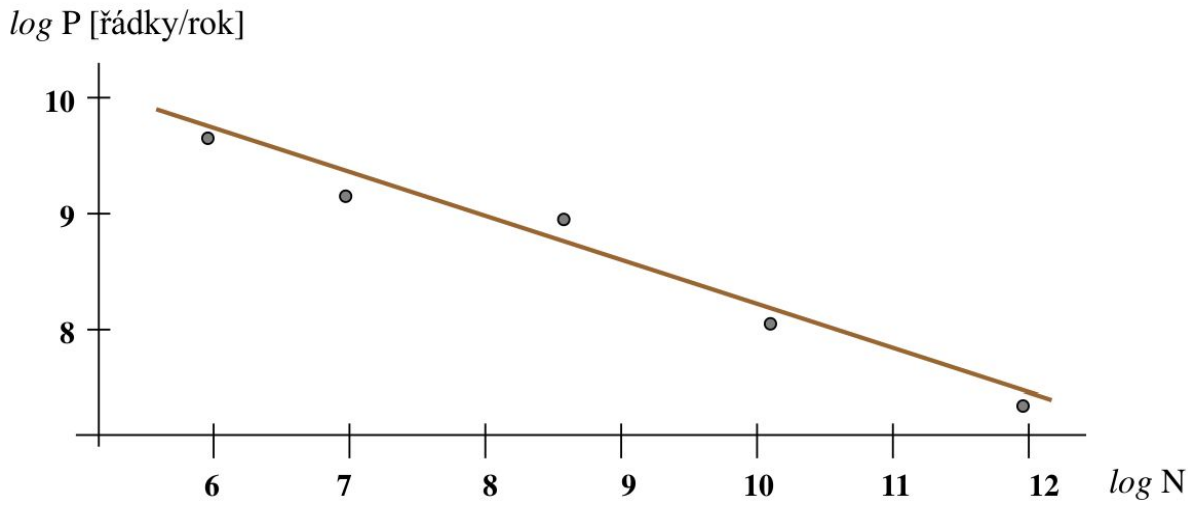
- Vyjmenovat všech 5 Lehmannových zákonů:
    - **Zákon trvalé proměny:** Systém používaný v reálném prostředí se neustále mění, dokud není levnější systém restrukturalizovat, nebo nahradit zcela novou verzí.
    - **Zákon rostoucí složitosti:** Při evolučních změnách je program stále méně strukturovaný a vzrůstá jeho vnitřní složitost. Odstranění narůstající složitosti vyžaduje dodatečné úsilí.
    - **Zákon vývoje programu:** Rychlost změn globálních atributů systému se může jevit v omezeném časovém intervalu jako náhodná. V dlouhodobém pohledu se však jedná o seberegulující proces, který lze statisticky sledovat a předvídat.
    - **Zákon invariantní spotřeby práce:** Celkový pokrok při vývoji projektů je statisticky invariantní. Jinak řečeno, rychlost vývoje programu je přibližně konstantní a nekoreluje s vynaloženými prostředky.
    - **Zákon omezené velikosti přírůstku:** Systém určuje přípustnou velikost přírůstku v nových verzích. Pokud je limita překročena, objeví se závažné problémy týkající se kvality a použitelnosti systému.

- Brooksův zákon - znění, popis o čem je, proč platí:
    - **Přidání řešitelské kapacity u opožděného projektu může zvětšit jeho zpoždění.**
    - Platí protože náklady na začlenění nového pracovníka do týmu jsou zpravidla větší než jeho přínos
    - Stávající programátor se musí věnovat nově příchozímu členovi a tím se okrádá o čas, zároveň nějakou dobu trvá, než nový programátor začne být produktivní.

- Popsat CMM - Capability Maturity Model (z wikipedie, otázky bývají i na jednotlivé úrovně):
    - Vývojový model navržený k vylepšení ostatních softwarových modelů.
    - Úrovně:
        1. **Initial** (chaotic, ad hoc, individual heroics) - the starting point for use of a new or undocumented repeat process.
        2. **Repeatable** - the process is at least documented sufficiently such that repeating the same steps may be attempted.
        3. **Defined** - the process is defined/confirmed as a standard business process.
        4. **Managed** - the process is quantitatively managed in accordance with agreed-upon metrics.
        5. **Optimizing** - process management includes deliberate process optimization/improvement.

