24. Testování a dokumentace

1. Význam

- jedná se o podstatnou část vývoje softwaru (waterfall)
- bez testování bychom si nebyli jistí správnou funkčností softwaru

2. Základní principy a nástroje

- pro vývoj se můžou například použít verzovací systémy, GIT
  - umožňuje trasovat, dělat regrese a dělat branche (případně rozdělit vývoj, poté se uchýlit k nejlepší verzi)
- *debugger* program, který je schopný sledovat průběh programu
- **tracing** se také používá při ladění softwaru, program generuje datalogger, z něho zpětně můžeme vyčíst, kde se případně stala chyba, diagnostika programu
- **remote-debugging**: debugger běží na jiném systému než samotný program

3. Metoda waterfall

- jedná se sekvenční vývojový proces, skládá se z *návrhu*, *implementace návrhu*, *validace implementace*, *integrace*, *údržby*
- kritizuje se často za to, že není prakticky možné dovést jednu fázi fázi životnho cyklu produktu k dokonalosti předtím, než se přejde k fázi následující, jednotlivé kroky jsou na sobě závislé a je obtížné se ve vývoji příliš vracet, není dovoleno prototypovat
- sašimi model je totéž co waterfall, akorát mají jednotlivé kroky mezi sebou zpětnou vazbu, takže to částečně řeší jeho problém o dokončených cyklech

4. Dokumentace

- psaný text, který doprovází daný software
- **dokumentace požadavků**: schopnosti systému, základ realizace
- **dokumentace architektury**: přehled softwaru
- **technická dokumentace**: dokumentace kódu, API, user interface
- **uživatelská dokumentace**: manuál pro koncového uživatele
- **marketingová dokumentace**: analýza trhu

5. Testování

- **Unit testing**: testuje se pouze jedna část programu (procedura, funkce programu, …), pakliže projde testem, přejde se na další, v ideálním případě by testovaná jednotka neměla být závislá na okolním programu, proto se vytvářejí pomocné objekty, které simulují okolní program
- **TDD**: test-driven development, testování je založeno na malých, stále se opakujících se krocích, které vedou k zeefektivnění vývoje, takže *unit testy*, cyklus je následovný:
  1. *Napsat test*: první napíšeme test, která má danou funkci programu ověřit, čímž se eliminuje odchýlení od záměru programu
  2. *Spustit test*: samozřejmě, že neprojde
  3. *Napsat program*: napsat kód tak, aby prošel následující fází

4. *Testování*: pakliže program neprojde, je dále vyvíjen, pakliže projde, jde na další fázi
5. *Refraktorace*: jinými slovy optimalizace, po každé změně provést 4. krok

– nevýhoda TDD spočívá v tom, že samotné testy mohou obsahovat chyby a také testy zvětšují velikost softwaru → nižší produktivita

- **Regresivní testování**: provádí se potom, co byly na softwaru provedeny změny, aktualizace, pokud se chyby objeví, říkáme, že došlo k *regresu*