

Základy testování a definice

Dovednosti, které byste jako tester měli mít:

- Analytické a logické myšlení
- Plánování a řízení úkolů a času
- Ochota učit se
- Hledání vylepšení ve stávajících řešeních
- Dobré verbální a písemné komunikační dovednosti
- Schopnosti týmové práce
- Vnímavost
- Smysl pro detail
- Znalost angličtiny na komunikativní úrovni
- Trpělivost

Testování

Obecně je testování proces zkoušení nebo ověřování, zda určitý výrobek, systém, služba nebo koncept splňuje stanovené požadavky či očekávání. Cílem je zjistit, zda funguje podle plánu, a odhalit případné chyby nebo nedostatky. Testování se používá v mnoha oblastech, nejen v softwaru, například:

- **Výrobní průmysl:** Testování fyzických produktů, jako jsou auta, elektronika nebo spotřební zboží, aby byla zajištěna jejich kvalita a bezpečnost.
- **Věda a výzkum:** Experimentální testování hypotéz, zařízení nebo technologií.
- **Vzdělávání:** Testování znalostí a dovedností studentů pomocí zkoušek nebo testů.
- **Medicína:** Testování nových léků nebo léčebných postupů ve klinických studiích.

Ve všech těchto případech je testování nástroj pro kontrolu kvality, který pomáhá identifikovat problémy před tím, než je produkt, služba nebo myšlenka uvedena do praxe.

Testování softwaru

Testování softwaru je proces, při kterém se ověřuje a validuje, zda software splňuje specifikované požadavky a funguje podle očekávání. Cílem testování je identifikovat chyby, problémy nebo nedostatky v softwaru, aby byly opraveny před jeho uvedením do provozu.

Softwarové testování se využívá v různých oblastech, kde je potřeba zajistit kvalitu a spolehlivost softwaru. Zde jsou některé z nejčastějších oblastí, kde se SW testování pravidelně provádí:

1. Webové aplikace

- **Příklady:** Internetové obchody, sociální sítě, bankovní aplikace, vzdělávací platformy.
- **Důvody testování:** Testování zajišťuje funkčnost na různých prohlížečích a zařízeních, ověřuje se výkon při velkém zatížení (např. velký počet uživatelů současně) a testuje se bezpečnost (ochrana před útoky a úniky dat), uživatelské rozhraní (UI) a uživatelské zkušenosti (UX).

2. Mobilní aplikace

- **Příklady:** Aplikace pro iOS a Android, jako jsou bankovní aplikace, aplikace pro zasílání zpráv, herní aplikace nebo aplikace pro fitness.
- **Důvody testování:** Testuje se kompatibilita s různými zařízeními, optimalizace výkonu (rychlost, spotřeba baterie), interakce s uživatelem a správné fungování na různých verzích operačního systému, UI a UX.

3. Desktopové aplikace

- **Příklady:** Kancelářské balíky (např. Microsoft Office), grafické programy (např. Adobe Photoshop), antivirové programy, účetní software.
- **Důvody testování:** Kontrola funkčnosti na různých operačních systémech, testování integrace s dalšími nástroji, UI a UX.

4. E-commerce systémy

- **Příklady:** Online obchody, jako jsou Amazon, eBay nebo Shopify.
- **Důvody testování:** Testování nákupního procesu (od výběru produktů až po platbu), bezpečnostní testování (ochrana platebních údajů), výkon (rychlost načítání při velké zátěži) a kontrola dostupnosti.

5. Bankovní a finanční systémy

- **Příklady:** Internetové bankovníctví, platební brány, obchodní platformy.
- **Důvody testování:** Testuje se bezpečnost transakcí, výkon při vysokém objemu transakcí, ochrana osobních údajů, funkčnost API a správná integrace s dalšími systémy.

6. Zdravotnické aplikace

- **Příklady:** Systémy pro správu nemocnic, aplikace pro sledování zdraví, lékařské přístroje s integrovaným softwarem.
- **Důvody testování:** Kritická bezpečnost a spolehlivost (např. správnost diagnostických údajů), integrace s hardwarem (lékařské přístroje), ochrana citlivých údajů pacientů.

7. Automobilový průmysl

- **Příklady:** Systémy pro řízení vozidel, autonomní řízení, infotainment systémy.
- **Důvody testování:** Bezpečnostní testování (reakce na krizové situace), testování spolehlivosti softwaru v reálném čase, testování kompatibility s hardwarem vozidla.

8. Herní průmysl

- **Příklady:** Počítačové hry, konzolové hry, mobilní hry.
- **Důvody testování:** Zajištění plynulosti herního zážitku (bez lagů), testování kompatibility na různých platformách, ověřování interakcí mezi hráči (např. u online her) a testování výkonu při různých grafických nastaveních.

9. Cloudové aplikace a SaaS systémy (Software as a Service)

- **Příklady:** Platformy jako Salesforce, Google Workspace, Microsoft 365.
- **Důvody testování:** Zajištění škálovatelnosti (podpora růstu počtu uživatelů), bezpečnost dat, správné fungování cloudových funkcí (např. synchronizace mezi uživateli) a vysoká dostupnost.

10. Telekomunikační a síťové systémy

- **Příklady:** Systémy pro mobilní operátory, VoIP služby (např. Skype), síťové služby a routery.
- **Důvody testování:** Testování výkonu při vysokém zatížení, správná správa síťových připojení, ověření kvality zvuku/video u komunikačních aplikací.

Testování softwaru je kritické v jakémkoli odvětví, kde je software klíčovou součástí produktu nebo služby, aby bylo zajištěno jeho správné fungování, bezpečnost a spolehlivost pro uživatele.

Testování podle definice

Dle definice ISO/IEC/IEEE 29119: Testování softwaru je systematický proces vyhodnocování, zda software splňuje specifikované požadavky a identifikování chyb pomocí technik, jako jsou analýza, realizace a ověřování funkcionality, výkonu, zabezpečení a dalších charakteristik softwaru.

Podle této normy zahrnuje testování následující aspekty:

- **Zajištění kvality:** Proces ověřování, zda software splňuje dané specifikace a standardy kvality.
- **Detekce chyb:** Identifikace chyb, nesrovnalostí nebo nežádoucího chování ve funkčnosti nebo jiných oblastech softwaru.
- **Validace:** Proces ověřování, zda software plní očekávání uživatele a požadavky specifikované v zadání.
- **Verifikace:** Proces zajištění, že software byl vyvinut správně podle zadaných specifikací.

Testování tedy není jen o hledání chyb, ale i o ověřování, že software splňuje jak funkční, tak nefunkční požadavky, jako je výkon nebo bezpečnost.