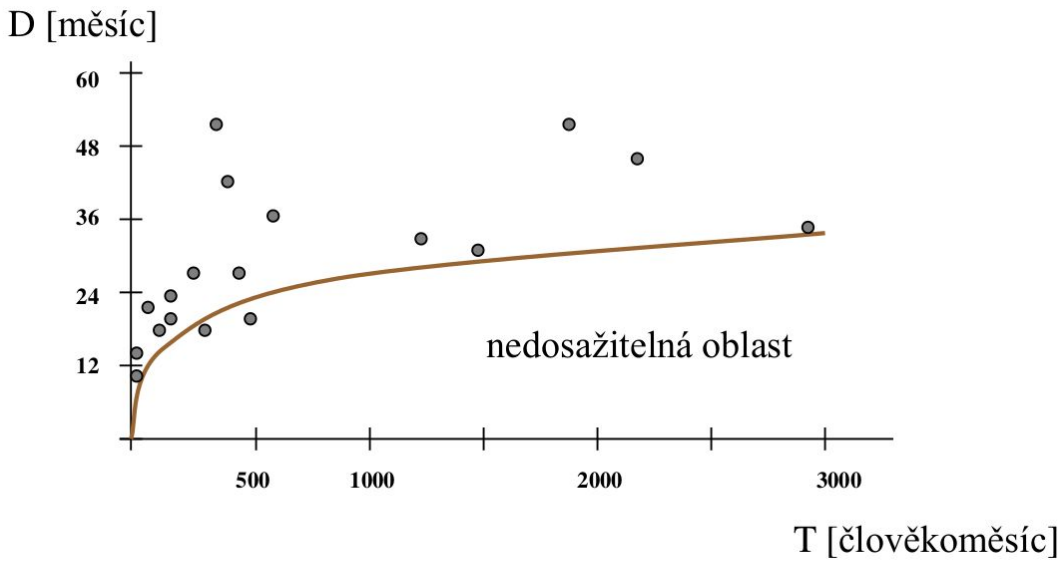
SOFTWAROVÁ FYZIKA, ODHADY

- Nakreslit graf závislosti produktivity na délce programu + popsat to:



    - S rostoucí délkou programu klesá produktivita programátorů.

- Putnamova rovnice, k čemu je, jak se počítá/používá?
    - $\mathbf{N = c \cdot T^{1/3} \cdot D^{4/3}}$
    - N - délka programu (počet řádek, SLOC - Software Lines Of Code)
    - c - škálovací faktor velikosti projektu
    - T - spotřeba práce (člověkoměsíce, MM)
    - D - doba realizace programu
    - Používá se k empirickému odhadu úsilí vývoje softwaru.
    - Důsledky:
        - Programy psané ve spěchu jsou delší.
        - Při zkrácení termínu na 83% je pracnost je dvojnásobná.



- COCOMO, k čemu je, jak se počítá:
  - Model používaný k odhadování ceny SW.
  - Zdroje empirických dat:
    - větší počet předchozích komplexních projektů
      - aplikace odlišného druhu s odlišnými cíly
      - odlišná vývojová prostředí
    - rozhovory s více manažery
  - **$E = a \cdot F \cdot (KSLOC)^b$**
  - **$T = c \cdot E^d$**
    - E - Effort Applied - vynaložené úsilí
    - F - korekční faktor (viz. další otázka)
    - T - Time - čas vývoje
    - KSLOC - tisíce řádků kódu
    - a,b,c,d: parametry volené podle úrovně modelu a vývojového módu

- Co je to korekční faktor F_c u COCOMO, jak se spočítá, kde se používá?
  - Korekční faktor upravující COCOMO na základě množiny subjektivních faktorů
  - Atributy, které mají vliv na korekční faktor F_c:
    - atributy SW produktu (spolehlivost, složitost, velikost DB)
    - HW atributy (požadavky prostředí, paměti, volatility HW)
    - atributy vývojového týmu (zkušenost týmu s analýzou, SW inženýrstvím, znalost programovacích jazyků, testováním)
    - atributy projektu (použití SW nástrojů, IDEček, plán vývoje)

- COCOMO II, popsat 3 modely:
  - APM (Aplication Composition Model)
    - pro projekty s použitím moderních nástrojů a GUI
  - EDM (Early Design Model)
    - pro hrubé odhady v úvodních etapách, kdy se architektura vyvíjí
  - PAM (Post Architecture Model)
    - pro odhady poté, co byla specifikována architektura

