Vědomosti

**Automatizace**

**Výhody:**

Používám nástroj – IntellIdeja k usnadnění práce či jiný

Šetří čas,Snižuje pravděpodobnost chyby při vykonávání testu

Umožňuje "nekonečně" mnoho spouštění testů (24/7)

Pozitivní vliv na regresní testování,dostáváme zpětnou vazbu na testovaný produkt často a rychle

**Nevýhody**

Je náročná na technické znalosti testerů, Správa/údržba zabírá mnoho času

Tester ztrácí spojení s testovaným systémem,Testy časem odhalují čím dál méně chyb,"Není" to testování. Riziko pesticidního paradoxu

Využít pro regresní testy, ušetřený čas využit k exploraci, zaměřit se na happy path testy a využít jejich potenciál

Dále je vhodná pro smoke testy, performance testy, unit testy, integrační testy

U automatizace je důležitá orientace v kódu, v dokumentaci vše nemusí být zanešeno

Browser= slouží k navigování v aplikaci

Asserter= slouží k ověřování

Automatizace je zpětná vazba na testovaný produkt, UI se nevyplatí automatizovat

Automatizovat tzv. happy path cases – testy, které by měly projít, základní funkcionality (například jdi na stránku seznam a zjisti, zda je tam tlačítko přihlásit)

Vím ,co tam bude. Důležité je starat o automatizované testy, náročné na čas,

Automatizované testy se nehodí u negativních testů, exploratory testingu

70% manuální testy, 30% regresní testy

Nevyplatí se to u náročných testů, E2Etestů-Vitalii, ale používá, u testů, které se dělají jen 2x

**Lowe -Code automation**

Zjednodušené programování

Low code = připravená funkcionalita, kterou můžeme používat

Dostaneme sadu kroků

Způsob testování aplikace, rychle pomocí naplánování,

generování a spouštění testů

• Tester nepotřebuje rozsáhlé znalosti programování

• Nutno mít k dispozici takzvaný "framework"

• Testujeme na UI vrstvě

Proč sada testu neprošla, kouknu na failed screen shot, systém podtrhne, co je špatně

**Výhody**

Jednoduché na naučení a použití, zvyšuje produktivitu, oproti klasické automatizaci, zvyšuje rychlost reakce na změny

**Nevýhody**

Pro komplexnější testovací scénáře může být pomalejší

Pro větší změny je zapotřebí zkušenějšího vývojáře

Používáme předpřipravené knihovní funkce /framework – sada předpřipravených testů

**Framework =** nástroj, sada knihovních funkcí, vytvořena od někoho, je nám k dispozici při testování – jednoduché na naučení a použití, set kódů

Full code automatization, Lowe code automatization, No code automatization - Selenium

Krokování (debugging) testu – zastavení testu na zabuggovaném kroku, proces, kdy se to na breakpointech zastavuje

Co je breakpoint (pauza)– kde se zastaví test, nepokračuje dál, pokad to funguje, manuálně pustím dál

Pro automatizaci je nejspolehlivější Mozilla Firefox

Druhy asertací – zda na té stránce něco je ANO/NE, co mohu očekávat (URL adresu, kontakt)

* Zda jsou základní elemety na stránce (tlačítko, přihlásit se)
* Co je možné otestovat , ověřit

Manuální verzus Automatizované testování

Určení mezi manuálním a automatizovaným testováním je důležité. Manuální testování se provádí osobně, klikáním skrz aplikaci nebo interakcí se softwarem a API pomocí odpovídajících nástrojů. Je to nákladné, protože vyžaduje, aby někdo nastavil prostředí a provedl testy sám, a může být náchylné k lidským chybám, protože tester může udělat překlepy nebo vynechat kroky ve skriptu testu.

Naopak automatizované testy provádí stroj, který spouští testovací skript napsaný předem. Tyto testy mohou mít různou složitost, od ověření jednoho způsobu v třídě po zajištění, že provádění sekvence složitých akcí v uživatelském rozhraní vede ke stejným výsledkům. **Je to mnohem robustnější a spolehlivější než manuální testy** – ale kvalita vašich automatizovaných testů závisí na tom, jak dobře byly napsány testovací skripty.

**Assertace**

= tvrzení, v našem kontextu vyhodnocení, neboli očekávané chování

Naše automatizované testy umí zatím jen klikat, vyplňovat a navigovat

Druhy assertaci:

• Obecně můžeme testovat kroky, které nám vyhodnotí např:

• Přítomnost elementu na stránce/Nepřítomnost elementu na stránce

• Konkrétní **hodnotu** v elementu – tlačítko na stránce, cancel ano či ne, drop down

• Konkrétní text zobrazený v daném elementu-

• Že je/není možné pole editovat aj.

• Záleží jen na tom, co nám vývojaři přidají do frameworku

Selenium je framework, který ovládá prohlížec, má napojení na několik programovacích jazyků (používáme Javu)Java ovládá selenium- selenium prohlížeč. Selenium pracuje s UI jen, JAVA dělá zbytek, selenium běží na pozadí automatizace

**Software**- je nehmotná část počítače, která řídí fyzickou část počítače (hardwere) a umožňuje jim provádět různé úkoly. Aplikace na základě potřeby zákazníka, která řeší jeho problém. Pomocí softweru řeším potřebu.

Kvalita je spokojenost a štestí každého, ktěří jsou zapojení . Kvalita se netýká jen technické specifikace, ale i očekávaného chování, které má produkt nebo služba a jeji vliv na zákazníka. Kvalita uspokojení potřeba zákazníka.

**Stakehodler** = zůčastněná strana, která je zapojena do projektu – má zájem, motivaci, odpovědnost, očekávání

**UX (User Experience)** je termín používaný v oblasti designu, vývoje produktů a uživatelského rozhraní k popisu celkového dojmu a interakce, které uživatelé mají s produktem, službou nebo systémem. UX se zaměřuje na zlepšení kvality uživatelského prostředí tak, aby uživatelé měli pozitivní, efektivní a uspokojivý zážitek při používání produktu.

**Klíčové prvky:**Uživatelský výzkum, informační architektura, UI – uživatelské rozhraní/design, Usability – použitelnost, testování a iterace a celkový dojem

**Wireframe** je jednoduchý, často černobílý nákres nebo diagram, který reprezentuje strukturu a rozložení prvků na webové stránce, mobilní aplikaci nebo jiném digitálním rozhraní. Wireframe je vizuální náčrt, který slouží k rychlému a efektivnímu zobrazení základních prvků a uspořádání obsahu na digitálním rozhraní, aniž by se zaměřoval na grafický design, barvy nebo detaily.

Hlavní účely: struktura a uspořádání,navigace, obsah,funkčnost

**A/B testing**

2 verze aplikace, cílem je zjistit, která z verzí má lepší výkonnost,dojem na zákazníka

Hodnotí skupina uživatelu A verze barevný design, B černobílý design.,vybere se ta verze, která více vyhovuje

-který ze dvou systémů, komponent funguje lépe

Testování bezpečnosti a testování pod zátěží

**MVP – Minimal Viable Product**

Koncept vývoje softweru, který popisuje strategii vytvoření produktu nebo služby s minimem funkcí, které je potřebné, aby produkt byl použitelný a dostal zpětnou vazbu od zákazníka. Používá se k testování a ověření, zda produkt má potenciál uspět u zákazníka.