Hlavní cíle testování

Cíle testování softwaru jsou klíčové pro zajištění, že software bude splňovat požadavky a fungovat správně. Hlavní cíle testování zahrnují:

- **Detekce chyb:** Primárním cílem testování je odhalit chyby (bugs) v softwaru, které by mohly způsobit nesprávné fungování nebo selhání systému.
- **Ověření splnění požadavků:** Testování má za cíl potvrdit, že software splňuje všechny funkční a nefunkční požadavky, které byly stanoveny během fáze návrhu.
- **Zajištění kvality:** Testování je zásadní pro zajištění kvality softwaru. To zahrnuje stabilitu, spolehlivost, bezpečnost, výkon a uživatelskou přívětivost.
- **Prevence chyb:** Testování pomáhá předcházet chybám tím, že poskytuje zpětnou vazbu vývojářům v rané fázi vývoje, což umožňuje opravit problémy ještě předtím, než se projeví v produkčním prostředí.
- **Snížení rizik:** Testování snižuje rizika nasazení softwaru do reálného prostředí. Odhalením kritických chyb před nasazením lze zabránit škodlivým důsledkům pro firmu nebo uživatele.
- **Ověření použitelnosti:** Testování zahrnuje i validaci uživatelského rozhraní a celkové použitelnosti, aby bylo zajištěno, že software je intuitivní a uživatelsky přívětivý.
- **Zajištění kompatibility:** Testování ověřuje, že software správně funguje na různých zařízeních, platformách, operačních systémech a prohlížečích.
- **Kontrola bezpečnosti:** Zajištění, že software neobsahuje zranitelnosti, které by mohly být zneužity k neoprávněnému přístupu, manipulaci nebo poškození dat.
- **Posouzení výkonu:** Ověření, zda software funguje efektivně při různém zatížení, jak rychle reaguje a jak se chová v různých situacích zátěže.

Každý z těchto cílů přispívá k dosažení celkové spolehlivosti a efektivity softwaru, což zajišťuje, že konečný produkt bude funkční, bezpečný a uživatelsky přívětivý.

Validace a verifikace

Validace: Kontrola, zda produkt splňuje očekávání a potřeby uživatele. Odpověď na otázku "Děláme správný produkt?"

Verifikace: Kontrola, zda je produkt vyroben správně podle specifikací a požadavků. Odpověď na otázku "Děláme produkt správně?"

Nejznámější selhání

1. NASA Mars Climate Orbiter (1999)

- **Co se stalo?** Mars Climate Orbiter byl ztracen kvůli chybě ve výpočtech. Jeden tým použil imperiální jednotky (libry-sekundy) a druhý tým používal metrické jednotky (newtony-sekundy).
- **Důsledek:** Sonda vstoupila do atmosféry Marsu příliš nízkou a shořela.
- **Poučení:** Testování a validace mezi různými systémy a týmy je nezbytné. Selhání koordinace mezi vývojáři a jejich nekompatibilita v měřicích jednotkách vedlo ke ztrátě projektu za 125 milionů dolarů.

2. Therac-25 Radiation Machine (1985-1987)

- **Co se stalo?** Zdravotnický přístroj Therac-25, určený k léčbě rakoviny, měl softwarovou chybu, která vedla k podání smrtelně vysokých dávek radiace pacientům.
- **Důsledek:** Minimálně pět lidí zemřelo a další byli těžce zraněni.
- **Poučení:** Kritické systémy v oblasti zdravotnictví musí být testovány extrémně důkladně. Zanedbání testování bezpečnostních mechanismů může mít fatální následky.

3. Patriot Missile Failure (1991)

- **Co se stalo?** Během války v Zálivu došlo k selhání Patriot raketového obranného systému. Software používal chybný výpočet časových intervalů, což vedlo k nesprávnému zaměření raket.
- **Důsledek:** Patriot nezasáhl raketu Scud, která zasáhla vojenský tábor a zabila 28 vojáků.
- **Poučení:** Testování reálných scénářů je klíčové, zvláště u obranných systémů. Malé časové odchylky mohou mít katastrofické následky.

4. Ariane 5 Flight 501 (1996)

- **Co se stalo?** Raketa Ariane 5 explodovala pouhých 37 sekund po startu kvůli přetečení hodnoty při konverzi 64bitového plovoucího čísla na 16bitové celé číslo.
- **Důsledek:** Zničení rakety a jejího nákladu stálo 370 milionů dolarů.
- **Poučení:** Testování přechodu softwaru z jednoho systému na druhý je zásadní. V tomto případě se znovu použil software z Ariane 4 bez důkladného otestování v kontextu nového systému.

5. Knight Capital (2012)

- **Co se stalo?** Investiční společnost Knight Capital přišla během 45 minut o 440 milionů dolarů kvůli softwarové chybě ve svém obchodním systému. Starý kód, který nebyl dostatečně otestován, byl aktivován během aktualizace softwaru.
- **Důsledek:** Firma musela být zachráněna investory a později prodána.
- **Poučení:** Testování softwarových změn a aktualizací je nezbytné, zejména v oblasti finančních trhů, kde může každý okamžik znamenat ztrátu obrovských částek.

6. British Airways IT Meltdown (2017)

- **Co se stalo?** Selhání IT systému vedlo ke zrušení tisíců letů a postihlo 75 000 cestujících. Hlavní problém spočíval ve špatně provedeném restartu systému po výpadku proudu.
- **Důsledek:** Ztráta milionů liber a poškození reputace.
- **Poučení:** Testování obnovy po havárii a schopnosti rychle se zotavit z výpadků je pro kritické systémy, jako jsou letecké společnosti, zásadní.

Sedm zásad testování

Testování ukazuje přítomnost vad, ne jejich nepřítomnost

Testování může ukázat, že v softwaru jsou chyby, ale nikdy nemůže prokázat, že žádné chyby neexistují.

Testování všeho není možné

Není možné otestovat všechny kombinace vstupů a stavů softwaru. Testování je vždy omezené a zaměřené na klíčové funkce.

Časné testování šetří čas a peníze

Testování by mělo začít co nejdříve v procesu vývoje softwaru, aby se chyby odhalily v raných fázích, což snižuje náklady na opravy.

Defekty se shlukují

Chyby se často nacházejí v několika malých částech systému, kde se opakovaně objevují problémy. Tyto oblasti by měly být testovány důkladněji.

Paradox pesticidů

Pokud se stejné testovací případy opakují znovu a znovu, přestanou odhalovat nové chyby. Je proto nutné pravidelně aktualizovat a rozšiřovat testovací scénáře.

Testování závisí na kontextu

Různé typy softwaru vyžadují různé přístupy k testování. Například kritické systémy potřebují důkladnější testování než běžné aplikace.

Neodhalení chyb neznamena kvalitní software

I když testování neodhalí žádné chyby, neznamena to, že software splňuje požadavky uživatelů nebo že je dostatečně kvalitní.

