# Pokročilé VCS – GIT

VCS (Version Control System) je klíčovým nástrojem pro správu verzí a spolupráci na softwarových projektech. GIT je jedním z nejpopulárnějších distribuovaných VCS. Jeho hlavními rysy jsou distribuovanost, rychlost a schopnost efektivně pracovat s velkými projekty.

Distribuovaný charakter GITu znamená, že každý vývojář má kompletní kopii repozitáře na svém lokálním počítači, což umožňuje efektivní offline práci a snadné větvení a slučování změn.

Jinak bych to asi porovnával s SVN a to je v základech komplet zpracovaný jako rozdíly.

# Principy tvorby software formou open source, návrh, vývoj.

Otevřený zdrojový kód (Open Source) je filozofie vývoje softwaru, která zdůrazňuje transparentnost, spolupráci a sdílení zdrojového kódu s komunitou. Zde jsou základní principy tvorby softwaru formou open source, zahrnující návrh a vývoj:

### 1. Transparentnost:

* **Otevřený přístup kódům:** Celý zdrojový kód projektu je veřejně dostupný a přístupný komunitě. Každý může zkontrolovat, jakým způsobem je software vytvořen.

### 2. Svoboda k úpravám:

* **Možnost přizpůsobení:** Každý má právo upravovat zdrojový kód podle svých potřeb. To podporuje inovace a přizpůsobení softwaru specifickým požadavkům uživatelů.

### 3. Otevřená spolupráce:

* **Kolaborativní vývoj:** Komunita může spolupracovat na vývoji softwaru, sdílet nápady a řešit problémy. To umožňuje zapojení široké škály talentovaných jednotlivců.

### 4. Licence open source:

* **Jasná licenční politika:** Projekt musí mít definovanou open-source licenci, která určuje podmínky použití, distribuce a modifikace kódu.

### 5. Veřejná komunikace:

* **Otevřená diskuse:** Komunikace o vývoji a rozhodnutích by měla být veřejná a přístupná všem. Diskuse mohou probíhat na fórech, issue trackeru nebo mailing listech.

### 6. Testování a recenze kódu:

* **Otevřený proces recenzí:** Komunita může zkoumat a recenzovat navržené změny v kódu. To zvyšuje kvalitu softwaru a zajišťuje, že změny jsou dobře promyšlené.

### 7. Stabilní vydání:

* **Pravidelná vydání:** Pravidelné vydávání nových verzí umožňuje uživatelům sledovat změny a využívat nové funkce nebo opravy chyb.

### 8. Záznam o změnách:

* **Transparentní historie změn:** Zaznamenání všech změn v kódu a dokumentace poskytuje přehled o vývoji projektu a umožňuje uživatelům sledovat jeho historii.

### 9. Inkluzivnost:

* **Otevřenost pro nové přispěvatele:** Projekt by měl být přístupný pro nové přispěvatele. Podpora pro různorodé přístupy a pohledy přináší bohatství komunitě.

### 10. Otevřený zpětný vazby:

* **Přijímání zpětné vazby:** Komunita by měla být otevřená k přijímání zpětné vazby od uživatelů a přizpůsobovat se měnícím se potřebám.

# Continous integration/continous delivery

CI/CD (Continuous Integration/Continuous Deployment) jsou moderní praktiky v oblasti softwarového vývoje, které mají za cíl zlepšit efektivitu, kvalitu a rychlost dodávek softwaru. Zde jsou jednoduché popisy obou termínů:

### Continuous Integration (CI) - Neustálá integrace:

Continuous Integration se zaměřuje na pravidelné slévání (integrování) změn kódu od členů týmu do sdíleného repozitáře. Hlavním cílem je minimalizovat konflikty a nalezení problémů v kódu co nejdříve v průběhu vývoje. Základní prvky CI zahrnují:

* **Automatické sestavení (build):** Neustálé sestavování zdrojového kódu pro ověření, že se projekt sestavuje bez chyb.
* **Automatizované testování:** Spuštění automatizovaných testů, aby se zajistilo, že nový kód neporušil existující funkčnost.
* **Code Review:** Kontrola kódu ostatními členy týmu na nalezení a opravu potenciálních problémů.