Počítačová síť = spojení 2 a více počítačů, aby mohly navzájem komunikovat a sdílet své prostředky – hardwerové nebo softwerové

* **síťový hardware** – umožňuje vlastní fyzické propojení
  + Síťová karta + přenosové médium + propojovací síťové prvky
* **síťový software** – stará se o přesuny dat, komunikaci, navazování spojení a další služby jako např. Zabezpečení

**Využití počítačové sítě**:

* Komunikace (Přenos textu, videa)
* Sdílení hardwarových zdrojů (Počítač nemusí mít vlastní tiskárnu, pomocí sítě může několik zařízení sdílet stejnou tiskárnu (např. ve firmě)
* Sdílení souborů, dat, informací (Přístup k informacím uloženým na jiném zařízení)
* Sdílení software -Aplikace na internetu

**Spojované** (analogová telefonní síť) – přepínání okruhů mezi komunikačními uzly, které je udržováno během celé komunikace a

**nespojované sítě** (internet) – není dána přesná cesta, data rozdělena do malých částí – pakety, které jsou odeslány do sítě. Na přijímací straně jsou z paketů data extrahována a opětovně složena do původní podoby.

Komunikační protokoly

* V síti se nachází více zařízení, které mohou odesílat nebo přijímat informace a je proto nutné stanovit, jak se budou domlouvat – proto vzniká komunikační protokol, který určuje co, jak a kdy ?
* Definuje formát a pořadí zpráv, které si komunikující strany vyměnují, stejně jako akce vykonané při odesílání nebo příjmu zpráv

IP,HTTP,FTP – komunikační protokol

Komunikace v počítačových sítích probíhá v několika fázích

* Otevření přenosu, samotný přenos dat,ukončení přenosu

Co je software? Programové vybavení, vše co není hardware (fyzické součásti počítače)

Dělíme jej na systémový software a aplikační software

**SS** - umožňuje efektivní používání počítace (součástí firmware – obšažený v hardwere (CD/DVD, zvuková karta a operační systém- spravuje počítač, vytváří prostředí pro programy)

**AS** – umožňuje uživateli vykonávat nějakou užitečnou činnost (grafické programy,vývojové nástroje, zábavní software)

Jak fungují sítě:

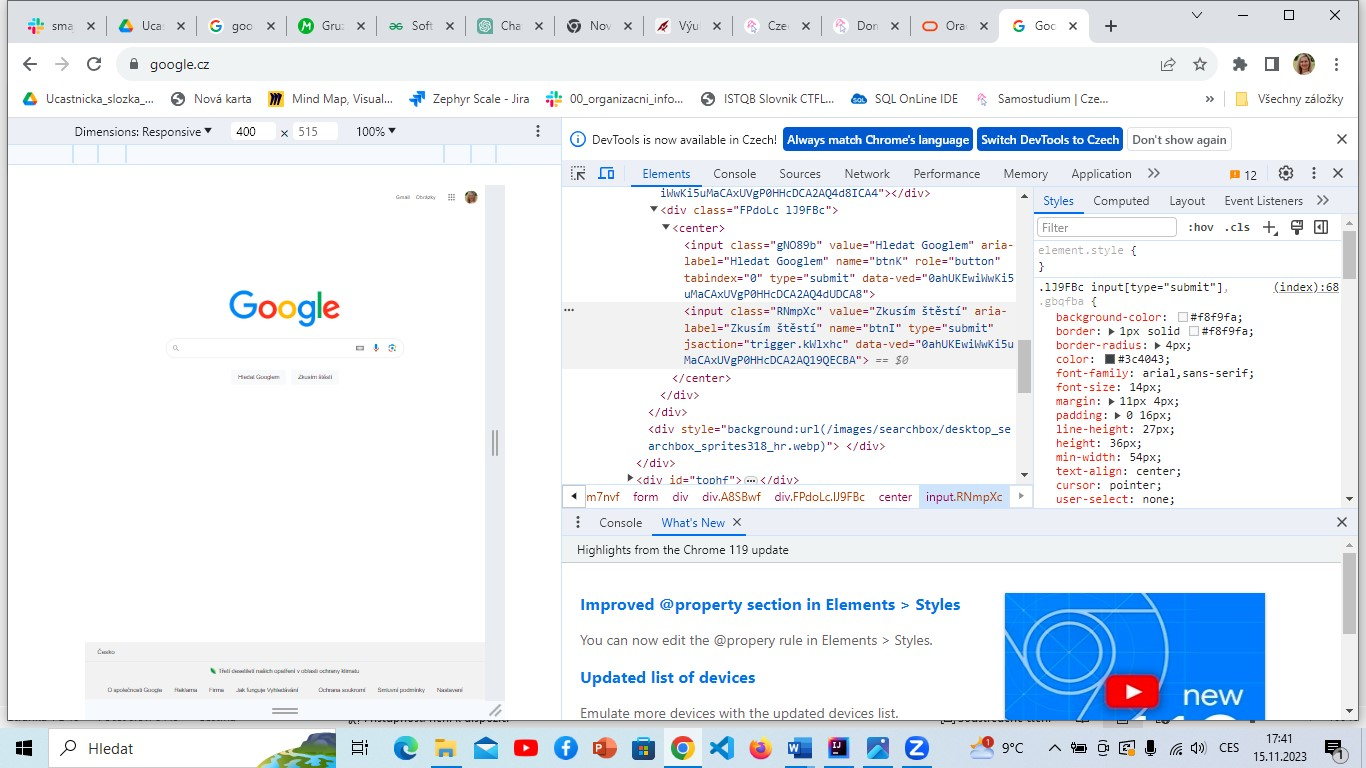
IP adresa, směrování (routing), DNS, Protokol

Síť má spoustu různých protokolů:

Wifi, ethernet, směrování mezi routery, HTTP, HTTPs

Cloud: server, počítač, který poskytuje nějakou službu, datová centra: spousta serverů pod jednou střechou

DOM = **Document Object Model,** stromová struktura, kterou si prohlížeč vytvoří pro zpracování stránky (zobrazíme pomocí DevTools – F12 > záložka Elements)



Vybírátko/selector

Vedle tlačítko – otevře nabídku, k ozkoušení webové stránky na různých zařízeních (miniatura mobilu), možnost výběru definice zařízení a rozlišení, místo myši klikátko jak na telefonu, musím se vykliknout zpět

* Umožňuje přistupovat k jednotlivým objektům HTML a pracovat s nimi

(Elementy, atributy, text)

* Díky stromové struktuře dovedeme rozlišit nadřazené, podřazené a rovnocenné elementy

Záložka:**elements** – hierarchie: browser vykresluje web pomoci selektoru, můžeme očnacovat jednotlivé části textu, loga aj.

U automatizace potřebuji znát DIV, class ..

Je možná validace na straně UI, změním v Dev tools názvy odstavců například,ale refreshnutím to zmizí. Uprávy může dělat jen správce webu, ale někdo může zneužit

Element má textový či checkboxový charakter. Součastí Elementu jsou atributy, kdy, každý hraje svoji roli.