- Funkční body:
    - Funkční bod = normalizovaná metrika softwarového projektu
        - Měří aplikační oblast, nezkoumá technickou oblast
        - Měří aplikační funkce a data, neměří kód
            - Tedy co by měla aplikace dělat a zda to dělá, ne jak to dělá
    - Spravuje International Function Point Users Group - www.ifpug.org
    - Typy funkčních bodů:
        - Funkční body vztažené k transakčním funkcím:
            - Externí vstupy (EI - External Inputs)
            - Externí výstupy (EO - External Outputs)
            - Externí dotazy (EQ - External Enquiry)
        - Funkční body vztažené k datovým funkcím:
            - Vnitřní logické soubory (ILF - Internal Logical Files)
            - Soubory vnějšího rozhraní (EIF - External Interface Files)

- Popište jakým způsobem dostaneme z neupravených funkčních bodů počet upravených funkčních bodů:
    - Před výpočtem musíme EI, EO, EQ, ILF, EIF roztřídit do skupin podle vah:

Váhy	nízká	průměrná	vysoká	celkem
EI	___ x 3 +	___ x 4 +	___ x 6 =	_____
EO	___ x 4 +	___ x 5 +	___ x 7 =	_____
EQ	___ x 3 +	___ x 4 +	___ x 6 =	_____
ILF	___ x 7 +	___ x 10 +	___ x 15 =	_____
EIF	___ x 5 +	___ x 7 +	___ x 10 =	_____

Neupravené funkční body celkem _____

    - Dále existuje 14 charakteristik hodnocených podle stupně vlivu na aplikaci
    - Každý faktor je hodnocený ve stupnici 0 – 5 takto: 0 = bez vlivu, 5 = podstatný
    - Počet funkčních bodů = [0.65 + (0.01 x součet hodnocení charakteristik systému)] × [počet neupravených funkčních bodů]

- Rozdíl mezi funkčními body projektu a produktu, vysvětlit:
    - **Funkční body produktu** - při vývoji aplikace “na zelené louce” jsou tyto body takové body, které zůstanou v aplikaci na konci vývoje
    - **Funkční body projektu** - prošly týmu rukama, zaplatili jsme za ně, nemusí na konci vývoje zůstat (např. body vynaložené na vyzkoušení nějakého postupu a následné předělání zpět), musíme s nimi však nutně počítat

- Jaký je rozdíl při počítání funkčních bodů u finálního SW produktu a při kalkulaci cen za SW projekt:

Type of Project	Project Function Points	Application Function Points Installed Function Pts. (IFP)
Development Project	Project FP = New (Added) FP + Conversion FP	Application FP = New (Added) FP
Enhancement Project	Project FP = Added FP + Changed FP + Deleted FP + Conversion FP	Application FP = Original FP - Deleted FP + Added FP + Δ Changed FP

- Položky v kalkulaci projektu:
  - Stanovení popisu projektu, požadavků, doby trvání
  - Stanovení činností (programování, testy, tvorba webu, zajičtění bezpečnosti, školení, dokumentace, technická podpora)
  - Stanovení rolí v projektu (PM, analytik, architekt, developer, tester, grafik,...)
    - Pro každou roli popis, náklady/MD (včetně elektřiny, kouření, notebooku,...) a výnosy/MD (hodnota zaměstnance), případně finanční kategorie
  - Tabulka činnost × role = pracnost (v MD)
  - Další náklady: licence na SW, nákup HW, subdodávky, podpora, rizika
  - Stanovení rizik (ID, popis, pravděpodobnost, dopady, náklady)
  - Sumarizace: náklady, marže, výnosy

- Jaké zakladní body by měla obsahovat servisní smlouva (doplněno z nějaké pdf smlouvy z netu, ve slajdech jsem to nikde nenašel):
  - Účel smlouvy
  - Dobu trvání smlouvy
  - Specifikace servisní činnosti, vymezení pojmů (např. aktualizace dat, školení uživatelů, customizace, provoz helpdesku)
  - Formu servisní činnosti
  - Způsob oznamování závad
  - Reakční doby a způsob odstranění závady
  - Součinnost objednatele

- Definovat SLA (Service Level Agreement), co obsahuje a příklad (z wikipedie):
  - SLA označuje smlouvu sjednanou mezi poskytovatelem služby a jejím uživatelem.
  - Poměrně obšírně se tématem poskytování IT služeb zabývá metodika ITIL.
  - Každé prodání (poskytnutí) produktu (software, služby, výrobku,...) je provedeno zároveň s definicí toho produktu.
  - Tato definice by měla uživateli i výrobci vymezit jasná pravidla, jak se o produkt dále starat, jak jej používat a jeho cenu. Například se jedná o záruku na produkt, běžné užití produktu, k čemu je produkt určen, kolik produkt stojí,...