### Continuous Deployment (CD) - Neustálé nasazování:

Continuous Deployment se posouvá o krok dál a zaměřuje se na automatické nasazování hotového softwaru do produkčního prostředí. Cílem je minimalizovat manuální kroky a rychle dostat novou funkcionalitu nebo opravy chyb k uživatelům. Klíčové prvky CD zahrnují:

* **Automatické nasazování:** Automatický proces nasazování softwaru do produkčního prostředí po úspěšném dokončení procesu CI.
* **Monitorování a logování:** Sledování výkonu a chování aplikace v reálném provozu, což umožňuje rychlou detekci a opravu případných problémů.
* **Zpětná vazba a rollback:** Poskytnutí zpětné vazby o výsledku nasazení a schopnost vrátit se k předchozí stabilní verzi v případě problémů.

# Architektura a realizace nemocničních informačních systémů

Nemocniční informační systémy (NIS) jsou klíčovými nástroji pro správu informací a optimalizaci procesů v nemocnicích. Jejich architektura a realizace musí být navrženy tak, aby podporovaly efektivní a bezpečné poskytování zdravotní péče. Zde jsou některé klíčové aspekty, které by měly být zohledněny při architektuře a realizaci nemocničních informačních systémů:

### 1. Modularita a Integrace:

* **Modulární struktura:** Rozdělení systému do modulů usnadňuje údržbu, rozšiřitelnost a aktualizace.
* **Integrace s dalšími systémy:** Kompatibilita a integrace s dalšími zdravotnickými informačními systémy a zařízeními (např. laboratorními zařízeními, elektronickými zdravotními záznamy) jsou klíčové pro celkovou efektivitu.

### 2. Bezpečnost a Ochrana Soukromí:

* **Šifrování dat:** Zajištění, že data jsou šifrovaná a chráněná proti neoprávněnému přístupu.
* **Ochrana osobních údajů:** Dodržování právních a etických standardů pro ochranu osobních údajů pacientů.

### 3. Uživatelské Rozhraní:

* **Přizpůsobitelnost:** Uživatelské rozhraní by mělo být intuitivní a přizpůsobitelné potřebám různých uživatelů (lékařů, sester, administrativního personálu).
* **Přehlednost a jednoduchost:** Snaha minimalizovat složitost a zrychlit učení uživatelů.

### 4. Správa Dat:

* **Centrální úložiště:** Vytvoření centrálního úložiště pro zdravotnické informace pacientů.
* **Zálohování a obnova dat:** Pravidelné zálohování dat a mechanismy pro obnovu v případě havárie.

### 5. Podpora Procesů Zdravotní Péče:

* **Elektronické zdravotní záznamy (EHR):** Implementace EHR pro přehlednou a aktuální historii pacienta.
* **Sledování pacientů:** Funkce sledování pacientů v reálném čase a upozornění na potenciální problémy.

### 6. Řízení Přístupu:

* **Role a oprávnění:** Definování jasných rolí a oprávnění pro každého uživatele.
* **Audity a sledování přístupu:** Monitorování a auditování přístupu k citlivým datům.

### 7. Náklady a Efektivita:

* **Optimalizace procesů:** Návrh systému tak, aby optimalizoval a automatizoval procesy, což může přispět k úspoře času a nákladů.

# Analýza požadavků

Analýza požadavků (Requirements Analysis) je klíčovým krokem v procesu vývoje softwaru. Tato fáze má za úkol porozumět potřebám uživatelů a systémovým požadavkům, které musí být splněny. Zde je obecný rámec pro analýzu požadavků:

### 1. Komunikace s Stakeholdery:

* **Identifikace stakeholderů:** Určení všech zúčastněných stran, včetně uživatelů, zákazníků, vedoucích projektu a dalších zainteresovaných stran.
* **Sběr informací:** Provedení rozhovorů, průzkumu a workshopů k získání hlubšího porozumění potřebám a očekáváním stakeholderů.

### 2. Dokumentace Existujících Procesů:

* **Mapování procesů:** Získání přehledu o existujících procesech v organizaci, které nový systém může ovlivnit.
* **Identifikace problémů:** Zjišťování slabých míst v současných procesech, které by mohl nový systém řešit.