Atributy jsou na Elementu id, class, style, tittle aj - oranožovo hnědé

Element a má atribut href, jeho hodnota označena na začátku a na konci tágem.

Ve Styles – kontroluji design – barvu a velikost písma (zabaleno v Divu- někdy nutno rozbalit)

Přes Styles -) filter najít

Klik na Elements v dev tools a pomocí CTRL plus F mohu vyhledat konkrétní slovo

Jinak změna pomoci Edit HTML, edit attribute, edit text

Mohu udělat screen shot přes Copy – CopyXpath(to je jiná metoda) – Capture no screenshot

**Panel Console:** informuje nás o chybách, textový výpis v chybě vypsán pomocí Javasriptu. Měl by tam být I prolink, kde tu chybu najdu. Najdu tam výpis textový o chybě, který můžu použít v bug reportu.

Warningy, informace o zastaralém kódu/části kódu, problém při načítání stránky

Pokud si chci posílat requesty- rychlé (fetch) – POST,GET

Panel **Network-** vidím veškerou komunikaci aplikace/webu se serverem

Poskytuje informace: verifikační – heslo

Kód, performance testy -časy načítání, kontrola rychlosti

Status cody v requestu – GET,POST atd

Obrázky, javascripty,STYLY- CSS

Co je v cachi, cookies

Record Network button:

Zaklik Disable check – načte se vše správně

Plus Preserve Log – aby I po přenatčení stránky zůstaly záznamy

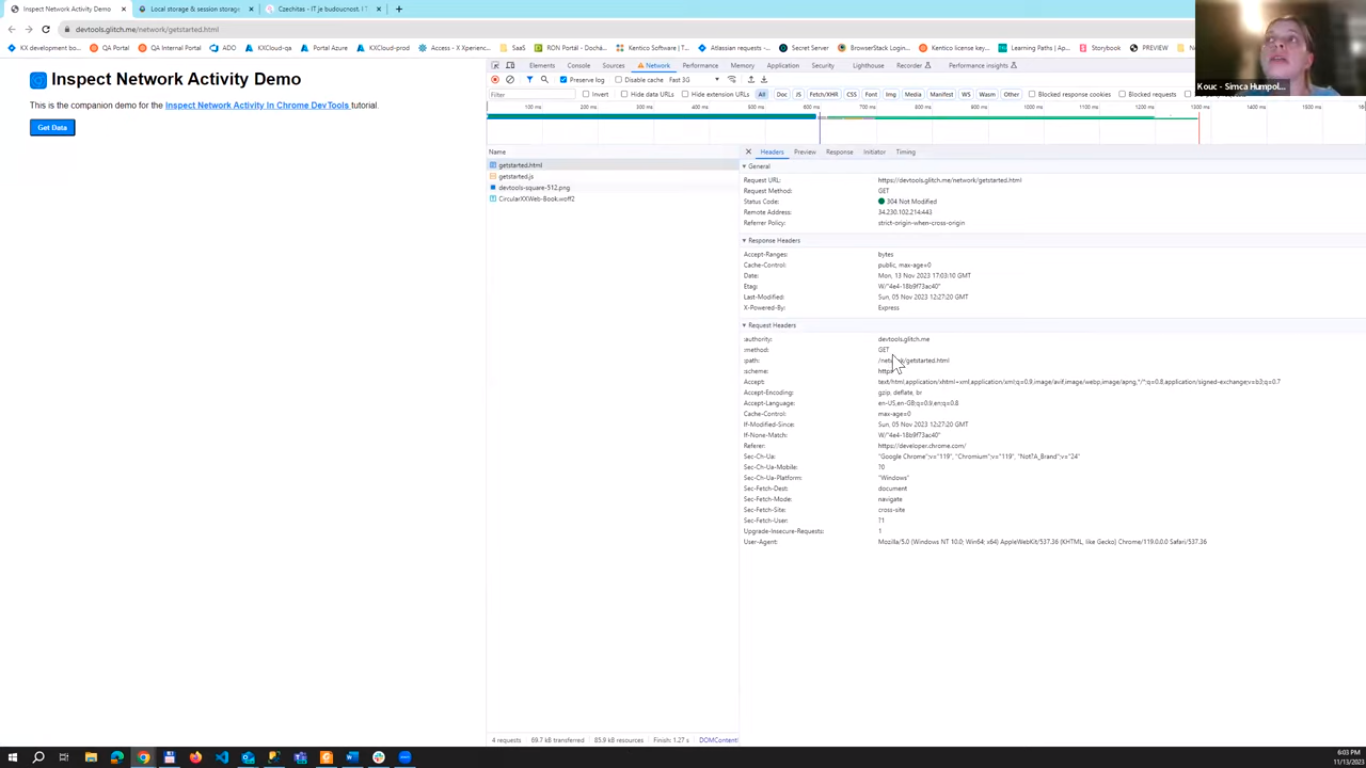
Pokud budu chtít vyčistit tak vymažu přes zaškrtlé kolečko

Mohu zkopírovat z Network metodu Copy as Fetch a klik CTRL plus L do Console a znovu udělat ten request a změnit ho v metodě(jeho tělo)

Jakoukoliv komunikaci v Network mohu zkopírovat do Console přes Copy as Fetch

Na Network Proklik šipka dolů- FAST3G mohu **nasimulovat** pomalejší připojení a zjistit , zda se mi stránka network načte a jakou rychlostí – že se I s pomalejším internetem se načte vše jak má a nespadne to – pokud spadne tak chyba

Vytváří falešné chování, jako bychom měli pomalejší síť



Podle určité hiearchie browser vykresluje web

Vyberu selektorem, konkrétní úsek nebo scroll in to view část ve vybraném úseku

Manipulace s elementy – validace na straně UI a validace vnitřní

Nměním v obsahu stránky,ale jako uživatel to mohu zneužit a poslat ve formuláři nevalidní hodnotu

Elementy hlavni tagovací značka Div,a,H1

Atributy jsou na elementu: id, class, style, tittle – oranžovo hnědá barva

**Záložka Applications** – stránka něco ukládá do storage

Local Storage – informace vydrží, kterou uložím – Dark mode obrazovky například – mohu samozřejmě I smazat, Session Storage – nevydrží informace, zda ukládá nějaké Cookies,pracuje s Cache Storage – v případě, že máme na webové aplikaci nainstalovanou cache- cache na úrovni aplikace, může tam ukládat informace

Přes panel záložka **Securit**y – v URL adrese zámeček, že je připojení, stránka zabepečená, zjistím certifikát a zda je připojení zabezpečené

Přes panel **Lighthouse** - analytický nástroj, jež umožní diagnostiku stránky v různých oblastech. Ohodnotí, zda splňuje nějaké standardy, upozorní na to, co je špatně. Vyhodnocení z pohledu Accesibility a SEO například

HTML= HyperText Markup Language – značkovací jazyk

Definuje obsah a strukturu stránky, pomocí HTML píšemé Kód, který říká, co na stránce bude a jakým způsobem to tam bude uložené

Značkami obalujeme text a dáváme textu význam

V programu Visual Studio Code zmáčku vykřičník a zobrazí se mi struktura HTML

**XML**- slouží k transferu dat

**HTML**- slouží k zobrazení dat

JavaScript – do HTML nebo webové stránky přidává nějaké chování – co se stane když se klikne na tlačítko, když uživatel vyplní špatně formulář

**Head** – záhlaví HTML stránky

Obsahuje metadata stránky, obsah není vidět, instrukce pro prohlížeč

Hlavička stránky, obsahuje vlastnosti stránky, parametry

Obsahuje elementy

<title>, <style>, <meta>, <link>, <script>, and <base>.