AGILNÍ METODIKY

- Manifest agilního vyvoje + hlavní zasady:
    - Umožnit změnu je mnohem efektivnější, než se jí snažit zabránit. Je třeba být připraven reagovat na nepředvídatelné události, protože ty nepochybně nastanou.
    - Individuality a interakce mají přednost před procesy a nástroji.
    - Fungující software má přednost před obsáhlou dokumentací.
    - Spolupráce se zákazníkem má přednost před sjednáváním smluv.
    - Reakce na změnu má přednost před plněním plánu.

- Kdy se vyplatí vyvíjet SW agilní metodikou:
    - Při menší velikosti vývojového týmu
    - Pokud je zákazník ochoten poskytnout určitou volnost ve výsledné funkcionalitě
    - Pokud projekt obsahuje agresivní deadliny
    - Pokud má projekt vysokou míru složitosti
    - Pokud vývojový tým ještě nepracoval na podobném typem projektu
        - na eshop vyvíjený stále dokola se tedy hodí vodopád
        - na vývoj velmi specifické komponenty závislé na určitých rozhraních/datech je lepší použít některou z agilních metodik

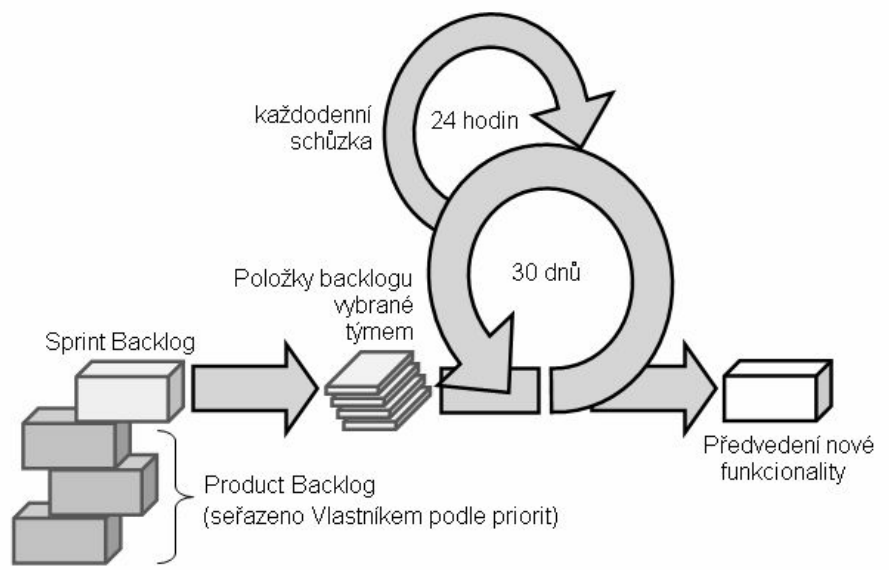
- Co mají společné generické a agilní vývojove metodiky:
    - Společnými rysy všech agilních metodik jsou:
        - Iterativní a inkrementální vývoj s krátkými iteracemi
        - Komunikace mezi zákazníkem a vývojovým týmem
        - Průběžné automatizované testování

- Agilní vývoj + jaké parametry zkoumá metrika prepojenia - viz. otázka Coupling níže

- Které atributy jsou proměnné při agilním vývoji + rozvést to:

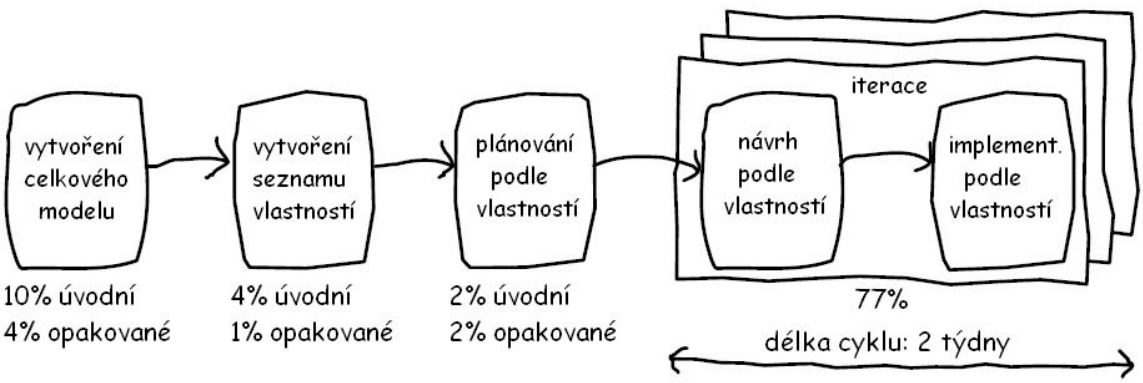
- Popsat Scrum:
    - Principem je iterativní vývoj definující 3-8 fází (tzv. sprintů), z nichž každá trvá přibližně měsíc.
    - Nedefinuje žádné konkrétní procesy, pouze zavádí pravidelné každodenní schůzky vývojového týmu (scrum meetings), kde si jednotliví členové sdělují, které položky byly od minulé schůzky dokončeny, které nové úkoly nyní přijdou na řadu a jaké překážky stojí vývoji v cestě a musí se řešit.