### 3. Definice Funkčních a Nefunkčních Požadavků:

* **Funkční požadavky:** Identifikace toho, co systém musí dělat. Popis konkrétních funkcí, které mají být implementovány.
* **Nefunkční požadavky:** Stanovení omezení a kvalitativních aspektů systému, jako jsou výkon, bezpečnost, dostupnost a škálovatelnost.

### 4. Validace a Verifikace Požadavků:

* **Validace:** Ověření, zda jsou identifikované požadavky skutečně potřebné a odpovídají cílům organizace.
* **Verifikace:** Zajištění, že požadavky jsou jasně definované, měřitelné, přenositelné a srozumitelné.

### 5. Prioritizace Požadavků:

* **Důležitost:** Stanovení, které požadavky mají nejvyšší prioritu a jsou klíčové pro úspěch projektu.
* **Rizika:** Identifikace rizik a jejich vliv na prioritizaci požadavků.

### 6. Prototypování a Modelování:

* **Prototypování:** Vytvoření prototypů nebo modelů, které mohou být použity k demonstrování funkcí systému a získání zpětné vazby od uživatelů.
* **UML diagramy:** Použití diagramů UML (Unified Modeling Language) pro vizualizaci a specifikaci struktury a chování systému.

### 7. Sledování Změn a Aktualizací:

* **Správa změn:** Stanovení procesu pro správu a sledování změn v požadavcích během celého vývoje.
* **Dokumentace aktualizací:** Zajištění, že veškeré změny jsou zdokumentovány a komunikovány stakeholderům.

### 8. Validace uživatelského zážitku (UX):

* **Testování uživatelského rozhraní:** Ověření, zda uživatelské rozhraní splňuje očekávání uživatelů a je snadno použitelné.
* **Zpětná vazba uživatelů:** Získání zpětné vazby od uživatelů prostřednictvím prototypů nebo testování uživatelského zážitku.

### 9. Dokumentace Požadavků:

* **Srozumitelná dokumentace:** Příprava komplexní dokumentace, která obsahuje všechny identifikované požadavky, scénáře uživatelských případů a další relevantní informace.

### 10. Zapojení Stakeholderů po Celý Čas:

* **Průběžná komunikace:** Udržování pravidelné komunikace se stakeholdery a průběžné zjišťování jejich potřeb a očekávání.
* **Agilní přístup:** Přijetí agilních metodologií může usnadnit pružnou reakci na změny během vývoje.

# Design architektury

Návrh architektury je klíčovým krokem v procesu vývoje softwaru. Během této fáze jsou definovány struktury, komponenty a interakce systému. Zde jsou některé klíčové aspekty, které by měly být zohledněny při návrhu architektury:

### 1. Identifikace Architektonických Požadavků:

* **Funkcionální požadavky:** Stanovení požadovaných funkcí systému.
* **Nefunkcionální požadavky:** Zohlednění nefunkcionálních aspektů, jako jsou výkon, bezpečnost, škálovatelnost a dostupnost.

### 2. Výběr Architektonického Stylu:

* **Monolitická architektura:** Jednotlivá, integrovaná aplikace.
* **Mikroslužby:** Rozdělení aplikace na malé, nezávislé služby.
* **Client-Server:** Oddělení prezentační vrstvy od logiky aplikace.
* **Capstone:** Architektura orientovaná na události.

### 3. Struktura a Komponenty:

* **Modularita:** Rozdělení systému do logických modulů a komponent, což zlepšuje údržbu a rozšiřitelnost.
* **Layered Architecture:** Rozvrstvení systému na vrstvy (např. prezentační, aplikační, datová).
* **Komunikace mezi komponentami:** Definování způsobu komunikace mezi jednotlivými komponentami.

### 4. Databázový Návrh:

* **Databázový model:** Návrh struktury databáze a relací mezi nimi.
* **Správa datových vztahů:** Zajištění konzistence a integrity dat.

### 5. Řízení Stavu:

* **Správa stavu:** Zohlednění, jak bude uchováván a spravován stav systému.
* **Uchovávání a obnovení stavu:** Zajištění možnosti ukládání a obnovení stavu aplikace (persistence).