Nejdůležitější je párová značka tittle,název stránky ve výsledcích vyhledávání,záložka v oblíbených, definuje název v liště prohlížeče

Připojuje se k tomu CSS soubor přes Link

Id – používáme, když chceme vytvořit menu, unikátní označení, sekce kódu

<h1 id="myHeader">My Header</h1>

**id**

Identifikátor, který by měl být jednoznačný. Používá se pro CSS a skripty. V zápisu CSS se jménu identifikátoru předřazuje křížek #. Příklad:

**<style>  
#mrnavej {font-size: 5px}  
</style>  
...  
<p id="mrnavej">Odstavec děsně mrňavým písmem.</p>**

Id unikatní identifikator

## class

Třída elementu používaná pro CSS. Umožňuje jednoduché označení elementu, který má mít jiný vzhled. V zápisu stylu se jménu třídy předřazuje tečka. Příklad:

**<style>  
.cervene {color: red}  
</style>  
...  
<p class="cervene">Odstavec červeným písmem</p>**

Style pomáhá ke stylování nějakého většího množství elementů, každý atribut hraje nějakou roli

ID je jedinečný identifikátor prvku nebo skupiny prvků

Class se používá k označení několika podobných prvků, které mají buď společnou logiku nebo společný vzhled

<p id = ‘Martina’ class = ‘Smajl’/>  
<p id = ‘Alca’ class = ‘Smajl’/>

Například, máme tým nazvaný "Smajl" (Smile), každý z vás má své vlastní ID (jméno), ale můžeme pro vás použít stejnou třídu, protože jste ze stejného týmu:

Type input – o jaký input jde, textový nebo checkbox

Element input s atributem type a hodnota atributu type je checkbox

Elemet a odkaz, jedním z jeho atributů je atribut href – kam vede ten odkaz, hodnota atributu href je https// může mít ještě atribut target – blank nebo class, který má hodnotu icons

Celý řádek je element, označen na začátku a na konci tagem

CSS = Kaskádové styly, nastavuje vzhled stránky (velikost písma, barvy textu)

ERP – Enterprise Resource Planning

znamená plánování podnikových zdrojů, což je typ softwaru pro řízení podniku

 označuje sadu softwaru, který organizace používají ke správě každodenních obchodních činností, jako je [účetnictví](https://www.oracle.com/cz/erp/financials/), [zásobování](https://www.oracle.com/cz/erp/procurement/), [řízení projektů](https://www.oracle.com/cz/erp/project-portfolio-management-cloud/), [řízení rizik a dodržování předpisů](https://www.oracle.com/cz/erp/risk-management/) a [operace dodavatelského řetězce](https://www.oracle.com/cz/scm/).

**CRM – Customer RelationshipManagement**

tedy v překladu řízení vztahů se zákazníky. Tzv. CRM software je program, díky němuž můžeme přehledně shromažďovat veškeré potřebné údaje o zákaznících naší firmy - kontaktní informace, celkové obraty, podíly na obratech v určitém segmentu apod

**BI – Business Intelligence**

je sběr, analýza a interpretace vnitrofiremních dat, na základě kterých management dělá strategická rozhodnutí a tím ovlivňuje obchodní výkonnost firmy.

AI-Artificial Intelligence

- je schopnost strojů napodobovat lidské schopnosti, jako je uvažování, učení se, plánování nebo kreativita

**IoT – Internet of Thinkgs**

je v [informatice](https://cs.wikipedia.org/wiki/Informatika) označení pro [síť](https://cs.wikipedia.org/wiki/Po%C4%8D%C3%ADta%C4%8Dov%C3%A1_s%C3%AD%C5%A5) fyzických zařízení, [vozidel](https://cs.wikipedia.org/wiki/Internet_vozidel), domácích spotřebičů a dalších zařízení, která jsou vybavena elektronikou, softwarem, [senzory](https://cs.wikipedia.org/wiki/Senzory), pohyblivými částmi a síťovou konektivitou, která umožňuje těmto zařízením se propojit a vyměňovat si [data](https://cs.wikipedia.org/wiki/Data).

Infrastruktura –

SW na zakázku

Enterprise SW, Integrace, Digital Transformation…

**MVP- Minimal Viable Product**

minimální životaschopný produkt, je [produkt](https://cs.wikipedia.org/wiki/V%C3%BDrobek) s nejmenší možnou funkcionalitou, který je však plně použitelný a umožňuje rychle získat zpětnou vazbu od zákazníků pro další vývoj

**Quality Assurance**

Tetsing je součástí Quality Assurance, ale není QA. Kvalita musí být sledována, vybuilděna.

Testing, aktuální stav, není záruka kvality. Vyžaduje analýzu, dokumentaci.

Cílem QA – proces odsouhlasení

* Kontinuální zlepšování, transparentnost a četnost, kontinuální učení, fail fast
* Morální, konzistentní
* Znalost- business, technical,soft skills
* Správný způsob testování,správný způsob kódování

**TESTOVÁNÍ**

Tetsing je měřitelný. Podává informace a podporuje informované rozhodnutí.

Testing redukuje nejistoty a risky. Testing je profesionální skepticismus. Testing je vidět aplikaci očima zákazníka.

Testing není

Testování nerozhoduje, podporuje rozhodnutí.

Testování nezlepšuje kvalitu.

Testování není prevencí chyb.

Cíle testingu

* Odhalit bugy, zjistit v jakém stavu je aplikace, zajistit QA – nejvyší vhodnou kvalitu produktu
* Poskytnout informace, najít důležité bugy, vytvořit si důvěru v product
* Odlišnost mezi tím co víme a co potřebujeme, je to proč testujeme
* Dívat se na testování z jiného pohledu

Zdroj vědění: znalosti (zdroj pravdy, jak to má fungovat), práce s požadavky, systém under test, doménova znalost, komunikace s lidmi

**Požadavek** = popis potřeby

Pareto´s principle / Zbytečné varianty

Je to vysoká priorita?

Skutečně to někdo potřebuje?

Jak to přispívá k větší hodnotě?

Existuje jednodušší alternativa?

Vyhnout se zbytečným variacím.