- Každý sprint je zakončen předvedením funkční demo-aplikace zákazníkovi, který poskytne zpětnou vazbu.



- Co je to "product backlog"? Kde/jak se používá?
  - Používá se u Scrumu.
  - Je to jednoduše seznam věcí které se na projektu musí udělat.
  - Každá položka obsahuje odhad potřebného času na dokončení a prioritu.
  - Nahrazuje klasickou specifikaci požadavků.
  - Položky backlogu mohout mít jak technickou (přidat tuto funkcionalitu, fixnout tento bug) tak uživatelskou povahu (zjednodušit uživatelské rozhraní).
  - Product owner ho používá při plánování sprintu k zadávání úkolů.

- Feature driven development:
  - Vývoj po malých kouscích (vlastnostech, rysech), což jsou elementární části 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.
  - Měří pokrok ve vývoji projektu, FDD detailně plánuje a kontroluje vývojový proces.
  - Zaměření na dodávání fungujících přírůstku každé dva týdny.



UNIFIED PROCESS

- Unified process (popsat některou z fází):
    - Proces definuje KDO udělá CO, a KDY a JAK to udělá
    - Unifikovaný proces je :
        - Iterativní a inkrementální
        - Řízen use casy
        - Zaměřen na architekturu systému
    - Fáze viz. další otázka

- Jaké jsou fáze dle Unified Procesu, popis.:
    - Zahájení - Definice rozsahu projektu, vytvoření byznisového cílu
        - Milník: vize
    - Příprava - Naplánování projektu, specifikace funkcionality, navrhnutí architektury
        - Milník: základ architektury
    - Konstrukce - Vývoj produktu
        - Milník: počáteční funkcionalita
    - Předávání - Předání produktu uživatelům
        - Milník: release produktu

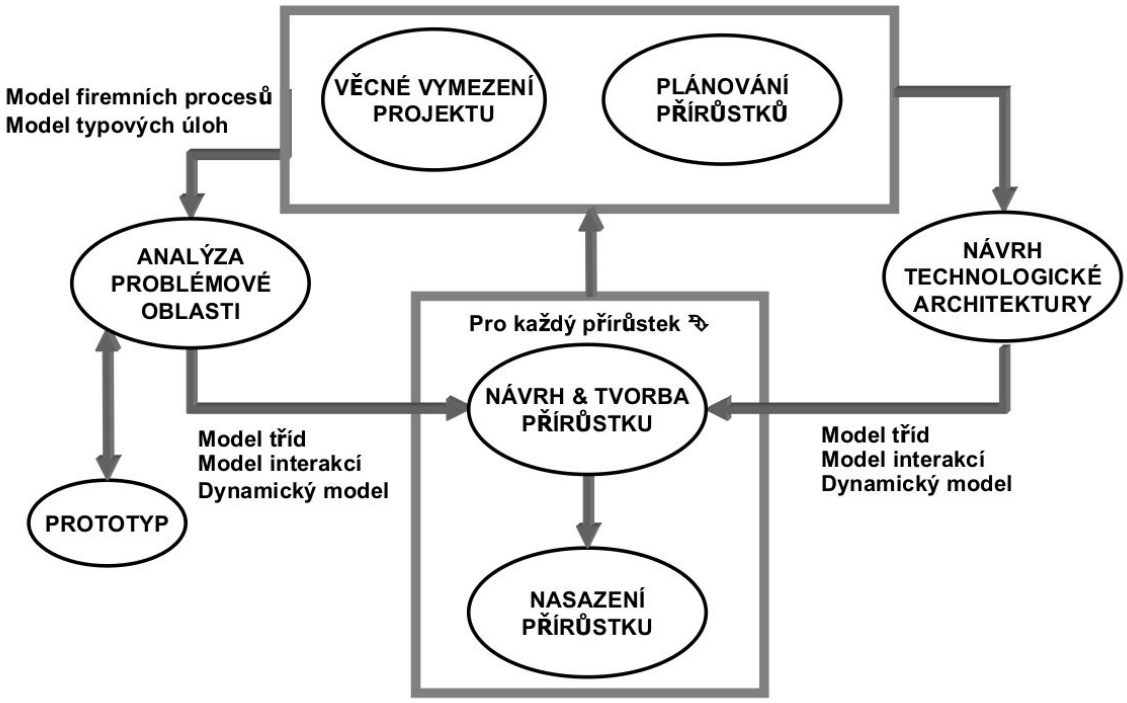
- Jaké diagramy používá Elaboration fáze Unified Procesu? Popsat jednotlivý diagram:
    - Diagram tříd
        - Popis objektového modelu systému. Znázorňuje objekty, jejich atributy, metody a vztahy mezi ostatními objekty.
    - Sekvenční diagram
        - Sekvenčně popisuje nějaký proces v systému. Znázorňuje komunikaci mezi objekty pomocí zpráv (funkční volání). Objekty “žijí” v průběhu tzv. lifelines.
    - Kolaborační diagram
        - Podobný sekvenčnímu diagramu. Sekvence je ale očíslována. Toto číslování specifikuje pořadí volání metod.
    - Stavový diagram
        - Znázorňuje takovou událost probíhající v systému, která má konečný počet stavů. Má formu stavového automatu. Jeho součástí jsou stavy, přechody a události.
    - Diagram aktivit
        - Zachycuje proměnné chování (flow) nějaké aktivity v systému. Popisuje sekvenci přechodu od jedné aktivity k druhé. Popisuje i paralelní, rozvětvené a souběžné flows systému.