Nejdůležitější pojmy

- **Bug (Chyba):** Defekt nebo chyba v softwaru, která způsobuje jeho nesprávnou funkci nebo neočekávané chování.
- **Test Case (Testovací případ):** Soubor vstupů, podmínek a očekávaných výsledků pro testování konkrétní funkce nebo části softwaru.
- **Test Plan (Testovací plán):** Dokument, který popisuje cíle testování, rozsah, přístupy, zdroje a harmonogram pro testování softwaru.
- **Test Scenario (Testovací scénář):** Vysoká úroveň popisu toho, co by mělo být testováno. Scénář definuje specifickou situaci nebo cestu uživatele, kterou je třeba otestovat.
- **Test Execution (Provádění testů):** Proces spouštění testovacích případů a porovnání očekávaných a skutečných výsledků.
- **Regression Testing (Regresní testování):** Testování, které ověřuje, zda změny v softwaru (např. opravy chyb nebo nové funkce) nezpůsobily nové chyby v již existujících funkcích.
- **Smoke Testing:** Rychlé, základní testování, které ověřuje, zda hlavní funkce softwaru fungují správně po novém nasazení nebo změně.
- **Functional Testing (Funkční testování):** Testování zaměřené na kontrolu, zda software plní své funkční požadavky podle specifikace.
- **Non-Functional Testing (Nefunkční testování):** Testování zaměřené na aspekty softwaru, které nejsou spojeny s funkcionalitou, jako jsou výkon, bezpečnost, použitelnost atd.
- **Manual Testing (Manuální testování):** Proces, při kterém tester ručně spouští testovací případy bez použití automatizačních nástrojů.

- **Automated Testing (Automatizované testování):** Použití softwarových nástrojů k automatizaci spouštění testovacích případů, zejména těch, které jsou časově náročné nebo se často opakují.
- **Unit Testing (Jednotkové testování):** Testování jednotlivých komponent nebo jednotek kódu izolovaně, obvykle prováděné vývojáři.
- **Integration Testing (Integrační testování):** Testování, které ověřuje, že různé komponenty nebo moduly spolu správně fungují, když jsou integrovány.
- **User Acceptance Testing (UAT):** Konečné testování softwaru uživateli nebo zákazníky před nasazením, které ověřuje, že systém splňuje jejich požadavky a je připraven k používání.
- **Performance Testing (Výkonnostní testování):** Testování zaměřené na měření rychlosti, odezvy, stability a škálovatelnosti softwaru při různých zátěžích.
- **Load Testing (Zátěžové testování):** Typ výkonnostního testování, které ověřuje, jak se software chová při běžném nebo nadprůměrném zatížení.
- **Stress Testing (Zátěžové testování při extrémních podmínkách):** Testování zaměřené na zjištění, jak software reaguje na extrémní zatížení, které může překračovat běžné provozní podmínky.
- **Defect Lifecycle (Životní cyklus chyby):** Proces od nahlášení chyby, přes její analýzu, opravu, retestování, až po její uzavření.
- **Exploratory Testing (Průzkumné testování):** Testování bez předem definovaných testovacích případů, kde tester improvizuje a zkoumá software podle svých zkušeností.
- **API Testing:** Testování aplikačních rozhraní (API) zaměřené na ověření správné komunikace mezi softwarovými komponentami.
- **Agile Testing:** Testovací přístup, který je součástí agilního vývoje, kde testování probíhá souběžně s vývojem v malých a častých iteracích.
- **Test Environment (Testovací prostředí):** Konfigurace hardwaru, softwaru, sítě a dalších komponent, které jsou potřebné pro testování softwaru.
- **Continuous Integration (CI):** Proces, kdy vývojáři pravidelně integrují svůj kód do společného repozitáře a automatické testy se spouští pokaždé, když je nový kód vložen.
- **Severity (Závažnost):** Označuje, jak závažná je nalezená chyba a jak moc ovlivňuje funkčnost systému.
- **Priority:** Určuje, jak rychle by měla být chyba opravena, což závisí na jejím dopadu na uživatele nebo na byznys.
- **Test Coverage (Pokrytí testováním):** Míra, do jaké byly pokryty funkce nebo kód testy. Pokrytí může být funkční, kódové nebo na základě rizik.
- **Test Data (Testovací data):** Data, která se používají během testování pro simulaci reálných vstupů a scénářů.
- **Boundary Value Testing (Testování hraničních hodnot):** Testovací technika, která se zaměřuje na vstupy na hranicích povolených hodnot (minimální, maximální).
- **Backend:** je ta část aplikace, která funguje na serveru a stará se o logiku, zpracování dat, správu databází a komunikaci s frontendem.
- **Frontend:** je ta část aplikace nebo webu, kterou uživatel přímo vidí a se kterou interaguje.

Testovací případy a chyby

Testovací podmínka

Testovací podmínka (anglicky Test Condition) je specifický aspekt nebo okolnost systému, který může být testován s cílem ověřit jeho funkčnost nebo chování. Může to být jakákoli vlastnost softwaru, funkce, modul, požadavek nebo jakýkoli prvek, který je důležitý pro dosažení cílů testování.

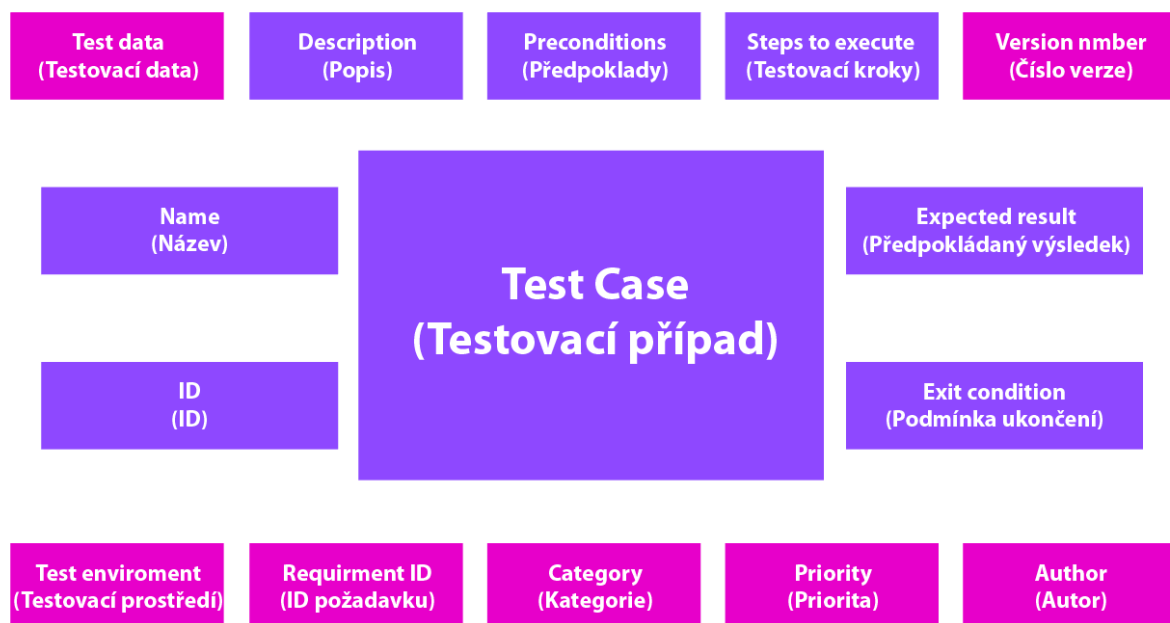
Příklady testovacích podmínek:

- **Funkční podmínka:** Například kontrola, zda tlačítko „Odeslat“ na formuláři skutečně odešle správně vyplněná data na server.
- **Nefunkční podmínka:** Testování, jak rychle se načítá stránka, pokud je více než 1000 uživatelů aktivních současně.
- **Bezpečnostní podmínka:** Ověření, že uživatelé bez oprávnění nemají přístup k citlivým datům.
- **Podmínka výkonu:** Zkontrolování, že systém zvládne zpracovat transakce v určitém časovém limitu.

