9. Softwarové inženýrství

Pokročilé VCS – GIT

Systémy správy verzí (VCS, Version Control System)
jsou základním nástrojem v moderním softwarovém vývoji. Umožňují evidovat historii změn zdrojového kódu, vracet se ke starším verzím a podporují týmovou spolupráci.

GIT
je dnes nejrozšířenější distribuovaný systém správy verzí. Každý uživatel má svou plnou kopii repozitáře včetně historie, což umožňuje rychlou a flexibilní práci offline, efektivní větvení a slučování větví (branching/merging) a minimalizaci rizika ztráty dat.

Na rozdíl od centralizovaných systémů jako
SVN (Subversion)
kde existuje jediný centrální repozitář, u GITu se pracuje s větší autonomií a bez závislosti na stálém připojení k serveru.

Testování software

Testování softwaru je proces ověřování, že aplikace nebo systém odpovídají požadavkům a fungují správně v různých situacích. Cílem je odhalit chyby co nejdříve, minimalizovat rizika a zajistit vysokou kvalitu výsledného produktu.

Hlavní typy testů zahrnují:

- **Unit testy:** Testování izolovaných částí kódu (funkcí, metod).
- **Integrační testy:** Ověření správné spolupráce mezi komponentami.
- **Systémové testy:** Testování celého systému jako celku.
- **Akceptační testy:** Testy z pohledu uživatele, zda splňuje očekávání.
- **Výkonové a zátěžové testy:** Měření rychlosti a stability při zátěži.
- **Bezpečnostní testy:** Ověření odolnosti vůči hrozbám (penetrace, zranitelnosti).

V moderním vývoji se často využívá **automatizované testování**, kdy jsou testy pravidelně spouštěny například v rámci CI/CD pipeline.

Principy tvorby software formou open source

Open source představuje filozofii vývoje, kde je zdrojový kód veřejně dostupný, každý jej může studovat, měnit a sdílet.

Klíčové principy open source vývoje jsou:

- **Transparentnost:** Každý má přístup ke zdrojovému kódu a může sledovat jeho vývoj.
- **Svoboda úprav:** Kdokoliv může kód přizpůsobit svým potřebám.
- **Otevřená spolupráce:** Na vývoji se podílí široká komunita, což urychluje inovace a zvyšuje kvalitu.
- **Licenční podmínky:** Projekt musí mít jasně danou open source licenci (např. GPL, MIT), která stanovuje pravidla použití a šíření.
- **Veřejná diskuse:** Vývoj, návrhy a změny se projednávají otevřeně, často ve formě issue trackerů, diskusních fór a mailing listů.
- **Kódové recenze:** Komunita má možnost navržené změny prohlížet a komentovat.
- **Pravidelná vydání:** Nové verze jsou zveřejňovány průběžně, uživatelé mají k dispozici aktuální i historické verze.
- **Inkluzivita:** Projekt je otevřený novým přispěvatelům bez ohledu na jejich původ či úroveň zkušeností.
- **Zpětná vazba:** Aktivní přijímání zpětné vazby od uživatelů i vývojářů.

Návrh a vývoj softwaru

Proces vývoje softwaru zahrnuje řadu aktivit, od počátečního sběru požadavků až po nasazení a údržbu systému.

Návrh softwaru zahrnuje analýzu potřeb, modelování systémové architektury, návrh komponent, uživatelského rozhraní a datových struktur.

Ve vývoji se často uplatňují agilní metodiky (např. Scrum, Kanban), které kladou důraz na iterativní postup, rychlé reakce na změny a úzkou spolupráci s uživateli.

Kromě toho se často využívají nástroje pro správu požadavků (Jira, Trello), diagramy UML pro vizualizaci systému a automatizace sestavování a testování v CI/CD.

Continuous integration/continuous delivery

Continuous Integration (CI) znamená pravidelné začleňování změn kódu do sdíleného repozitáře, kde se změny ihned automaticky sestavují a testují. Cílem je rychle odhalit chyby, zamezit konfliktům a zvýšit kvalitu výsledného softwaru.

Continuous Delivery/Deployment (CD) posouvá tento princip dále – po úspěšném otestování lze změny automaticky nebo poloautomaticky nasadit až do produkčního prostředí.

Hlavní přínosy CI/CD:

- **Automatizace buildů a testů** – každý commit automaticky spustí sestavení a testování.
- **Automatizované nasazení** – možnost rychlého a bezpečného nasazení nových verzí.
- **Rychlá zpětná vazba** – chyby jsou detekovány v rané fázi, snazší rollback.
- **Zvýšená produktivita a kvalita** – vývojáři mohou rychle iterovat a vydávat nové funkce.

Architektura a realizace nemocničních informačních systémů

Nemocniční informační systémy (NIS) jsou komplexní softwarové platformy určené pro správu zdravotnické dokumentace, plánování léčby, komunikaci mezi odděleními a správu administrativy v nemocnicích.

