[Zpět na přehled okruhů](../README.md)

# Počítačový systém v softwarovém inženýrství

Počítačový systém v softwarovém inženýrství zahrnuje interakci hardwaru a softwaru pro dosažení určitého cíle nebo řešení konkrétního problému. Zde jsou některé klíčové aspekty počítačových systémů v softwarovém inženýrství:

1. **Hardwarová a softwarová architektura:**
   * **Hardwarová architektura:** Zahrnuje fyzické komponenty počítače, jako jsou procesory, paměť, periferní zařízení a další. Architektura ovlivňuje výkonnost a schopnosti počítačového systému.
   * **Softwarová architektura:** Definuje strukturu a organizaci softwarových komponentů, které běží na hardwaru. Architektura zahrnuje návrh a rozdělení funkcionalit mezi různé části softwaru.
2. **Operační systém:**
   * **Operační systém (OS):** Je základním softwarem, který spravuje hardwarové a softwarové zdroje počítače. Zajišťuje komunikaci mezi hardwarovými komponenty a aplikacemi, řídí procesy, spravuje paměť a zajišťuje spolehlivý běh celého systému.
3. **Vývoj softwaru:**
   * **Životní cyklus softwarového vývoje:** Zahrnuje fáze od definice požadavků a návrhu přes implementaci a testování až po údržbu a aktualizace.
   * **Metodologie vývoje:** Různé metody a postupy pro řízení vývoje softwaru, jako jsou Vodopádový model, Agilní metody (Scrum, Kanban) nebo DevOps.
4. **Softwarové inženýrství a návrh:**
   * **Návrhové vzory:** Opakující se řešení pro běžné problémy v návrhu softwaru.
   * **Objektově orientovaný návrh (OOP):** Paradigma, které koncepci rozděluje do objektů, které mohou obsahovat data a funkce.
5. **Testování a kvalita softwaru:**
   * **Testování softwaru:** Identifikuje chyby a ověřuje správnost a spolehlivost softwaru.
   * **Zajišťování kvality:** Praktiky a procesy, které se snaží zajistit vysokou kvalitu softwaru od počátečního návrhu až po nasazení.
6. **Správa konfigurace a verzování:**
   * **Správa konfigurace:** Řídí změny v softwarových konfiguracích a sleduje vztahy mezi různými verzemi softwaru.
   * **Verzování:** Umožňuje sledování historie změn v kódu a umožňuje obnovení předchozích verzí.
7. **Bezpečnost:**
   * **Bezpečnostní mechanismy:** Implementace zabezpečení na úrovni softwaru, například šifrování dat, ověřování totožnosti nebo správa oprávnění.
8. **Distribuované systémy:**
   * **Distribuované aplikace:** Softwarové aplikace, které běží na více než jednom počítači a komunikují přes síť.
   * **Síťová komunikace:** Zahrnuje protokoly a techniky pro komunikaci mezi různými částmi distribuovaného systému.
9. **Správa projektu:**
   * **Nástroje a metody pro správu projektu:** Pomáhají organizovat práci týmu, plánovat úkoly a sledovat pokrok v rámci softwarového vývoje.

Toto jsou obecné oblasti a koncepty spojené s počítačovým systémem v softwarovém inženýrství. Pokud máte konkrétní otázky nebo potřebujete další informace o některém z těchto témat, dejte mi vědět!

# Nástroje pro vývoje software a jeho životní cyklus

Existuje mnoho nástrojů pro vývoj softwaru a správu jeho životního cyklu. Tyto nástroje pomáhají týmům v organizaci, sledování a optimalizaci procesů od počáteční fáze analýzy až po nasazení a údržbu. Zde je seznam několika kategorií nástrojů a několika konkrétních příkladů:

### **Správa projektu a spolupráce:**

1. **Jira (Atlassian):** Výkonný nástroj pro sledování úkolů, správu projektů a agilní plánování.

### **Správa verzování a kódu:**

1. **Git:** Distribuovaný systém pro správu verzí kódu.
2. **GitHub, GitLab:** Webové platformy pro hostování projektů postavených na Gitu, poskytující sledování chyb, kolaboraci a další funkce.