Testovací podmínky jsou často definovány na základě požadavků nebo specifikací systému a slouží k přípravě testovacích scénářů a případů.

Testovací případ

Testovací případ (anglicky Test Case) je konkrétní sada podmínek, kroků a vstupů, které jsou navrženy k otestování určité funkce nebo části softwaru. Každý testovací případ má jasně definované vstupy, očekávané výsledky a kroky, které musí tester provést, aby ověřil, zda software funguje podle očekávání.



Struktura testovacího případu obvykle zahrnuje:

- **ID testovacího případu:** Jedinečný identifikátor pro snadné sledování.
- **Název testovacího případu:** Stručný popis toho, co bude testováno.
- **Předpoklady:** Podmínky, které musí být splněny před spuštěním testu (např. uživatel musí být přihlášen).

- **Testovací kroky:** Jasně definovaný seznam kroků, které musí tester provést.
- **Vstupy:** Hodnoty, které budou použity během testu (např. údaje, které se zadávají do formuláře).
- **Očekávaný výsledek:** Popis toho, co by se mělo stát po provedení testu, pokud je systém v pořádku.
- **Skutečný výsledek:** To, co se skutečně stalo během testování (doplní se po provedení testu).
- **Stav testu:** Zda byl test úspěšný (PASSED), neúspěšný (FAILED), nebo zda je potřeba provést další akce.

Příklad testovacího případu:

- **ID:** TC001
- **Název:** Test funkčnosti přihlášení uživatele
- **Předpoklady:** Uživatelský účet je zaregistrován.
- **Kroky:**
 - Otevřete přihlašovací stránku.
 - Zadejte správné uživatelské jméno a heslo.
 - Klikněte na tlačítko „Přihlásit“.
- **Vstupy:**
 - Uživatelské jméno: testuser
 - Heslo: Test1234
- **Očekávaný výsledek:** Uživatel je úspěšně přihlášen a přesměrován na domovskou stránku.
- **Skutečný výsledek:** (Doplní se po testování)
- **Stav testu:** (Passed/Failed)

Test - TC001

+ Add @ Apps

Description

ID: TC001

Název: Test funkčnosti přihlášení uživatele

Předpoklady: Uživatelský účet je zaregistrován.

Linked issues

is tested by

SCRUM-34 Test test case

JS TO DO

QAly Plus - Test Management

Add Test Case

Detail view

Expand All Test Cases

SCRUM-34 Test test case

LAST EXECUTION · No previous executions

Show execution history Execute

TEST STEP	TEST DATA	EXPECTED RESULT	TEST ATTACHMENTS
1 Otevřete přihlašovací stránku			
2 Zadejte správné uživatelské jméno a heslo.	Uživatelské jméno: testuser Heslo: Test1234		
3 Klikněte na tlačítko „Přihlásit“.		Uživatel je úspěšně přihlášen a přesměrován na domovskou stránku	
			or use drag and drop, or click on input and output

Testovací případy jsou klíčové pro systematické a opakovatelné testování, aby bylo zajištěno, že software splňuje požadavky a funguje bez chyb.

Defekt/chyba (bug)

Defekt (Chyba, Bug) je nežádoucí chyba nebo nesoulad v softwaru, který způsobuje, že systém nefunguje tak, jak by měl, nebo že neplní očekávané požadavky. Defekt může nastat v jakékoli fázi vývoje softwaru, od návrhu, kódování až po integraci a nasazení.

Typické příčiny defektů:

- **Chyby v kódu:** Nesprávně napsaný kód, který způsobuje neočekávané chování systému.
- **Chyby v logice:** Špatná logika v algoritmech nebo při zpracování dat.
- **Špatně definované požadavky:** Pokud jsou požadavky nejasné nebo nesprávně interpretovány, může to vést k vytvoření nesprávné funkcionality.
- **Chyby při integraci:** Když jednotlivé komponenty nebo moduly nejsou správně integrovány.
- **Nefunkční uživatelské rozhraní:** Pokud je uživatelské rozhraní navrženo tak, že vede k neočekávanému chování nebo je pro uživatele zmatené.
- **Zátěžové problémy:** Při vysoké zátěži se systém chová jinak než při běžném provozu, což může způsobit chyby.

Klasifikace defektů podle závažnosti:

- **Kritická chyba (Critical):** Chyba, která způsobuje úplné selhání systému nebo jeho hlavních funkcí, například zhroucení aplikace.
- **Závažná chyba (Major):** Chyba, která zásadně ovlivňuje funkčnost, ale systém stále může částečně fungovat.
- **Méně závažná chyba (Minor):** Chyba, která způsobuje nesprávnou funkčnost, ale nemá zásadní dopad na použití systému.
- **Kosmetická chyba (Cosmetic):** Chyba, která neovlivňuje funkčnost systému, ale ovlivňuje vzhled nebo uživatelský zážitek (např. špatné zarovnání textu).

Životní cyklus chyby (Defect Life Cycle):

- **Nahlášení chyby:** Tester nebo uživatel nahlásí chybu do systému pro sledování chyb (např. JIRA, Bugzilla).
- **Nová (New):** Chyba je zaevidována a čeká na přidělení.
- **Přidělená (Assigned):** Chyba je přiřazena vývojáři nebo týmu k řešení.
- **Oprava (Fixed):** Chyba byla vývojářem opravena.
- **Retest:** Tester ověřuje, zda byla chyba úspěšně opravena.
- **Uzavření (Closed):** Pokud je chyba opravena, je uzavřena. Pokud ne, proces se opakuje.
- **Odmítnutá (Rejected):** Pokud vývojář nebo manažer kvality zjistí, že nahlášená chyba není platná, může být odmítnuta.

Příklad:

- **Chyba:** Při kliknutí na tlačítko „Odeslat“ na formuláři se aplikace neočekávaně zavře.
- **Očekávané chování:** Formulář by měl být odeslán bez zavření aplikace.

Chyby (defekty) jsou nedílnou součástí softwarového vývoje a testování je klíčovým procesem pro jejich detekci a opravu, aby finální software splňoval kvalitu a fungoval správně.