- Jaké diagramy se používají v Use Case Model v UP?
    - Use case diagram
        - Znázorňuje případy použití systému. Různé role (actors) mohou mít různé případy použtí.
    - Sekvenční diagram (viz. výše)
    - Stavový diagram (viz. výše)
    - Diagram aktivit (viz. výše)

SELECT PERSPECTIVE

- Select Perspective:
  - Kombinace **procesního modelování** a **objektově orientované metodiky**
    - Businness Process Reengineering (BPR)
    - Object Modelling Technique (OMT)
  - Využívá UML diagramy a procesní mapy
  - Inkrementální postup
  - Architektura systému založená na komponentách

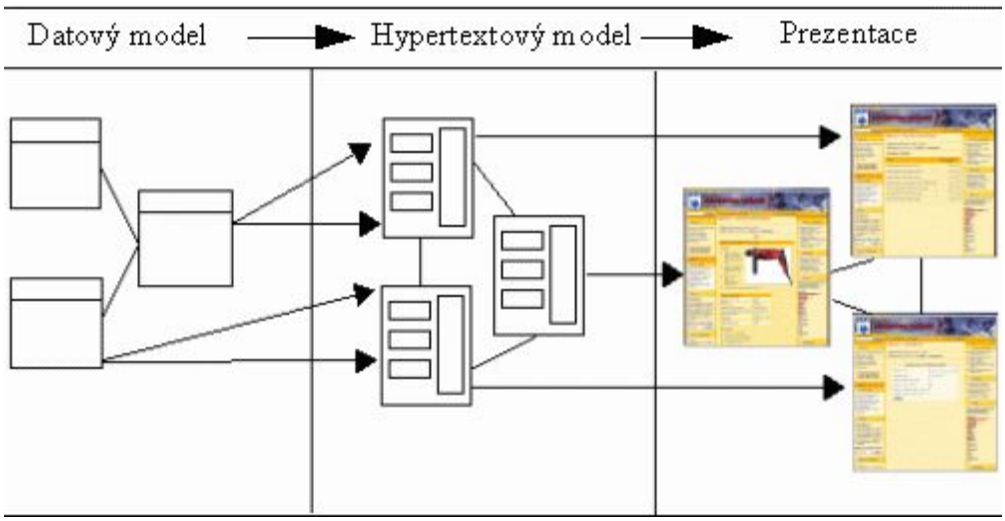
- Popsat modely SP a pořadí jejich vytváření:
  1. Modelování firemních procesů
  2. Modelování případů užití
  3. Modelování tříd objektů
  4. Modelování interakcí objektů
  5. Modelování dynamiky objektů
  6. Modelování komponent