### **Návrh a vývoj:**

1. **IDE (Integrated Development Environment):**
   * **Visual Studio Code, IntelliJ IDEA:** IDE pro mnoho programovacích jazyků.
2. **Nástroje pro kolaborativní vývoj:**
   * **VS Live Share, CodeTogether:** Umožňují vývojářům spolupracovat v reálném čase.

### **Testování:**

1. **Selenium:** Automatizovaný nástroj pro testování webových aplikací.
2. **JUnit, NUnit, pytest:** Rámce pro jednotkové testování v různých programovacích jazycích.

### **Integrace a doručování:**

1. **GitLab CI/CD:** Nástroje pro automatizaci procesů integrace a doručování.
2. **Docker:** Platforma pro kontejnerizaci aplikací, což usnadňuje přenositelnost a nasazování.

### **Správa požadavků a analýza:**

1. **Confluence (Atlassian):** Nástroj pro sdílení a spolupráci na dokumentaci.

### **Monitorování a řízení chyb:**

1. **Datadog:** Nástroje pro sledování výkonu a analýzu dat.
2. **Jira:** Systémy pro sledování chyb a správu úkolů.

### **Dokumentace:**

1. **Swagger:** Nástroje pro dokumentaci API.
2. **Sphinx:** Generátory dokumentace kódu.

Toto jsou pouze některé příklady a existuje mnoho dalších specializovaných nástrojů v každé kategorii. Volba konkrétních nástrojů závisí na potřebách týmu, technologiích používaných ve vašem projektu a preferencích vývojářů.

# Git vs SVN

Subversion (SVN) a Git jsou oba systémy pro správu verzí, ale mají několik klíčových rozdílů, které ovlivňují způsob, jakým se používají a jakým způsobem zpracovávají historii změn v kódu.

### 1. **Distribuovaný vs. Centrální:**

* **Git:** Je distribuovaný systém pro správu verzí. Každý vývojář má kompletní repozitář na svém lokálním počítači, což umožňuje offline práci, rychlé operace a vytváření větví a slučování bez potřeby připojení k centrálnímu serveru.
* **SVN:** Je centrální systém, což znamená, že všichni vývojáři se připojují k jednomu centrálnímu repozitáři pro stahování a odesílání změn.

### 2. **Revize:**

* **Git:** Každý vývojář má celý repozitář na svém počítači, takže revize jsou lokální a nezávislé na připojení k serveru. Každý commit má unikátní identifikátor (SHA-1 hash).
* **SVN:** Revize jsou přidělovány centrálně a jsou globálně identifikovány jedním číslem.

### 3. **Větvení a Slučování:**

* **Git:** Silné podporuje větvení a slučování. Vytváření nových větví je snadné a slučování změn je efektivní a často používané.
* **SVN:** Větvení a slučování jsou možné, ale může to být méně intuitivní a někdy obtížnější ve srovnání s Gitem.

### 4. **Rychlost:**

* **Git:** Je obecně považován za rychlejší, zejména při operacích, které nevyžadují připojení k serveru.
* **SVN:** Může být pomalejší, zejména při operacích, které vyžadují přístup k centrálnímu serveru.

### 5. **Offline Práce:**

* **Git:** Umožňuje vývojářům pracovat offline a později odeslat své změny na server.
* **SVN:** Vyžaduje připojení k serveru pro většinu operací.

### 6. **Správa Adresářové Struktury:**

* **Git:** Nijak neomezuje strukturu adresářů. Máte jednoho hlavního správce, který má celou historii.
* **SVN:** Má pevnější strukturu adresářů a podporuje standardní adresářovou strukturu.

### 7. **Historie Smazaných Souborů:**

* **Git:** Uchovává historii smazaných souborů.
* **SVN:** Historie smazaných souborů se ukládá pouze v případě, že byl soubor smazán ve stejné revizi, ve které byl také přidán.

### 8. **Práce se Vzdálenými Větvemi:**

* **Git:** Vzdálené větve jsou snadno přístupné a manipulovatelné.
* **SVN:** Vzdálené větve mohou vyžadovat specifické operace.

Oba systémy mají své výhody a nevýhody a volba mezi nimi závisí na potřebách projektu a preferencích týmu. Git je v současné době mnohem populárnější, zejména kvůli jeho distribuované povaze a široké podpoře vývojářské komunity.