### 6. Bezpečnostní Návrh:

* **Ochrana dat:** Zajištění, že citlivá data jsou řádně šifrována a chráněna.
* **Oprávnění a autentizace:** Definování oprávnění a autentizačních mechanismů.

### 7. Rozhraní a Uživatelská Zkušenost:

* **Uživatelské rozhraní:** Návrh intuitivního a uživatelsky přívětivého rozhraní.
* **API:** Definice veřejných rozhraní pro komunikaci s jinými systémy.

### 8. Škálovatelnost a Výkon:

* **Horizontální a vertikální škálovatelnost:** Možnost škálovat systém buď přidáním dalších instancí (horizontální) nebo zvětšováním kapacity jednotlivých komponent (vertikální).
* **Optimalizace výkonu:** Identifikace a optimalizace bottlenecks pro zajištění rychlé odezvy.

### 9. Správa Chyb a Logování:

* **Logování:** Implementace logování událostí pro sledování chyb a diagnostiku problémů.
* **Správa chyb:** Definování mechanismů pro zacházení s chybami a obnovení systému po selhání.

### 10. Testovatelnost:

* **Unit testy a integrační testy:** Zajištění, že jednotlivé komponenty jsou testovatelné odděleně.
* **Automatizované testování:** Implementace automatizovaných testů pro průběžné zajištění kvality.

### 11. Dokumentace Architektury:

* **Architektonický dokument:** Vytvoření detailní dokumentace popisující strukturu, rozhodnutí a důležité souvislosti v architektuře systému.
* **Dokumentace API:** Popis veřejného API pro vývojáře, kteří budou používat nebo integrovat systém.

# Design komponent systému

Design komponent systému je proces definování struktury softwarových komponent a jejich vzájemných vztahů tak, aby společně tvořily funkční a efektivní systém. Zde jsou klíčové kroky při návrhu komponent systému:

### 1. Identifikace Komponent:

* **Funkční moduly:** Rozdělení systému na logické funkční moduly nebo komponenty.
* **Nefunkční komponenty:** Identifikace komponent, které se starají o nefunkční aspekty, například bezpečnost, škálovatelnost, a persistenci dat.

### 2. Definice Rozhraní:

* **Externí rozhraní:** Definování veřejných rozhraní komponent, která jsou přístupná ostatním částem systému nebo externím systémům.
* **Interní rozhraní:** Určení, jak komponenty spolu komunikují.

### 3. Struktura Datových Toků:

* **Datové toky mezi komponentami:** Stanovení, jakým způsobem data proudí mezi jednotlivými komponentami.
* **Transformace dat:** Určení, jak jsou data transformována během tohoto toku.

### 4. Rozvržení Komponent na Architektonických Vrstvách:

* **Rozvržení do vrstev:** Při použití vrstevní architektury, uspořádání komponent do logických vrstev (např. prezentační, aplikační, datová vrstva).
* **Rozhraní mezi vrstvami:** Definice rozhraní mezi jednotlivými vrstvami.

### 5. Identifikace Závislostí:

* **Závislosti mezi komponentami:** Identifikace vzájemných závislostí mezi jednotlivými komponentami a moduly.
* **Způsoby komunikace:** Určení způsobů komunikace a sdílení dat mezi komponentami.

### 6. Rozhraní pro Externí Systémy:

* **API a služby:** Definice API (rozhraní pro programování aplikací) pro externí systémy a služby, které mohou být využívány.
* **Standardy pro integraci:** Zajištění, že rozhraní jsou navržena v souladu se standardy pro snadnou integraci.

### 7. Škálovatelnost a Efektivita:

* **Škálovatelnost komponent:** Zajištění, že jednotlivé komponenty jsou navrženy tak, aby bylo možné je škálovat horizontálně nebo vertikálně.
* **Optimalizace výkonu:** Identifikace a optimalizace komponent s ohledem na výkon a odezvu systému.

### 8. Bezpečnostní Aspekty:

* **Bezpečnostní komponenty:** Začlenění bezpečnostních komponent a funkcí pro zajištění bezpečného chodu systému.
* **Oprávnění a autentizace:** Implementace oprávnění a autentizačních mechanismů na úrovni komponent.