Debug vs Test

Debug a Test jsou dva rozdílné procesy v rámci vývoje a zajištění kvality softwaru, ale oba hrají klíčovou roli v identifikaci a opravě chyb. Níže jsou hlavní rozdíly mezi těmito procesy:

1. Definice:

- **Debug (Ladění):** Proces identifikace, diagnostiky a opravování chyb (defektů) v kódu. Debugging probíhá poté, co je chyba objevena, obvykle buď vývojářem, testerem nebo uživatelem. Vývojář používá ladicí nástroje (debuggery), aby zjistil, proč se chyba stala, jaké bylo její příčiny a jak ji opravit.
- **Testování:** Proces, který má za cíl identifikovat chyby a zajistit, že software funguje podle požadavků a očekávání. Testování se zaměřuje na odhalování problémů tím, že spouští předem definované testovací scénáře a ověřuje, zda systém splňuje své funkční a nefunkční požadavky.

2. Účel:

- **Debug:** Opravit konkrétní chybu nebo problém v kódu, který již byl identifikován. Debugging je reaktivní proces – reaguje na zjištěnou chybu.
- **Test:** Ověřit, zda systém funguje správně a zda splňuje specifikace. Testování je preventivní proces – je zaměřeno na nalezení chyb před tím, než se software dostane k uživatelům.

3. Kdo jej provádí:

- **Debug:** Ladění obvykle provádí vývojáři, protože vyžaduje hluboké porozumění kódu a nástroje pro ladění.
- **Test:** Testování provádějí jak vývojáři (jednotkové testy, integrační testy), tak specializovaní testéři, kteří používají manuální nebo automatizované nástroje pro ověřování softwaru.

4. Kdy se používá:

- **Debug:** Používá se po nalezení chyby během testování nebo běhu aplikace. Je to proces, který se zaměřuje na opravu již identifikovaných problémů.
- **Test:** Používá se během vývojového cyklu k prevenci a detekci chyb. Testování probíhá pravidelně, během vývoje, před nasazením a někdy i po nasazení softwaru.

5. Nástroje:

- **Debug:** Vývojáři používají ladicí nástroje (např. GDB, Xdebug, nebo ladicí prostředky vestavěné do IDE jako Visual Studio nebo IntelliJ IDEA), které umožňují krokování kódu, sledování proměnných, nastavení breakpointů a analýzu běhu programu.
- **Test:** Testéři nebo automatizované systémy používají různé testovací nástroje (např. Selenium, JUnit, TestNG, Postman pro API testování), které pomáhají spouštět testovací scénáře a ověřovat výsledky.

6. Výsledek:

- **Debug:** Cílem je nalézt konkrétní chybu v kódu a opravit ji, aby aplikace fungovala správně.
- **Test:** Cílem je zjistit, zda aplikace splňuje požadavky a zda nejsou v systému další chyby. Testování poskytuje zpětnou vazbu o kvalitě softwaru.

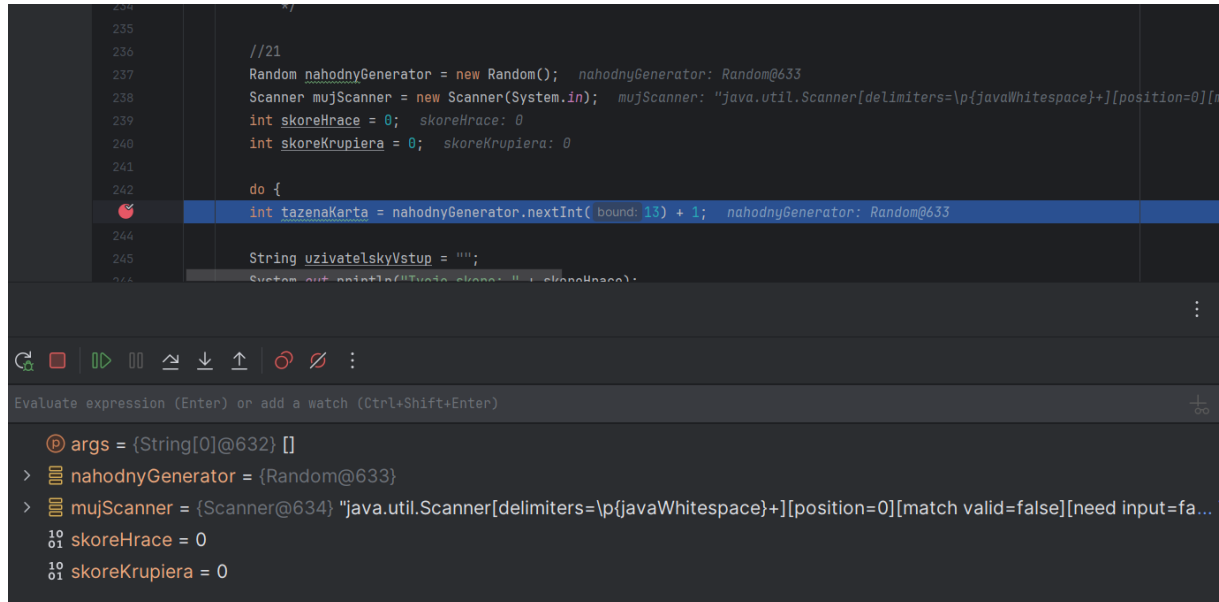
Příklady:

- **Debug:** Vývojář zjistí, že funkce pro přihlášení uživatele vyvolává chybu. Pomocí ladicího nástroje zjistí, že chyba je způsobena nesprávným zpracováním uživatelského jména, a opraví ji.
- **Test:** Tester provádí testovací scénář, ve kterém kontroluje funkčnost přihlášení uživatele s různými kombinacemi uživatelských jmen a hesel. Během testování zjistí, že systém nesprávně ověřuje určité kombinace, a nahlásí chybu vývojářům.

Shrnutí:

- Debugging je specifický proces zaměřený na analýzu a opravu již objevených chyb.
- Testování je širší proces, jehož cílem je najít chyby a zajistit, aby software fungoval správně podle požadavků.

Oba procesy jsou nezbytné pro dosažení kvalitního softwaru, ale testování je více zaměřeno na prevenci chyb a jejich včasnou detekci, zatímco debugování je o jejich konkrétní opravě.