Klíčové aspekty architektury a realizace:

- **Modularita:** Systém je rozdělen do modulů (evidence pacientů, lékařské záznamy, laboratorní data, účetnictví aj.), což usnadňuje údržbu a rozšiřitelnost.
- **Integrace:** NIS se musí propojit s dalšími systémy (například laboratorními analyzátory, dalšími IS, elektronickými zdravotními záznamy, PACS).
- **Bezpečnost:** Data pacientů musí být chráněna před neoprávněným přístupem (šifrování, auditní záznamy, řízení přístupu dle rolí).
- **Uživatelská přívětivost:** Různorodí uživatelé (lékaři, sestry, administrativní pracovníci) musí být schopni systém efektivně ovládat, rozhraní by mělo být intuitivní.
- **Dostupnost a zálohování:** Data musejí být dostupná 24/7, systém musí mít zálohování a obnovu v případě havárie.
- **Podpora zdravotnických procesů:** Sledování pacientů, elektronická dokumentace, upozornění na důležité události.

Analýza požadavků

Analýza požadavků je zásadní fází vývoje softwaru, kde se zjišťují, upřesňují a validují potřeby zadavatele, uživatelů i dalších zainteresovaných stran.

Základní kroky:

- **Identifikace stakeholderů:** Určení všech osob a skupin, které budou ovlivněny systémem.
- **Sběr požadavků:** Rozhovory, workshopy, analýza stávajících procesů.
- **Rozdělení požadavků:** Na funkční (co má systém dělat) a nefunkční (jak to má dělat, např. rychlost, bezpečnost).
- **Prioritizace:** Stanovení, které požadavky mají nejvyšší prioritu.
- **Prototypování:** Tvorba návrhů rozhraní a funkčnosti, získání zpětné vazby.
- **Validace a verifikace:** Ověření, že požadavky odpovídají skutečným potřebám.
- **Sledování změn:** Požadavky je nutné průběžně aktualizovat, spravovat a dokumentovat.

Design architektury

Návrh architektury určuje základní stavební bloky systému, jejich strukturu a vzájemné vztahy.

Zásady návrhu architektury:

- **Výběr architektonického stylu:** Např. monolit, mikroslužby, klient-server, vícevrstvá architektura.
- **Modularita:** Rozdělení systému do modulů/komponent s jasně definovanými rozhraními.
- **Správa dat:** Návrh datových modelů, databází, uložení a integrace dat.
- **Bezpečnost:** Ochrana citlivých dat, autentizace, oprávnění, šifrování.
- **Škálovatelnost:** Systém musí být připraven růst (více uživatelů, více dat).
- **Rozhraní (API):** Definice způsobů komunikace s jinými systémy.
- **Dokumentace:** Architektura musí být srozumitelně popsána pro další vývojáře a správce.

Design komponent systému

Návrh komponent systému se zaměřuje na rozdělení systému do jednotlivých částí, které mají konkrétní odpovědnost (funkcionalitu).

- **Identifikace komponent:** Stanovení hlavních logických částí systému.
- **Definice rozhraní:** Specifikace způsobu komunikace mezi komponentami.
- **Datové toky:** Jak data proudí systémem, případné transformace dat.
- **Vrstevní struktura:** Rozdělení na vrstvy (prezentační, aplikační, datová).
- **Závislosti:** Určení vztahů a vazeb mezi komponentami, minimalizace provázanosti.
- **Škálovatelnost a testovatelnost:** Komponenty mají být nezávislé, snadno rozšiřitelné a dobře testovatelné.
- **Bezpečnost:** Zajištění, že jednotlivé komponenty budou správně řídit přístup k datům a funkcím.
- **Dokumentace:** Každá komponenta by měla mít jasný popis své funkce a rozhraní (např. pomocí UML diagramů).

Testování a nasazení

Testování a nasazení uzavírají životní cyklus vývoje softwaru a zajišťují, že aplikace je připravena na ostré nasazení.

Testování zahrnuje:

- **Jednotkové (unit) testy:** Testují izolované části kódu.
- **Integrační testy:** Prověřují spolupráci mezi komponentami.
- **Systémové a akceptační testy:** Ověřují správné fungování celého systému a splnění požadavků.
- **Výkonové a bezpečnostní testy:** Zajišťují stabilitu při zátěži a odolnost vůči hrozbám.
- **Automatizované testy:** Zrychlují vývoj a snižují počet regresních chyb.

Nasazení zahrnuje:

- **Přípravu:** Zálohování dat, kontrolu prostředí, nastavení konfigurací.
- **Automatizované nasazení:** Minimalizuje chyby a zrychluje přechod do produkce.
- **Monitorování:** Sledování funkčnosti, výkonu a chování systému po nasazení.
- **Rollback plán:** Možnost rychlého návratu k předchozí verzi v případě problémů.
- **Komunikace:** Informování uživatelů a správců o změnách a nových funkcích.
- **Optimalizace:** Sběr zpětné vazby, odstraňování chyb a další rozvoj systému.