### 9. Dekompozice Komplexních Komponent:

* **Dělení komplexních komponent:** Pokud jsou některé komponenty příliš komplexní, rozdělení je do menších a snadněji spravovatelných částí.
* **Vztahy mezi částmi:** Zajištění, aby nové komponenty vzájemně správně komunikovaly a byly vzájemně provázané.

### 10. Testovatelnost:

* **Jednotkové testy:** Zajištění, že každá komponenta může být testována samostatně.
* **Integrační testy:** Testování vzájemné komunikace mezi komponentami.

### 11. Dokumentace:

* **Dokumentace architektury:** Vytvoření detailní dokumentace popisující strukturu, rozhodnutí a důležité souvislosti v návrhu komponent systému.
* **Diagramy a schémata:** Použití vizuálních nástrojů pro lépe srozumitelnou prezentaci návrhu.

# Testování a nasazení

Testování a nasazení jsou klíčovými fázemi v životním cyklu vývoje softwaru. Tato fáze zahrnuje ověření, zda vytvořený software splňuje stanovené požadavky a jestli je připravený k nasazení do produkčního prostředí. Zde jsou klíčové kroky pro testování a nasazení:

### Testování:

1. **Unit Testy:**
   * Testování jednotlivých částí (modulů) kódu na izolované úrovni.
   * Cílem je ověřit správnost funkcionality každé jednotlivé komponenty.
2. **Integrační Testy:**
   * Testování vzájemné interakce mezi jednotlivými komponentami nebo moduly.
   * Ověření, že integrované části spolupracují správně.
3. **Systémové Testy:**
   * Komplexnější testování celého systému.
   * Kontrola splnění všech funkcionálních a nefunkčních požadavků.
4. **Akceptační Testy:**
   * Ověření, zda systém splňuje očekávání uživatelů a zákazníků.
   * Často prováděno uživateli nebo týmem zákazníka.
5. **Výkonové a Zátěžové Testy:**
   * Ověření schopnosti systému zvládat vysoké zátěže nebo zjistit slabá místa výkonu.
   * Identifikace možných bottlenecků.
6. **Zabezpečovací Testy:**
   * Kontrola odolnosti systému vůči různým bezpečnostním hrozbám.
   * Testování zranitelností a správnosti implementace bezpečnostních opatření.
7. **Automatizované Testy:**
   * Vytvoření automatizovaných skriptů pro pravidelné spouštění testů.
   * Zajištění rychlé zpětné vazby během vývoje.

### Nasazení:

1. **Příprava na Nasazení:**
   * Zkontrolovat, zda jsou všechny testy úspěšné.
   * Zajistit, že všechny potřebné závislosti a konfigurace jsou správně nastaveny.
2. **Plánování Nasazení:**
   * Plánovat termín a postup nasazení do produkčního prostředí.
   * Komunikovat s relevantními stakeholdery ohledně možných výpadků.
3. **Zálohování Dat:**
   * Provést kompletní zálohu dat před nasazením pro případné obnovení v případě problémů.
4. **Automatizované Nasazení:**
   * Použít automatizované nástroje pro nasazení, což minimalizuje lidské chyby a zrychluje proces.
5. **Monitorování a Řízení Chyb:**
   * Monitorovat chod aplikace po nasazení.
   * Připravit plán na rychlé řešení problémů, pokud něco selže.
6. **Verifikace a Validace:**
   * Ověřit, zda systém funguje správně v produkčním prostředí.
   * Zkontrolovat, zda jsou všechny funkcionalit a požadavky splněny.
7. **Komunikace a Dokumentace:**
   * Informovat uživatele a stakeholdery o nové verzi.
   * Zaktualizovat dokumentaci a znovu poskytnout školení uživatelům, pokud je to potřeba.
8. **Rollback Plán:**
   * Mít plán pro případné rollback, pokud by došlo k nečekaným problémům po nasazení.
9. **Zhodnocení a Optimalizace:**
   * Shromáždit zpětnou vazbu od uživatelů a systémových operátorů.
   * Identifikovat oblasti pro optimalizaci a vylepšení do budoucna.