The screenshot shows an IDE with a Java code editor and a debugger window. The code editor displays the following Java code:

```
//21
Random nahodnyGenerator = new Random(); nahodnyGenerator: Random@633
Scanner mujScanner = new Scanner(System.in); mujScanner: "java.util.Scanner[delimiters=\p{javaWhitespace}+][position=0][t
int skoreHrace = 0; skoreHrace: 0
int skoreKrupiera = 0; skoreKrupiera: 0

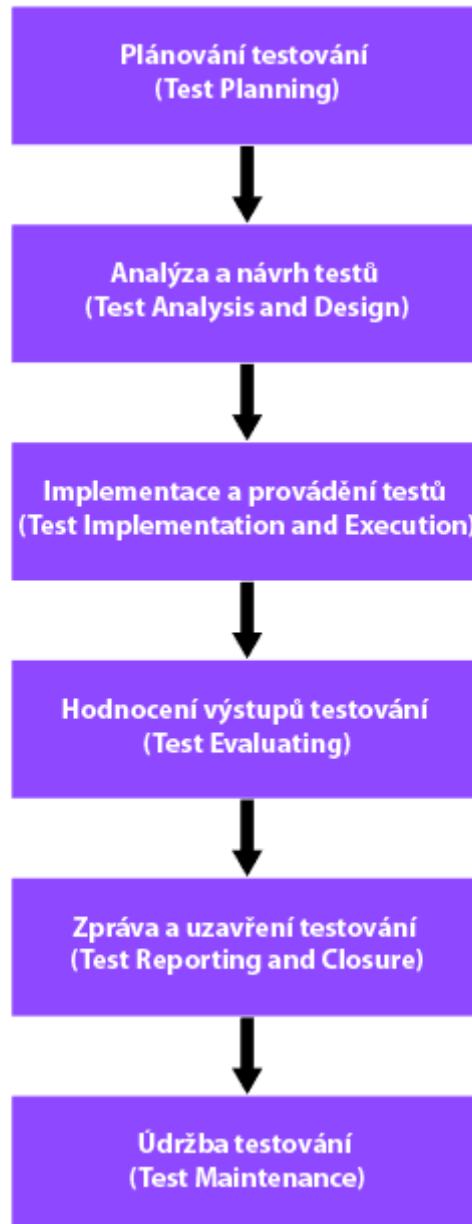
do {
    int tazenaKarta = nahodnyGenerator.nextInt( bound: 13) + 1; nahodnyGenerator: Random@633
    String uzivatelskyVstup = "";
    System.out.println("Tazena karta: " + skoreHrace);
```

The debugger window shows the following variables and their values:

- args = {String[0]@632 []}
- nahodnyGenerator = {Random@633}
- mujScanner = {Scanner@634} "java.util.Scanner[delimiters=\p{javaWhitespace}+][position=0][match valid=false][need input=fa...
- skoreHrace = 0
- skoreKrupiera = 0

Proces testování

Proces testování softwaru je systematický přístup k ověřování a validaci, že software splňuje stanovené požadavky a funguje správně. Tento proces obvykle zahrnuje několik fází, které mohou zahrnovat:



1. Plánování testování (Test Planning)

- **Cíl:** Stanovit strategii testování, definovat cíle a rozsah testování, identifikovat zdroje a harmonogram.
- **Aktivita:** Vytvoření testovacího plánu, určení typů testů (funkční, nefunkční, regresní, atd.), stanovení kritérií pro vstup a výstup, rozdělení rolí a odpovědností.

2. Analýza a návrh testů (Test Analysis and Design)

- **Cíl:** Analyzovat požadavky a navrhnout konkrétní testy, které ověří, zda software splňuje očekávání.

- **Aktivita:** Identifikace testovacích podmínek, vytváření testovacích scénářů a testovacích případů, definování vstupních dat a očekávaných výsledků.

3. Implementace a provádění testů (Test Implementation and Execution)

- **Cíl:** Příprava testovacího prostředí a provádění testů podle navržených testovacích případů.
- **Aktivita:** Nastavení testovacího prostředí, provádění testů (manuálně nebo automatizovaně), zaznamenávání výsledků a dokumentace chyb.

4. Hodnocení výstupů testování (Test Evaluating)

- **Cíl:** Vyhodnotit výsledky testování a zjistit, zda software splňuje požadované standardy kvality.
- **Aktivita:** Analýza zjištěných chyb, sledování pokroku a úspěšnosti testování, porovnání skutečných výsledků s očekávanými výsledky.

5. Zpráva a uzavření testování (Test Reporting and Closure)

- **Cíl:** Zpráva o výsledcích testování a uzavření testovacího cyklu.
- **Aktivita:** Vytvoření testovacích zpráv, dokumentace nalezených chyb a doporučení pro zlepšení, analýza úspěšnosti testování a ukončení testovacích aktivit.

6. Údržba testování (Test Maintenance)

- **Cíl:** Udržovat a aktualizovat testy v souladu s vývojem softwaru a změnami požadavků.
- **Aktivita:** Aktualizace testovacích případů, přidávání nových testů pro nové funkce, monitorování a reportování chyb v nových verzích.

Klíčové faktory úspěchu testování:

- **Jasně požadavky:** Bez dobře definovaných požadavků může být obtížné navrhnout efektivní testy.
- **Dostatečná dokumentace:** Dokumentace je klíčová pro sledování pokroku a výsledků testování.
- **Spolupráce týmu:** Úzká spolupráce mezi vývojáři, testery a dalšími zainteresovanými stranami zajišťuje, že testování je efektivní a cílené.

Testování je zásadní pro zajištění kvality softwaru a pomáhá odhalit a opravit chyby před tím, než se produkt dostane k uživatelům.

Plán testování, zpráva o testování

Plán testování (Test Plan)

Plán testování je dokument, který definuje strategii a rozsah testování softwaru. Obsahuje informace potřebné pro organizaci a provedení testování, včetně cílů, metod, zdrojů a harmonogramu. Zde jsou klíčové komponenty plánu testování:

1. Úvod

- **Cíl:** Krátké shrnutí testovacího projektu a jeho účelu.
- **Obsah:** Popis aplikace, její funkcionality a důvod testování.

2. Cíle testování

- **Cíl:** Stanovení konkrétních cílů, které má testování dosáhnout.
- **Obsah:** Například ověření funkčnosti, výkonu, bezpečnosti apod.

3. Rozsah testování

- **Cíl:** Určit, co bude a co nebude testováno.
- **Obsah:** Specifikace testovaných funkcí a oblastí, které se nebudou testovat (např. určité moduly, platformy apod.).

4. Testovací strategie

- **Cíl:** Stanovení metod, které budou použity pro testování.
- **Obsah:** Popis typů testů (funkční, regresní, integrační, atd.), které se použijí, a zda budou prováděny manuálně nebo automatizovaně.

5. Zdroje a odpovědnosti

- **Cíl:** Identifikace týmu a rolí v testovacím procesu.
- **Obsah:** Kdo bude odpovědný za které části testování (testery, vývojáře, manažery).

6. Harmonogram

- **Cíl:** Stanovení časového rámce pro testovací aktivity.
- **Obsah:** Klíčové milníky, jako je začátek a konec testování, termíny pro jednotlivé fáze.

7. Testovací prostředí

- **Cíl:** Definování prostředí, ve kterém bude testování probíhat.
- **Obsah:** Specifikace hardwaru, softwaru a konfigurace potřebných pro testování.

8. Kritéria pro úspěch a selhání

- **Cíl:** Určit, co se považuje za úspěšné dokončení testování.
- **Obsah:** Definice metrik a ukazatelů kvality.