# Síťové a webové technologie + databázové systémy

Síťové webové technologie se zabývají vývojem a provozem aplikací a služeb na internetu. Zahrnují širokou škálu nástrojů, protokolů, jazyků a frameworků, které umožňují komunikaci a interakci mezi klienty a servery přes síť. Níže jsou uvedeny některé klíčové složky síťových webových technologií:

### 1. **Protokoly:**

* **HTTP/HTTPS (Hypertext Transfer Protocol/Secure):** Protokol pro přenos hypertextových dat mezi webovým prohlížečem a webovým serverem. HTTPS je zabezpečená verze HTTP, která využívá šifrování.
* **WebSocket:** Protokol umožňující dvoucestnou komunikaci mezi klientem a serverem přes jedno TCP spojení. Je vhodný pro aplikace s reálným časem.
* **FTP (File Transfer Protocol):** Protokol pro přenos souborů mezi klientem a serverem.

### 2. **Jazyky pro Klienta:**

* **HTML (Hypertext Markup Language):** Značkovací jazyk používaný pro strukturování obsahu webových stránek.
* **CSS (Cascading Style Sheets):** Jazyk pro popis vizuálního vzhledu HTML dokumentů.
* **JavaScript:** Skriptovací jazyk, který umožňuje interaktivitu na straně klienta. Populární frameworky jsou např. React.

### 3. **Jazyky pro Server:**

* **PHP:** Serverový skriptovací jazyk často používaný pro vytváření dynamických webových stránek - hodně zastaralé.
* **Node.js:** JavaScriptový framework na straně serveru, který umožňuje asynchronní I/O a tvorbu škálovatelných webových aplikací.

### 4. **Databázové Technologie:**

* **MySQL:** Relační databázové systémy často používané pro ukládání dat webových aplikací.
* **MongoDB:** NoSQL databázové systémy vhodné pro ukládání nestrukturovaných nebo polostrukturovaných dat.

### 5. **Frameworky:**

* **Express.js:** Minimální a flexibilní Node.js framework pro tvorbu webových a mobilních aplikací.
* **Django:** Pythonový framework, který usnadňuje rychlý vývoj robustních webových aplikací.

### 6. **API (Application Programming Interface):**

* **REST (Representational State Transfer):** Architektonický styl pro návrh webových služeb, který využívá standardní HTTP protokol.
* **GraphQL:** Dotazovací jazyk pro API, který umožňuje klientům specifikovat, jaká data potřebují.

### 7. **Webové Servery:**

* **Apache:** Open-source webový server, často v kombinaci s PHP.

Tyto technologie a nástroje společně umožňují vývojářům vytvářet moderní a výkonné webové aplikace a služby. Je důležité sledovat aktuální trendy a novinky v této oblasti, protože se technologie neustále vyvíjejí.

# Cloud a webové API

1. **Cloud (Oblak):**
   * *Co to je?* Cloud, neboli “oblak,” je zjednodušený termín pro označení internetu. Je to prostředí, kde můžete ukládat, spravovat a přistupovat k datům a aplikacím prostřednictvím internetu namísto lokálního úložiště nebo zařízení. Firmy poskytují cloudové služby, které umožňují uživatelům vzdálený přístup k různým zdrojům a funkcím.
   * *Jak to funguje?* Cloudové služby jsou poskytovány prostřednictvím serverů, které jsou umístěny v datových centrech. Uživatelé mohou přistupovat k těmto službám prostřednictvím internetu a využívat je podle svých potřeb, aniž by potřebovali vlastnit nebo spravovat fyzické hardware.
2. **Webové API (Application Programming Interface):**
   * *Co to je?* Webové API je sada pravidel a protokolů, které umožňují komunikaci mezi různými softwarovými aplikacemi. Je to způsob, jakým programy komunikují a sdílejí data mezi sebou.
   * *Jak to funguje?* Webová API poskytuje sadu konkrétních pravidel, která určují, jaké operace a funkce může aplikace provádět. Pomocí HTTP (Hypertext Transfer Protocol) nebo jiných protokolů může jedna aplikace zasílat požadavky na webové API jiné aplikace, aby získala nebo aktualizovala data. To umožňuje integraci různých systémů a aplikací.