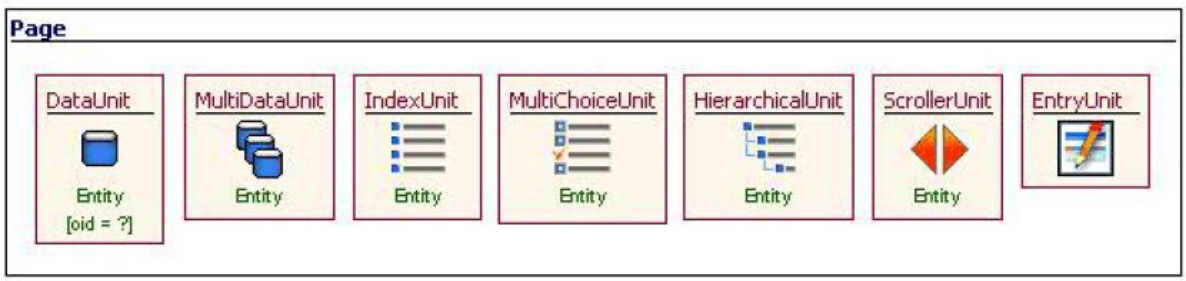
WebML

- Popsat WebML:
  - Web Modeling Language.
  - Modelovací nástroj i metodika.
  - Při vývoji (modelování) webů je důležitým prvkem navigace.
  - Umožňuje vytvořit komplexní model webové aplikace.
  - Není „zbytečně“ robustní jako např. UML.

- Podpora v CASE nástrojích:
  - WebRatio Site Development Studio (podporuje generování kódu z navržených modelů)
  - MS Visio (obsahuje šablony diagramů).
- Skládá se z těchto modelů:
  - Datový model
  - Hypertextový model
  - Prezentační model



- Popsat hypertextový model WebML:
  - Je tvořen dvěma submodely, které jsou spolu těsně spjaty.
  - Prvním je **kompozice webové aplikace**, kde je definováno složení prezentace z jednotlivých stránek a složení stránek z jednotlivých předdefinovaných elementů (units). Ty představují atomické součásti obsahu stránky a jsou vázány na entity datového modelu, ze kterých čerpají svůj obsah nebo chování.
  - Druhou částí hypertextového modelu je **navigace**, která zobrazuje, jak jsou mezi sebou jednotlivé stránky - respektive jejich elementy provázány. Definuje strukturu a funkčnost webové aplikace na konceptuální úrovni, která je nezávislá na implementačních detailech.

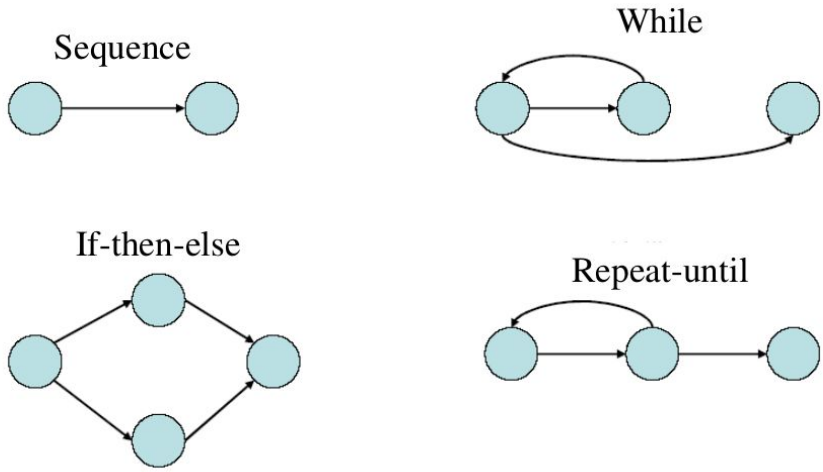


- Jaké dva modely používá hypertextový model WebML (a popsat je):
  - Viz. předchozí otázka

MĚŘENÍ, METRIKY

- Charakteristiky metrik založené na velikosti kódu (size oriented):
  - Velikost softwarového produktu
  - Lines Of Code (LOC)
  - 1000 Lines Of Code (KLOC)
  - Vynaložené úsilí v člověkoměsících (E/MM)
  - Počet chyb/KLOC
  - Cena/LOC
  - Počet stránek dokumentace/KLOC
    - LOC jsou závislé na programátorovi i jazyku

- Charakteristiku jedné metriky složitosti podrobně (cyklomatická, etc.):
  - McCabeovy flow grafy:



  - Cyklomatická složitost
    - Množina nezávislých cest v grafu
    - V(G) = E – N + 2
      - E - počet hran grafu
      - N - počet uzlů grafu
    - V(G) = P + 1
      - P - počet predikátových uzlů