Zpráva o testování (Test Report)

Zpráva o testování je dokument, který shrnuje výsledky testování a poskytuje přehled o kvalitě softwaru. Obsahuje informace o nalezených chybách, testovaných funkcích a celkovém výkonu. Zde jsou klíčové komponenty zprávy o testování:

1. Úvod

- Cíl: Krátké shrnutí zprávy a cílů testování.
- Obsah: Popis testovaného softwaru a důvod provedení testování.

2. Testovací prostředí

- Cíl: Informace o testovacím prostředí.
- Obsah: Specifikace hardwaru, softwaru a konfigurace, které byly použity během testování.

3. Testovací aktivity

- Cíl: Přehled o provedených testech.
- Obsah: Seznam testovacích případů, které byly vykonány, a stručný popis testovaných funkcí.

4. Výsledky testování

- Cíl: Shrnutí výsledků testů.
- Obsah: Počet úspěšných a neúspěšných testů, přehled nalezených chyb a jejich závažnost.

5. Zjištěné defekty

- Cíl: Detailní popis nalezených chyb.
- Obsah: ID chyby, popis, závažnost, status (otevřeno, opraveno, uzavřeno) a příslušné testovací případy.

6. Doporučení

- Cíl: Poskytování návrhů na zlepšení.
- Obsah: Doporučení pro opravy chyb a případné úpravy testovacího procesu.

7. Závěr

- Cíl: Shrnutí výsledků a celkového hodnocení kvality softwaru.
- Obsah: Celkové hodnocení, zda software splňuje stanovená kritéria a je připraven k nasazení.

Test Execution Report

Export [icon] [icon]

Description:
A report that summarizes tests that were executed in the selected period.

Filters:

Project
[icon] [redacted] (N) [icon]

Test Cycle
Select Test Cycle [icon]

User
All users [icon]

Status
All statuses x [icon]

Date range
01/12/2023 → 24/10/2024 [icon] Refresh report [icon]

Test Case	Test Cycle	Created On	Execution Assignee	Test Result	Actions
[icon] FIN-79 Create test for - API - PLOT	None	05/Sep/2024	[icon] Patrik [redacted]	UNEXECUTED	
[icon] FIN-93 Create test for OMDb API - Alohomora test case Identifikujte datum vzniku filmu	None	05/Sep/2024	Unassigned	UNEXECUTED	
[icon] FIN-94 Create test for OMDb API - Alohomora test case Identifikujte délku filmu	None	05/Sep/2024	[icon] Tomáš	PASSED	
[icon] FIN-89 Create test for OMDb API - Alohomora test case Identifikujte zapletku filmu 2	None	05/Sep/2024	[icon] Matěj	PASSED	
[icon] FIN-93 Create test for OMDb API - Alohomora test case Identifikujte datum vzniku filmu	None	05/Sep/2024	[icon] Tomáš	PASSED	
[icon] FIN-89 Create test for OMDb API - Alohomora test case Identifikujte zapletku filmu 2	None	05/Sep/2024	[icon] Matěj	PASSED	
[icon] FIN-88 Create test for OMDb API - Alohomora test case Identifikujte zapletku filmu 1	None	05/Sep/2024	[icon] Matěj	PASSED	
[icon] FIN-86 Create test for OMDb API - Alohomora test case - overenie ratingov	None	05/Sep/2024	[icon] Tomáš	PASSED	
[icon] FIN-86 Create test for OMDb API - Alohomora test case - overenie ratingov	None	05/Sep/2024	Unassigned	UNEXECUTED	
[icon] FIN-86 Create test for OMDb API - Alohomora test case - overenie ratingov	None	05/Sep/2024	[icon] Tomáš	PASSED	

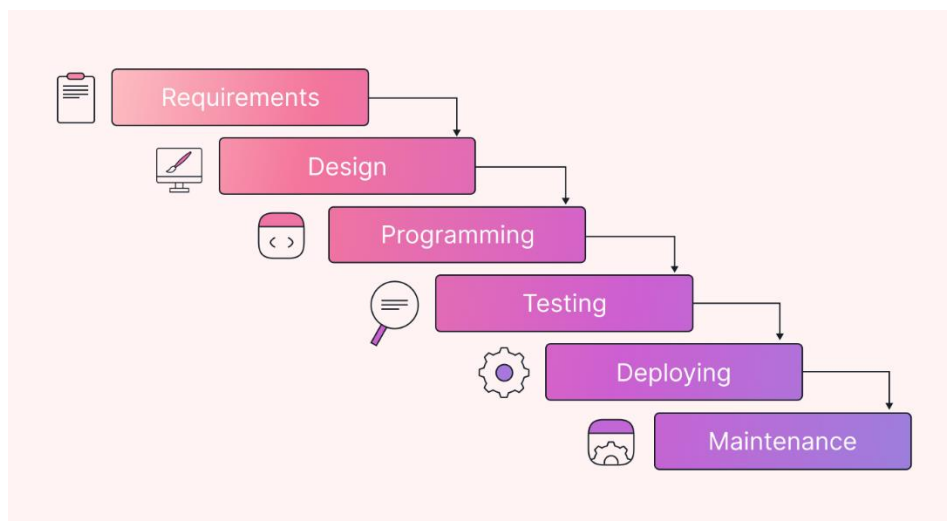
Tyto dokumenty jsou klíčové pro úspěšný proces testování, protože poskytují strukturu a jasnost v testovacích aktivitách a výsledcích. Pomáhají také udržovat komunikaci mezi testovacím týmem, vývojáři a managementem.

Modely vývoje softwaru

Modely vývoje softwaru jsou různé přístupy k plánování, organizaci a řízení vývojového procesu softwaru. Tyto modely poskytují strukturu a postupy, které určují, jakým způsobem bude software vytvářen, testován, nasazován a udržován