**Priorita:** význam důležitosti – navrhuje tester

Určuje pořadí ve sprintu

**Severita** – závažnost problému, jak rychle ho chceme vyřešit, hodnotíme v týmu

Je ovlivněna počtem lidí, zda příjdem o peníze, reputace firmy,co vytváří aplikaci

(přiklad: vysoká severita a priorita – platební údaje)

Nízká severita, nízka priorita – gramatická chyba, překlep

Nízka priorita, vysoká severita – například helthcare management software nezobrazí správně informace o alergii pacienta, můžete být životu ohrožující událost

Severitu nastavuje seniornější tester

**Status code**

400 – bad request body není nebo headers špatně zapsáno, špatně body , headers

**401- neauthorizováno nepřihlášeno**

**403- zakázáno, víme, kdo je uživatel, nemá přístup do stránky**

404 – not found (problém v URL)

405 – metoda not allowed, špatná metoda post nebo get

200

201

300 – přesměřování uživatele

**Aktivity v průběhu IT projektu**

1.Analýza

-účastní se analytik, PO, odborníci, zástupci uživatelů

Před analýzou se provádí interview, průzkumy, prototypy - ) výstupem je specifikace (hlavně srozumitelná všem), Pozadavaky, use-case, user stories

2.Návrh

Jak si poradit s komplexitou, **architektura** s více vrstvami

*prezenční vrstva* – frontend (UI=uživatelské rozhraní , komunikuje s uživatelem)

*aplikační vrstva - backend* = jádro/srdce aplikace, logika,fce, výpočty a rozhraní API

*datová vrstva* – bezpečně uloží data a umožní dotazování

P5. Prezenční frontendová vrstva – Mobilní aplikace, webová, desktopová, CLI, více frontendů s jedním backendem

3.Implementace

Programování, program = recept, algoritmizace

Programovací jazyky

a)Webové: Frontend – HTML (značkovací jazyk), CSS (selektivní jazyk), JS, Backend – PHP

b)mobilní- Android – Java (Kotlin),iOS-Swift

c)univerzální jazyky -C#,Java

Programujeme z polotovarů

•**Knihovny,Frameworky** = česky aplikační rámec nebo vývojová platforma, je softwarová struktura pro podporu programování, vývoje a organizaci jiných softwarových projektů. Může obsahovat Knihovny API

•JavaScript-React, Angular, Vue.js

•PHP- Nette , Slim, Laravel

•CSS- Bootstrap

Open.Source(OSS)

Programování

Nástroje:

•IDE (Integrated Development Environment)

Visual Studio,PhpStorm

•Editor kódu- Visual Studio Code, Sublime Text

•Verzovací system git

Programovací prostředí: vývojové, testovací, produkční

4.Testování

Programátoři si testují vlastní kód (unit testy, integrační testy).

Testeři proklikávají aplikaci. Specializované testy: výkonostní, bezpečnostní

Typy testování: manuální a automatizované

Průběh testování - ) Vytvoření testovacího protokolu (Test Case). Pokud se najde chyba (BUG) vytvoří se Ticket v systému

**Testerské nástroje**

•JIRA, Selenium, Cypress.io

5.Nasazení do provozu

6.Provoz

Podpora: Zákazníci musí být spokojeni

Level 1-řešení základních požadavků dnes často přebírá AI

Level 2-řešení pokročilejších problémů

Level 3-chyby řešené programátory

Nástroje: JIRA, ZenDesk

SLA - Service Level Agreement

Řízení

**Agilní vývoj**



**Stakehodler** = zůčastněná strana

**Persona**= používá se k popisu hypotetických uživatelů nebo zákazníků produktu, abychom lépe porozuměli jejich kontextu a k čemu jim aplikace bude sloužit

Způsob definice požadavků na systém, který máme jako tým doručit

**SDLC** – software development life cycle – životní cyklus vývoje Softwaru

Činnosti provaděne v každe fazi vyvoje softwaru a

vzájemné logické a chronologicke souvislosti mezi

nimi.

Jde o agilní sprint

Zahrnuje

1. Plánování and maping
2. Analýzu
3. Design
4. Implementaci
5. Testing a integraci
6. Maintenance – údržba

Jak dlouhý je cyklus, kolik cyklů, kdo je kde zapojený

**Sekvence:** Každá fáze začíná po předchozí, nepřekrývají se, jedno velké doručení, dlouhá doba

**Sekvence jsou tvořeny:** Iterative – v časovém úseku a Incremental – části práce, kterou dodáváme (Agilní přístup = funkční kousky skládat do finálního produktu)

Cílem je zbavit se fází - pracovat iterativně a flexibilně

Zvýšit efektivitu a pracovat na propojování jednotlivých úkolů s obdobnými funkcionalitami

Dodávat pravidelně malé incremety- části práce – místo velkého balíku dodání na konci

Může trvat delší dobu, ale je rozděleno na kratší části - zvládatelné segmenty, aby se zajistil průběžný pokrok a přizpůsobitelnost

Najít funkční řešení, kooperace s klientem, potřeby s klientem se mohou měnit

Agile = soubor praktik, nejčastější metodika scrum

Agile = komunální proud realné hodnoty, kterou udržujeme v tempu

Agile= soubor různých technik, které mají společnou metodologii

Agile je souhrn metodologií SDLC SCRUM,WATERFALL,CANBAN,SCRUMBAN

Manifest Agilního vývoje software

Objevujeme lepší způsoby vývoje software tím,

že jej tvoříme a pomáháme při jeho tvorbě ostatním.

Při této práci jsme dospěli k těmto hodnotám:

**Jednotlivci a interakce** před procesy a nástroji

**Fungující software** před vyčerpávající dokumentací

**Spolupráce se zákazníkem** před vyjednáváním o smlouvě

**Reagování na změny** před dodržováním plánu

Jakkoliv jsou body napravo hodnotné,

bodů nalevo si ceníme více.

**Výhody:**

**Benefit**: doménová znalost, zlepšování projekt od projektu, získávání více informací, týmová práce, testeři jsou součástí vývoje

Věci lze analyzovat, než jsou skutečně vyvíjeny (požadavky, prototypy atd.)

Věci jsou dodávány v malých, testovatelných kusech - ověřitelných

Není to jen o hledání chyb; při mluvení o chybách je nejdůležitější je předcházet.Cílem je zkracování feedbackových smyček.

Zákazníci jsou zapojeni – poskytují zpětnou vazbu a připomínky

Iterative plus Incremental = Agile

Co dělá tester při jednotlivých ceremoniích

Obvykle junior testaři nemluví se zákazníky a vyššími šéfy; jsou někde v pozadí, dokud nezískají dostatek znalostí nebo zkušeností na to, aby mohli reprezentovat tým. Tento proces obvykle trvá od 2 měsíců do 2 let

Celkově by měl být tester zahrnut do všech ceremonií, aby lépe porozuměl projektu, produktu, identifikoval klíčové osoby, kterým může klást otázky, a tak dále. Také je třeba přemýšlet o tom, jak bude testována konkrétní funkce, poskytovat odhady a vyjadřovat obavy ohledně implementace (pokud nějaké vidíte). Ale toto platí spíše pro scrum model.

Ve více tradičním světě podobném modelu waterfall existuje přísná hierarchie a možnost mluvit mají pouze vedoucí týmu a zkušenější QA; junior testaři na začátku nemají takové možnosti.

Tester se účastní všech ceremonií – doptáváš se

Grooming – zda spadá do definition of ready, před začátkem sprintu, zda rozumíš úkolům,sizing

Planning – prvních pět nejdůležitějších vytáhneš ? kolik zvládnou story pointů, estimace na základě hotového sizingu v groomingu, ve scrumu není vedoucí, podporuje , na zíkladě zkušeností odhadnou svoje schopnosti

Sprint review

Stand up – co dělá, kde se zasekl, co mu jde

1. SCRUM/STAND UP – pracovat v týmu pro společný cíl

Synchronizace s týmem, koná se zpravidla každý den, cílem stanovit si krátké cíle, měl by být super rychlý

Jednotliví členové reportují svou odvedenou práci, na čem budou dále pracovat a jestli mají nějakou překážku, která jim znemožnujě dále pokračovat

Synchronizace práce s týmem, krátký call

Plánování práce na následujících 24 hodin.