- Co je Halsteadova složitost?
  - Softwarová metrika výpočtu složitosti na základě statistické analýzy kódu.
  - Používá tyto proměnné:
    - n_1 - počet unikátních operátorů
    - n_2 - počet unikátních operandů
    - N_1 - celkový počet operátorů
    - N_2 - celkový počet operandů
  - Pomocí nich podle specifických vzorečků definuje tyto metriky:
    - Length: $N = N_1 + N_2$
    - Vocabulary: $n = n_1 + n_2$

        - Estimated length: $\tilde{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2$
        - Purity ratio: PR = $\tilde{N}$ / N
        - Volume: $V = N \log_2 n$
            - Number of bits to provide a unique designator for each of the n items in the program vocabulary.
        - Program effort: E=V/L
            - L = V*/V
            - V* is the volume of most compact design implementation

- Popsat a napsat využití jedné metriky (viz. výše)
- Jaké metriky/parametry používá Coupling + popsat:
    - Data and control flow
        - d_i - input data parameters
        - c_i - input control parameters
        - d_o - output data parameters
        - c_o - output control parameters
    - Global
        - g_d - global variables for data
        - g_c - global variables for control
    - Environmental
        - w - fan in number of modules called
        - r - fan out number modules that call module
    - Metrics:
        - $M_c = k/m$, k =1
        - $m = d_i + ac_i + d_o + bc_o + g_d + cg_c + w + r$
            - a, b, c, k can be adjusted based on actual data

TESTOVÁNÍ

- Popsat integrační postupy:
    - Shora dolů (TDT)
        - Nejprve je implementováno jádro (kostra) systému.
        - Zkombinováno do minimální „skořápky“ systému.
        - Pro doplnění neúplných částí se použijí „protézy“ nahrazované postupně aktuálními moduly.
        - Odhaluje chyby analýzy a návrhu, je v souladu s prototypováním.
        - Nevýhody:
            - Složité objekty, moduly, nelze jednoduše zaměnit za „protézu“.
            - Výsledky testů na vyšších úrovních nemusí bý přímo „viditelné“.
    - Zdola nahoru (BUT)
        - Začne s individuálními moduly a sestavuje zdola.
        - Individuální jednotky (po testování jednotek) jsou kombinovány do subsystémů. Subsystémy jsou kombinovány do celku.
        - Klasický testovací proces s nadřazenými testovacími objekty - „drivers“.
        - Nevýhody:
            - Čas a náklady na konstrukci „drivers“ pro testování jsou obvykle vyšší, než u „protéz“.
            - Až v závěru vznikne program použitelný pro předvedení, ve formě „prototypu“.

CHYBY

- IBMovská ortogonální klasifikace defektů, popsat 2 příklady z této klasifikace:
    - **Funkce** - chyba ovlivňující schopnosti, rozhraní uživatelů, rozhraní výrobku, rozhraní s HW architekturou nebo globální datovou strukturou.
    - **Rozhraní** - chyba při interakci s ostatními komponentami nebo ovladači přes volání, makra, řídící bloky nebo seznamy parametrů.
    - **Ověřování** - chyba v logice programu, která selže při validaci dat a hodnot před tím, než jsou použity.
    - **Přiřazení** - chyba při inicializaci datové struktury nebo bloku kódu.
    - **Časování/serializace** - chyba, která zahrnuje časování sdílených a RT prostředků.
    - **Sestavení/balení/spojování** - chyba související s problémy s repozitory projektu, změnami vedení, nebo správou verzí.
    - **Dokumentace** - chyba, která ovlivňuje publikace a návody pro údržbu.
    - **Algoritmus** - chyba, která se týká efektivity nebo správnosti algoritmu nebo datové struktury, ne však jejich návrhu.

INSPEKCE

- Kdo je přítomný při inspekci a popsat:
    - Vedoucí týmu (moderátor), zapisovatel, autor, recenzenti / inspektoři
    - Cíle inspekce:
        - Odhalit chyby ve funkci, logice a implementaci software.
        - Ověřit, že zkoumaná položka splňuje požadavky.
        - Zajistit, že položka byla prezentována s použitím předdefinovaných standardů.
        - Zajistit jednotný vývoj.
        - Zvýšit řiditelnost projektů.
    - Inspekce může být typu neformálního (konverzace, přezkoušení mezi spolupracovníky, neformální prezentace), či formálního (formální prezentace, prohlídka, recenze, inspekce).
    - Inspektor by měl rozumět kontextu, projít si všechny materiály výrobku, vytvořit si poznámky a připomínky formulovat jako otázky.