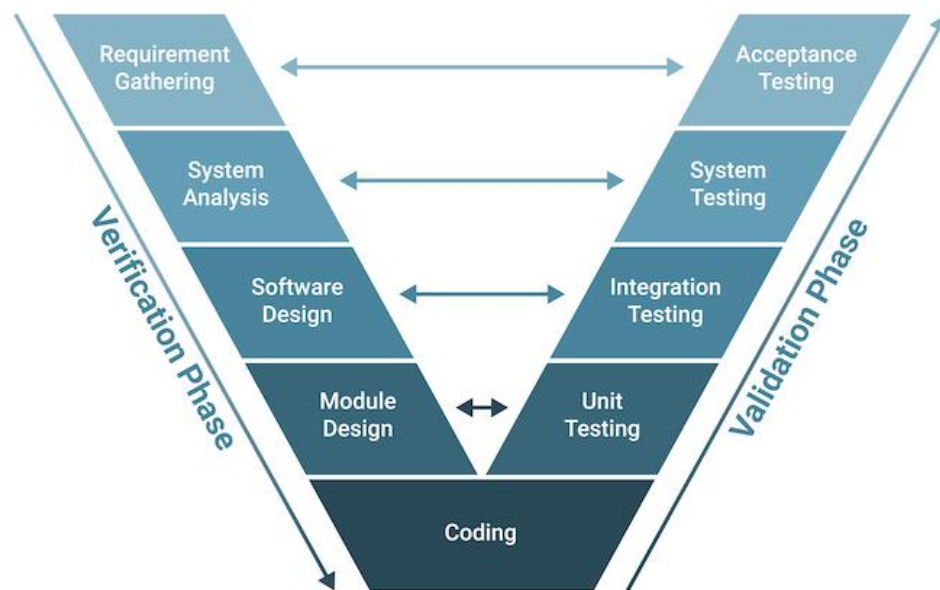
Vodopádový (sekvenční) model

- Charakteristika: Jedná se o lineární, sekvenční model, kde každá fáze musí být dokončena před tím, než se přejde do další. Mezi hlavní fáze patří sběr požadavků, návrh, implementace, testování, nasazení a údržba.
- Výhody: Jasná struktura, snadná správa, jednoduché sledování pokroku.
- Nevýhody: Těžko reaguje na změny během vývoje, problémy objevené pozdě mohou být drahé na opravu.
- Vhodné pro: Projekty s jasně definovanými požadavky, které se nebudou měnit.



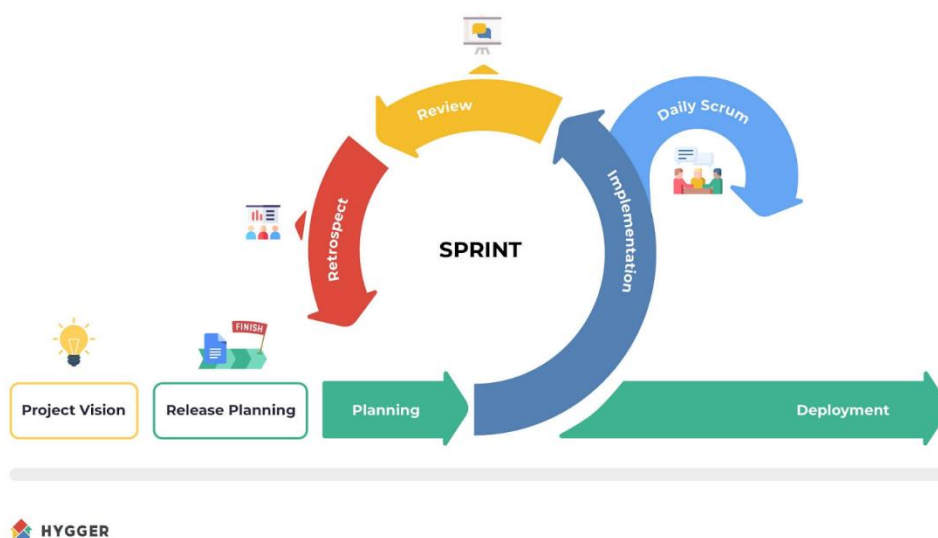
V Model

- Charakteristika: Rozšíření vodopádového modelu, kde každá fáze vývoje má odpovídající fázi testování. Testování začíná už od fáze sběru požadavků.
- Výhody: Přísné testování během každé fáze zajišťuje včasnou detekci chyb.
- Nevýhody: Stejně jako vodopádový model je obtížné reagovat na změny během vývoje.
- Vhodné pro: Projekty s pevnými požadavky, kde je kladen důraz na kvalitu.



Scrum/Agilní vývoj

- Charakteristika: Scrum je specifický agilní rámec, který se zaměřuje na krátké, časově ohraničené sprinty (obvykle 2-4 týdny) s jasně definovanými cíli. Tým pracuje na dodání funkčního softwaru po každém sprintu.
- Výhody: Pravidelné dodávky, vysoká flexibilita, neustálé zlepšování produktu.
- Nevýhody: Vyžaduje vysokou disciplínu a pravidelnou komunikaci.
- Vhodné pro: Týmy, které potřebují rychle reagovat na změny a poskytovat pravidelné aktualizace.



Kariéra testera softwaru

Vývojář (Developer)

- **Popis:** Vývojář je zodpovědný za návrh, implementaci a údržbu softwarových aplikací. Pracuje s programovacími jazyky, frameworky a nástroji pro vývoj a testování kódu. Může se specializovat na frontend (uživatelské rozhraní), backend (serverová logika) nebo full-stack (kombinace obojího).
- **Hlavní úkoly:** Psát a testovat kód, analyzovat požadavky, spolupracovat s ostatními členy týmu, udržovat a vylepšovat existující aplikace.

Softwarový architekt (Software Architect)

- **Popis:** Softwarový architekt navrhuje a řídí strukturu a design softwarových systémů. Zodpovídá za rozhodování o technologiích, standardech a postupech, které se používají při vývoji softwaru. Je spojovacím článkem mezi technickým týmem a vedením.
- **Hlavní úkoly:** Vytváření architektonických diagramů, hodnocení technických rozhodnutí, návrh řešení pro komplexní problémy a zajištění souladu s obchodními cíli.

DevOps

- **Popis:** DevOps inženýr se zaměřuje na zlepšení spolupráce mezi vývojovým a operačním týmem, aby se urychlil cyklus vývoje a nasazení softwaru. Používá automatizaci a monitoring, aby zajistil, že software funguje spolehlivě a efektivně.
- **Hlavní úkoly:** Automatizace nasazení a správy infrastruktury, monitorování výkonu aplikací, zajištění bezpečnosti a dodržování standardů.

Scrum Master

- **Popis:** Scrum Master je facilitátor pro týmy, které používají Scrum metodologii. Pomáhá týmu dodržovat Scrum procesy a praktiky, odstraňuje překážky a zajišťuje, aby tým měl všechny potřebné zdroje k úspěšnému dodání sprintů.
- **Hlavní úkoly:** Organizování a vedení denních stand-up meetingů, zajištění správné implementace Scrum praktik, komunikace s externími zainteresovanými stranami.

Produktový manažer (Product Manager)

- **Popis:** Produktový manažer se zaměřuje na plánování, vývoj a marketing softwarového produktu. Řídí životní cyklus produktu a zajišťuje, aby splňoval potřeby trhu a zákazníků.
- **Hlavní úkoly:** Vytváření a udržování roadmapy produktu, shromažďování zpětné vazby od zákazníků, koordinace mezi týmy.

UX/UI Designer

- **Popis:** UX/UI designer se zaměřuje na uživatelskou zkušenost (UX) a uživatelské rozhraní (UI) aplikací. Navrhuje vizuální aspekty a interakce, aby byl produkt uživatelsky přívětivý a esteticky příjemný.
- **Hlavní úkoly:** Vytváření prototypů, design uživatelských rozhraní, provádění uživatelského testování a shromažďování zpětné vazby.

Tyto pozice jsou klíčové pro úspěšné fungování IT firmy a přispívají k dosažení jejích cílů a kvalitě vyvíjeného softwaru.