* Každý reportuje svou odvedenou práci, na čem bude pracovat a zda-li má nějakou překážku, která mu znemožňuje dále pokračovat.Kontrola pokroku směrem k cíli sprintu. Podrobné diskuse se provádějí po ranním setkání v menších skupinách. Ranní setkání je o poslechu ne o povídání, super rychlý

**Backlog**

* Náhled na to, co tým bude dělat v průběhu projektu
* Místo, které shromažduje veškeré projektové úlohy (EPIC,US,TASK,BUG), které jsou definované pro dokončení celého projektu
* Živá věc, často se mění, spravuje ho PO

PBL´s Features (vlastnosti) jsou součástí funkcionalit , které mají být vyvinuty v rámci projektu nebo produktu. Tyto vlastnosti jsou specifikace toho, co by daná aplikace měla dělat nebo jaké funkce obsahovat. V rámci agilní mentodiky jsou tyto vlastnosti rozdělený na podúkoly v rámci jejich priority a důležitosti pro zákazníky

Dělíme na :

EPIC – typ projektové úlohy, který popisuje větší část funkcionality vyvíjeného systému. Například přihlášení/registrace, platební metody aj.

Epic se dále dělí na menší projektové úlohy USER STORIES a ty se dále dělí na Tasky

USER STORY

Obsahuje technickou specifikaci, nebo je na ni prolink

* Specifikuje, jakou hodnotu máme zákaznkovi dodat
* Dílčí funkcionalitu systému
* Měly by být psány z pohledu uživatele
* Měla by obsahovat akceptační kritéria AK – tedy kdy bude zákazník považovat US za kompletně hotovou
* Mezi AK může například patřit

1. Implementace kódu
2. Unit testing – co to je ?
3. Analýza požadavku
4. Test design
5. Množství – co to je ?
6. Severita bugu – co to je ?

Každá US je estimována pomocí tzv. story pointů – odráží náročnost dané US (technickou a časovou). Jak dlouho bude implementace US trvat a jak je pro testera technicky náročná. Čím vyšší hodnota SP, tím náročnější estimace (individuální v rámci firmy)

**Estimace:** při sprint planningu meetingu, odhad jak dlouho nám naše práce bude trvat. U tasku je to několik hodin u US několik dní. Jak často bývá planning meeting?

Task – úkol

* Je typ projektové úlohy, která je součástí US. Soubor tasků pod danou US má za cíl implementovat funkcionalitu popsanou v US a doručit tak hodnotu zákazníkovi

(implementace kódu pro vývojáře, design testů pro testery, definice požadavků pro PO aj), estimace tasků v hodinách než daná US bude dokončena

Product backlog Iceberg nás informuje o důležitosti plánování

**planning** – společný meeting všech lidí v týmu, kde se zavazujeme, jako funkcionalitu a hodnotu doručíme zákazníkovi v rámci jednoho sprintu. Stanovujeme odhad práce a zda je reálné doručit danou funkcionalitu v rámci daného sprintu. Kolik máme kapacity, Sprint je závazek /commitment přechází z Backlogu do sprint backlog a následuje fáze sprint planning

1. **Sprint planning**

* Koná se na začátku každého sprintu a je základní součástí agilni metodiky SCRUM
* Rozdělují se tasky mezi developery a testry, flexibilita týmu důležitá
* Časově omezený rámec na 2 až 4 hodiny
* Stanovení obchodních cílů, sprint goalů, na základě odhadu a předpokladu zařadit dané itemy do sprint backlogu
* Rozdělení práce a plánování práce, odhad trvání jednotlivých úkolů
* Vezmeme z Backlogu do Sprint Backlogu důležité TOP úkoly, ale jedině, když se to setká **s Definition of Ready** ( Způsob práce a srozumění, který tým má) Pochopit daný úsek práce: Proč, pro koho, možná rizika. Soubor kritérií, že můžeme s položkou pracovat. Příklad. Co je uvnitř storky (tittle, summary, seznam akceptačních kritérií, priority, smart požadavky, odhad, hloubka)
* Efektivně přemýšlet nad SIZING (tým musí znát velikost jednotlivých items a dobu práce). Přemýšlet nad rozdělením malých a velkých úkolů v rámci týmu. Mít jasný plán a znalosti, jaké techniky a nástroje použít. Lepší rozdělit items na měnší úkoly.

**Smart požadavky** – ohodnotí requierement, jestli je testovatelný nebo ne. Rozumím požadavku, musíme změřit cíle

**Estimace:** při sprint planningu meetingu, odhad jak dlouho nám naše práce bude trvat. U tasku je to několik hodin u US několik dní. Jak často bývá planning meeting?

V rámci sprintu:

TO DO, Dev in Progress, Ready for test, test in Progress, ready to accept and accepted.

Sprint prochází různými stavy, workflow of sprint. Na konci daného sprintu musí být Hotovo **Definition of Done** = Vyvinuto, otestováno, opraveno,automatizováno, zdokumentováno aj..Stanovem Scrum team, kdy je vyřešeno, seznam věcí, které musí splňovat – otestováno,zautomatizováno, splňuje požadavky, nejsou defekty, jsou splněna akceptační kritéria. Developeři opravili chyby. Technical writer doplnil dokumentaci

Finishing the sprint na základě Customer acceptance/Akceptační kritéria

1. Backlog Grooming (refinement)

Probereme úkoly na další sprint i s PO

SCRUM Master a PO zkontrolují, zda zadání rozumíme

Konzultace s PO – diskuse o specifikacích jednotlivých tásků, které jsou součástí PBL´s a připraveny ke groomingu

Tým poskytuje zpětnou vazbu a ptá se na otázky. Také jiné týmy, lidé mohou přinášet témata k řešení

Lidé v týmu si vyjasňují, jak bude funkcionalita, kterou se tým zavázal implementovat v týmu fungovat, jak to chápou všichni lidé v týmu a zda je to v souladu s tím, co chce zákazník

1. Sprint review/DEMO

* Společný meeting všech lidí v týmu, kde se prezentuje dodaná funkcionalita v rámci uplynulého sprintu
* Měl by se účastnit i zákazník nebo jeho zástupce a reagovat, pokud vyvinutá funkcionalita je v souladu s požadavky klienta
* Včas podchytit situaci a zjistit nesrovnalosti, jak to klient chce a nechce – uvědomění si potřeby

Demo reprezentuje Úspěch/Neúspěch.

Získáme zpětnou vazbu.

Představit cíl pro další sprint.