Kombinací obou - cloudových služeb a webových API - můžete vytvářet moderní, propojené a flexibilní aplikace, které mohou využívat širokou škálu zdrojů a poskytovat různé služby. Například můžete mít webovou aplikaci hostovanou v cloudu, která komunikuje s webovým API pro získání a aktualizaci dat.

# Softwarový proces

Softwarový proces (nebo vývojový proces) je systematický postup, který vývojový tým používá k plánování, navrhování, vytváření, testování a udržování softwarového produktu. Cílem je dosáhnout kvalitního, efektivního a spolehlivého softwaru. Existuje několik modelů a přístupů k softwarovým procesům, z nichž některé jsou:

### 1. **Vodopádový Model:**

* **Charakteristika:** Lineární postup, kde každá fáze začíná až po dokončení předchozí.
* **Fáze:** Požadavky, Návrh, Implementace, Testování, Údržba.
* **Výhody:** Jednoduchý a snadno pochopitelný.
* **Nevýhody:** Může být nepružný a obtížný pro změny.

### 2. **Agilní Model:**

* **Charakteristika:** Flexibilní přístup s důrazem na spolupráci, změny a dodávky hodnoty zákazníkovi.
* **Metody:** Scrum, Kanban, XP (Extreme Programming).
* **Výhody:** Rychlé reakce na změny, spolupráce s klientem.
* **Nevýhody:** Vyžaduje aktivní účast klienta, některé organizace mohou mít problémy s agilní transformací.

### 3. **Inkrementální Model:**

* **Charakteristika:** Postupné přidávání nových funkcí a vylepšení k existujícím.
* **Fáze:** Postupně přidávané inkrementy s novými funkcemi.
* **Výhody:** Postupné dodávky, možnost rychlého nasazení částí produktu.
* **Nevýhody:** Požaduje pečlivé plánování a identifikaci klíčových funkcí.

Každý model má své výhody a nevýhody, a vhodný přístup se může lišit v závislosti na povaze projektu, jeho velikosti, potřebách zákazníka a preferencích týmu. V dnešní době jsou agilní přístupy stále více preferovány pro svou schopnost flexibilně reagovat na změny v průběhu vývoje.

# Role ve vývoji software

Samozřejmě, zkusím to napsat jednodušeji:

1. **Projektový Manažer:** Osoba, která plánuje, organizuje a koordinuje práci na projektu.
2. **Analitik :** Lidé, kteří rozumí potřebám zákazníka a vytvářejí plány, jak je splnit.
3. **Architekt:** Odborník na plánování, který rozhoduje, jak systém bude vypadat a jak bude fungovat.
4. **Vývojáři:** Lidé, kteří píšou kód a tvoří jádro programu.
5. **Tester:** Osoba, která zkouší program, aby našla chyby a ujistila se, že vše funguje jak má.
6. **UI/UX Designer:** Tým, který vytváří pěkný a snadno použitelný vzhled programu.
7. **DevOps Engineer:** Odborník, který automatizuje procesy a zajišťuje, aby všechno bylo hladké.
8. **Scrum Master (v Agilním Vývoji):** Osoba, která zajišťuje, aby tým správně pracoval podle agilních pravidel.

Tyto role spolupracují na vytvoření a zdokonalení softwaru, aby byl užitečný a uspokojoval potřeby uživatelů.

# Vodopádový proces

Vodopádový proces je model vývoje softwaru, který zahrnuje postupné a sekvenční kroky, kde každá fáze začíná až po dokončení předchozí. Tento model byl jedním z prvních přístupů v oblasti softwarového inženýrství. Vodopádový model je často znázorňován jako lineární postupný proces, kde každá fáze představuje konkrétní sadu aktivit.

### Fáze Vodopádového Procesu:

1. **Definice Požadavků (Requirements):**
   * V této fázi jsou shromážděny a analyzovány požadavky na softwarový produkt. Jedná se o jeden z klíčových kroků, který stanovuje, co by měl software dělat a jak by měl fungovat.
2. **Návrh (Design):**
   * Po definici požadavků následuje fáze návrhu, kde jsou navrženy architektura, rozhraní a datové struktury. Tato fáze specifikuje, jak bude software postavený na základě požadavků.
3. **Implementace (Implementation):**
   * V této fázi jsou navržené koncepty převedeny na skutečný kód. Programátoři začínají psát kód podle návrhu.
4. **Testování (Testing):**
   * Po implementaci následuje fáze testování, kde jsou ověřovány funkcionality, bezpečnost a další aspekty. Chyby a nedostatky jsou identifikovány a opravovány.
5. **Dodání (Deployment):**
   * Po úspěšném testování je software připraven k nasazení. Tato fáze zahrnuje instalaci software do produkčního prostředí a jeho připravení pro používání koncovými uživateli.
6. **Údržba (Maintenance):**
   * Poslední fáze je věnována údržbě, kde jsou opravovány chyby, aktualizace a provedení změn v reakci na nové požadavky nebo potřeby uživatelů.