Přizpůsobit produktový backlog

5. sprint retrospective

Zpětná vazba, zlepšení funkčnosti týmu, ne skateholder – většinou se na něj nadává

* Je to konstruktivní, společný meeting s týmem, kde se vyhodnocuje uplynulý sprint z pohledu lidí v týmu (co šlo dobře, co špatně, co potřebuje zlepšení)
* Může se vypíchnout i konkrétní úspěch jednotlivce či týmu
* Cíl je neustále zlepšovat proces a odstranit problémy, které se vyskytují v týmu

Delivery/Doručení

Položka z produktového backlogu nebo úkol, by měl být schopen k implementaci bez nutnosti zasahování, úprav

Zahrnuje řešení: analýzu, návrh aj. jak dosáhnout, aby položka splňovala kritéria pro doručitelnost

Zprávu:dokumentace, která nás informuje, zda položka splňuje kritéria pro doručení

Ostatní: testování, kvalita, zajištění kódu aj.

|  |  |  |
| --- | --- | --- |
| SCRUM | | |
| **5 ceremonií** | **3 roles** | 3 increments |
| Sprint | Product Owner | Product Backlog |
| Sprint planning | Scrum Master | Sprint Backlog |
| Daily Scrum | Team Development | Product Increment |
| Sprint Review |  |  |
| Retrospective |  |  |
| Plus Backlog Gromming |  |  |

Popsat co je **Product Increment**- je výstupem z kazdého sprintu v metodice SCRUM

Jiné agilní přístupy

1. Iterative

* LeSS - Large Scale Scrum
* SAFe - Scaled Agile Framework
* Nexus
* Spotify
* RUP - Rational Unified Process

1. Continuous

* Kanban (metoda vizuálního řízení pracovního toku, používá se napřiklád v softwerovém vývoji a v projektovém řízení
* Lean (může být aplikován v různých kontextech a odvětvích a je často používán k dosažení větší efektivity, zvýšení kvality a dosažení vyšší spokojenosti zákazníka)
* Extreme programming (je agilní metodika pro vývoj softwaru, která byla vyvinuta s cílem zvýšit kvalitu a efektivitu vývoje a soustředit se na potřeby zákazníka. XP klade důraz na spolupráci, jednoduchost a rychlost dodávky softwaru)

1. Hybrids

Mezi iterative a Continuous a mezi Agile a Waterfall

Product owner/Projektový manažer

Nastavení pravidel hry, rozsah, plánování a koordinaci, rizika ?

* Řídí tým?
* Řeší obsah (přání klienta), zastupuje v týmu pohled zákazníka, má největší znalosti o vyvíjeném systému
* Definuje požadavky,rozděluje a třídí úkoly, soustřeďuje hodnotu – motivací je doručit nejvyšší možnou hodnotu zákazníkovi
* Zabezpečuje, že rozumíme, co po nás chce
* Není členům týmu nadřazen, spíše členy týmu podporuje a zajišťuje informace, které potřebuje k úspěšnému doručení softweru
* Transaprentní ,viditelné, srozumitelné
* Projekťácké nástroje: MS Project (WBS). JIRA. Trello, Google Drive, OneDrive, Confluence, Sharepoint apod.

SCRUM MASTER

Nastavení pravidel hry, rozsah, plánování a koordinaci, rizika. Stará se o to, aby byl scrum v týmu efektivní

Role zastoupena v projektech řízených agilní metodikou

* Řízení obsahu, zodpovědný za to, že nikdo nemá žádnou překážku, zeefektivnění týmu, pozitivní motivátor, chrání tým před vyrušováním
* Facilitátor, řeší problémy,vyvcvičuje tým k soběstačnosti, pomáhá PO

Development tým - odpovědný jako tým, doručuje potencionálně uvolnitelné produktové části – increments, samoorganizovaný, cross functional

Funkční testy- jazykové mutace, finanční částky

Testuje**, co aplikace dělá**

**-testuje funkce aplikace a ověřuje, zda fungují podle očekávání**

**- navázáno na business requirement, například omezeny E2E test**

Nefunkční testy – testuje, jak to aplikace dělá

testuje další aspekty aplikace, které ovlivňují spokojenost uživatelů a kvalitu aplikace

Performace testy: Hodnotí, jak se systém chová při určité zátěži, testují spolehlivost, rychlost, škálovatelnost, odezvu

**Performance** – výkonnost, **Security**

**Usability** – Použitelnost, **Compatibility** – jak moc funkční v různých typech zařízení

**Maintainability** – udržovatelnost

**Scalability**- kolik uživatelů používá

**Availability** – dostupnost zařízení, jazyka, verze,**Integrity**

**Testování v praxi:**

Testování ukazuje přítomnost bugu, ne jejich absenci. Upravujeme síť TC v průběhu testování. Vyčerpávající testování je nemožné, musíme si stanovit hranice (počet lidí, čas), určit konec testování na základě společného koncensu. Testování poskytuje informace, na základě nichž se někdo další rozhoduje (Stakeholder.) Není naše práce rozhodnout o konci projektu.Včasné testování je levnější a bezpečnější. Čím dříve na problém přijdu, tím rychleji můžeme opravit. Jak zajistit, že se bug neobjeví po releasu ? Intenzivní spolupráce s developery, dobrá specifikace a specifikace požadavků. Zajetá pravidla na kontrolu kódu. Zaměřit se na úsek největší chybovosti. Snaha odhadnout bugy a předcházet vzniku bugů.

Defekty se shlukují v pár modulech. Aplikace v návaznosti úkolů několik souvisejících defektů, zmínit v reportech, najít kořenovou příčinu.

Chyba v prvku na více místech aplikace, dobré ověřit, hledat příčinu defektu, zaměřit se na testovací oblast.

Testing je přístupný, kdokoliv může najít chyby

**7 principů SW testing**

1.Testing ukazuje přítomnost defektů, nikoliv jejich absenci

2.Vyčerpávající testování je nemožné

3.Brzké testování šetří čas a peníze

4.Defekty se shlukují

5.Buď opatrný ohledně Pesticidního paradoxu

6.Testování je závislé na kontextu

(legislativa, jazyk, businness doména, země)

7. Absence chyb je klam

Když neznáme aplikaci, mužeme kouknout na podobnou

Když něco otestujeme a nemá to chybu, neznamená, že je to bez chyb

Statické testování – testujeme bez spouštění kódu , white box

Př.Branch Coverage,Walkthrough, Peerreview

Př. Kód neběží, ale najdeme v Gitu například heslo Admina uvnitř kódu

Nástroje na statické testování kódu: Automat code review

Např. špatný protokol či knihovna z internetu

Dívám se na scénáře, testuji data, požadavky

Dynamické testování- kód běží, proklikáváme se běžící aplikací

Př.Scalability

White box – tester zná vnitřní strukturu aplikace, kód, návrh, implementaci

Zaměřuje se na jednotlivé komponenty, moduly

Př.integrační testování, systémové testování

(neřeším, co se děje uvnitř kódu) a black box ( příklad pokud aplikace obsahuje jen API), funkční a nefunkční testování

BB řeší vnější chování, neznáme vnitřní strukturu, na základě vstupů a výstupů zjišťujeme informace

Příklad. Akceptační testování, integrační testování

State transition testing-

Typ softwerového testování, který se provádí k ověření změny stavu aplikace při různých vstupech - pozitivní a negativní vstupy

Oetstovat výkon systému, otestovat chování systému při různých vstupech, otetsovat změnu přechodového stavu aplikace

V: pomáhá porozumět chování systému, reprezentuje správné chování systému

N.není možné provádět všude a není vždy spolehlivé

**Analýza hranic (BVA):** Analýza hranic je technika testování softwaru, která slouží k prozkoumání chování aplikace pomocí testovacích dat, která existují na hranicích nebo extrémních hodnotách rozsahu vstupních dat. Jednoduše řečeno, pro daný rozsah hodnot vstupních dat se používají hranicové hodnoty (hodnoty na horních a dolních limitech) jako vstupní data pro testování. Tato technika je široce využívána, protože se předpokládá, že chyby jsou pravděpodobnější na těchto hranicových hodnotách.