### Charakteristiky Vodopádového Modelu:

* **Jednosměrnost:** Každá fáze vodopádového modelu musí být dokončena před tím, než začne následující.
* **Žádná zpětná vazba:** Počáteční definice požadavků by měla být co nejpevnější, a proto je obtížné nebo nemožné vrátit se zpět k předchozím fázím.
* **Vhodné pro stabilní požadavky:** Vodopádový model je vhodný pro projekty, kde jsou požadavky dobře definované a mění se minimálně během vývoje.

Vodopádový model má některé omezení, jako je obtížnost adaptace na změny po zahájení vývoje, a proto byl nahrazen modernějšími metodologiemi, jako je iterativní nebo agilní vývoj. Tyto modernější přístupy lépe zohledňují změny v požadavcích a umožňují vývoj týmu pružně reagovat na nové informace a požadavky uživatelů.

# RUP

Rational Unified Process (RUP) je iterativní a inkrementální procesní rámec pro softwarový vývoj, který byl vyvinut firmou Rational Software Corporation, jež byla později akvizována společností IBM. RUP byl publikován v 90. letech 20. století a byl navržen tak, aby poskytoval strukturovaný přístup k vývoji softwaru, který zahrnuje definované fáze, aktivity, role, a artefakty.

### Klíčové Charakteristiky RUP:

1. **Iterativní a Inkrementální:**
   * RUP zdůrazňuje iterativní a inkrementální přístup k vývoji, kde se vývoj postupně zlepšuje v průběhu několika opakování (iterací). Každá iterace přidává další funkcionalitu nebo opravuje předchozí nedostatky.
2. **Fáze Vývoje:**
   * **Inception (Počátek):** Identifikace cílů projektu, definice oblasti působnosti a první odhad nákladů a časových rámců.
   * **Elaboration (Rozpracování):** Detailní analýza, architektura systému, identifikace rizik, vytvoření plánu vývoje.
   * **Construction (Výstavba):** Implementace a testování systému v iterativních cyklech.
   * **Transition (Přechod):** Přechod od vývoje k provozu, nasazení systému a zajištění uživatelské podpory.
3. **Artefakty a Role:**
   * **Artefakty:** RUP definuje artefakty jako dokumenty, modely nebo skripty, které jsou vytvářeny během různých fází vývoje a slouží k dokumentaci, komunikaci a podpoře rozhodování.
   * **Role:** RUP přiřazuje různé role do projektu, jako jsou analytik, architekt, designer, tester, atd., přičemž každá role má své specifické odpovědnosti.
4. **Use Case-Oriented:**
   * RUP klade velký důraz na použití případů užití (use cases) při definici funkcionalit a interakcí systému s uživateli.
5. **Ovládání Kvality:**
   * Proces RUP zahrnuje ovládání kvality v průběhu vývoje, včetně pravidelných hodnocení a kontrol kvality artefaktů.
6. **Nástroje:**
   * RUP nezavazuje k používání konkrétních nástrojů, ale poskytuje podporu pro nástroje, které podporují proces a artefakty vytvářené během vývoje.

RUP byl v minulosti široce používán, zejména v projektovém prostředích, které byly zvyklé na strukturované a detailně zdokumentované přístupy. Nicméně, v průběhu času se agilní metodologie staly stále populárnější a mnoho organizací přešlo na tyto agilní přístupy, které jsou pružnější a lépe schopné reagovat na měnící se požadavky.

# UML

UML (Unified Modeling Language) je standardní jazyk pro vizualizaci, specifikaci, konstrukci a dokumentaci systémů, zejména v oblasti softwarového inženýrství. UML byl vytvořen jako sjednocený nástroj pro modelování a komunikaci mezi vývojáři, designéry a dalšími zúčastněnými stranami v procesu vývoje softwaru. Byl vyvinut skupinou odborníků v oblasti softwarového inženýrství a nyní je spravován sdružením Object Management Group (OMG).

### Klíčové Prvky UML:

1. **Diagramy:**
   * **Use Case Diagrams:** Popisují interakce mezi systémem a jeho uživateli, mapují funkční požadavky.
   * **Class Diagrams:** Modelují strukturu systému z pohledu tříd a jejich vztahů.
   * **Sequence Diagrams:** Zobrazují interakce mezi objekty nebo třídami v čase.
   * **Activity Diagrams:** Popisují postupné kroky v procesu nebo toku práce.
   * **State Machine Diagrams:** Modelují chování objektu nebo třídy v reakci na různé události.
2. **Třídy a Objekty:**
   * **Třída:** Reprezentuje šablonu nebo definici objektu a obsahuje atributy a metody.
   * **Objekt:** Konkrétní instance třídy v průběhu provádění programu.
3. **Vztahy:**
   * **Association:** Vztah mezi dvěma třídami, který popisuje, jak jsou objekty jedné třídy spojeny s objekty druhé třídy.
   * **Aggregation:** Silnější forma asociace, která reprezentuje „část-od“ vztah.
   * **Composition:** Ještě silnější forma asociace, kde jedna třída vlastní a ovládá druhou třídu.
4. **Aktivity a Stavy:**
   * **Aktivita:** Jednotlivý krok nebo činnost v rámci procesu.
   * **Stav:** Specifický stav, ve kterém může objekt nebo systém existovat.
5. **Use Cases:**
   * **Use Case:** Scénář popisující interakci mezi uživatelem a systémem, zaměřuje se na funkcionalitu systému.
6. **Diagramy Komponent:**
   * **Component Diagrams:** Zobrazují komponenty systému a jejich vzájemné vztahy.
7. **Modelování Balíčků:**
   * **Package Diagrams:** Organizují a strukturalizují prvky systému do balíčků.
8. **Rozšířené Notace:**
   * **Stereotypy:** Rozšiřují základní notace UML, aby lépe vyhovovaly potřebám specifickým pro daný projekt nebo doménu.
   * **Tagged Values:** Umožňují přidávat dodatečné informace k UML prvkům.

UML slouží jako efektivní nástroj pro komunikaci mezi vývojáři, designéry a zainteresovanými stranami v průběhu vývoje softwaru. Poskytuje standardizovaný způsob popisu a modelování softwarových systémů, což pomáhá zajistit jednotnost a srozumitelnost v rámci týmu nebo organizace.

# Agilní metodiky vývoje software

Agilní metodiky vývoje softwaru představují skupinu přístupů a metodik, které klade důraz na pružnost, spolupráci, a adaptabilitu při vývoji softwaru. Tyto metody zdůrazňují iterativní a inkrementální přístup k vývoji, kde se software vyvíjí postupně a může být pružně měněn v průběhu projektu v reakci na změny požadavků.

Zde jsou některé z nejznámějších agilních metodik:

### 1. **Scrum:**

* **Charakteristika:** Scrum je rámec pro agilní vývoj, který se zaměřuje na rozdělení práce do krátkých, opakovatelných cyklů nazývaných “sprints”. Každý sprint má obvykle délku 2 až 4 týdny a končí dodáním funkčního produktu. Scrum zdůrazňuje pravidelné setkávání týmu (Scrum Daily Standup) a transparentnost pracovního postupu.

### 2. **Kanban:**

* **Charakteristika:** Kanban je metoda založená na vizuálním řízení pracovního postupu. Práce je reprezentována na Kanbanové tabuli, která má sloupce představující různé fáze vývoje (např. “To Do”, “In Progress”, “Done”). Tým omezuje množství práce, kterou může provádět ve stejnou dobu (WIP - Work In Progress), což napomáhá udržet efektivitu.

Tyto metody jsou pružné a mohou být upraveny podle specifických potřeb projektu. Agilní přístup k vývoji softwaru je stále populárnější díky své schopnosti rychle reagovat na změny, zapojení zákazníků a zdůraznění efektivní spolupráce v týmu.

[Zpět na přehled okruhů](../README.md)