**Co je Ekvivalence Partitioning (EP):** Ekvivalence Partitioning je další technika návrhu testovacích případů, která rozděluje vstupní doménu programu do tříd dat, ze kterých lze odvodit testovací případy. V této technice jsou hodnoty vstupu rozděleny do skupin a testovací případy jsou pak navrženy tak, aby pokrývaly každou skupinu. Cílem je snížit celkový počet testovacích případů a zároveň zajistit adekvátní testování každé skupiny. Ekvivalence Partitioning je obzvláště užitečné v případě velkého množství možných hodnot vstupu.

**BVA vs Ekvivalence Partitioning:** Obě techniky, Analýza hranic a Ekvivalence Partitioning, jsou nástroje pro návrh testovacích případů, které si klade za cíl optimalizovat proces testování. Nicméně se zaměřují na různé aspekty:

1. **Zaměření:**
   * **BVA:** Soustředí se na testování hranic, protože se předpokládá, že chyby jsou pravděpodobné na extrémech rozsahů vstupních dat.
   * **Ekvivalence Partitioning:** Rozděluje vstupní doménu do tříd a zaměřuje se na testování reprezentativních hodnot z každé třídy.
2. **Účel:**
   * **BVA:** Zajišťuje, že software se chová správně na hranicích vstupní domény.
   * **Ekvivalence Partitioning:** Snaží se snížit počet testovacích případů a zároveň poskytnout efektivní pokrytí testů testováním reprezentativní hodnoty z každé partiční skupiny.
3. **Aplikace:**
   * **BVA:** Často se používá, když jsou vstupní doméně specifické hranice nebo hraniční hodnoty, které vyžadují důkladné testování.
   * **Ekvivalence Partitioning:** Užitečné, když lze vstupní doménu rozdělit do různých tříd nebo partičních skupin.
4. **Scénář:**
   * **BVA:** Vhodné pro situace, kde se očekává, že software selže při extrémních hodnotách vstupu.
   * **Ekvivalence Partitioning:** Efektivní při velkém množství možných hodnot vstupu a testování každé hodnoty individuálně by bylo nepraktické.

Shrnutím lze říci, že Analýza hranic a Ekvivalence Partitioning jsou obě cenné techniky v testování softwaru, každá se svým zaměřením a použitím v konkrétních scénářích. Analýza hranic klade důraz na testování na hranicích, zatímco Ekvivalence Partitioning se snaží optimalizovat pokrytí testů testováním reprezentativních hodnot z různých partičních skupin vstupní domény.

Testování ve 4 kvadrantech, vztahuje se k businessu či více k technologickým aspektům

Vztah k práci v týmu či produktu samotnému

Automatizovaně či manuálně či se testuje za pomoci speciálních nástrojů (jaké)

Testování založeno na riziku

**Riziko:** vnímání, co se může stát, jaký může mít dopad a s jakou pravděpodobností.

Dopad (finanční, právní a ohrožující zdraví)

**Riziko:** projektové, procesní (nedostatečné vybavení, odlišné přístupy týmu), produktové

Riziko má pravděpodobnost a dopad

= míra pravděpodobnosti, co se stane, když neřešíme zavčas

Vysoká rizika se testují , co nejvíce (určuje analytik)

Risk, jak budeme hodnotit a pracovat s ním

Test scoping – co budeme testovat

Test scaling – jak moc hluboko

Test prioritisation – kdy a jak ?

Vysoká pravděpodobnost rizika – velká sada dat, vysoký dosah, více prozkoukat- zatěžuje klienta

Risk based testing – zaměřit se efektivně na rizika,regresní

Zhodnotíme pravděpodobnost a dopad na základě rizik objevených a soustředíme se , kde je vysoké riziko a vysoký dopad

Měnit testy a testovací data,zaměřit se na testy s riziky, funkcionální a nefunkcionální charakteristiky

Pesticidní paradox

Bugy se stávají imunní proti určité strategii testování

Stejná sada testů/testovacíh sad -) přestaneme nacházet bugy, ale neznamená, že nejsou

**Řešení:** doplnit testovací sadu, provozní slepota

Regresní testy:

Opakujeme stále dokola stejné testy

Přetestováváme věci, které dříve fungovaly. Přidáváním nových věcí mohlo dojít ke změně či zásahy do kódu nevyprodukovaly další bugy

**Principy regresního testování:**

Co fungovalo v prvním sprintu, již nemusí fungovat ve druhém

Předcházíme tomu a provádíme testy

Vybrat regresní sadu pro testování je složitý proces ? Vybrat testy, kde bylo nejvíce chyb, testují funkcionalitu, která je nejvíce využívaná. Testují část systému, která podléhá častým změnám. Testují tu nejkritičtější část systému.Čím častěji spouštíme tím lépe, testy se vytvářejí a prioritizují.

Regresní testy jsou jen v agilním prostředí ?

**Retestování**-kontrolujeme, zda jsou bugy opravené

Principy regresního testování

* Kompletní regrese
* Selekce testů (testy , které jsou důležité a méně důležité), dělíme na manuální a automatizované testy v rámci regrese
* *Méně podrobné testy, více přemýšlíme o odchylkách*
* *Tester analyzuje požadavky, vytváří testy, spouští testy a testuje. Poskytuje zpětnou vazbu, reportuje chyby. Přetestovává Účastní se ceremonií a ptá se na otázky*
* *V jaké formě můžeme požadavky obdržet – User story, Epic, use case, personas, dokumentace, email…*
* *Mind Mapa- vizualizační scénář*

Contininious intagration **CI**

je softwarový vývojový postup, který je navržen k zlepšení kvality softwaru a urychlení vývojového cyklu. Hlavním cílem CI je pravidelně a automaticky integrovat změny provedené v kódu do společného repozitáře a následně automaticky provádět sestavení, testování a nasazení softwaru. To se provádí opakovaně, často několikrát denně.

Hlavní prvky: automatizace, pravidelnost, automatizované testování,nasazení a zpětná vazba

* Programátor naprogramuje, zda testy spolu interagují (aut.testy), procesy ověří kompatibilitu
* Integrace plus testing, kód se naprogramuje, built(nastaví)
* Automatizovaný proces vývoje, který okamžitě po provedené změně v repozitáři integruje, testuje a slučuje provedené změny

Continious delivery **CD**

je moderní metodika v oblasti softwarového vývoje, která staví na základech Continuous Integration (CI) a má za cíl zlepšit proces doručování softwaru tím, že umožní rychle a spolehlivě nasazovat softwarové změny do produkčního prostředí. Continuous Delivery je spojeno s průběžným testováním, automatizací a zjednodušením procesu nasazování, což vývojovým týmům umožňuje mít vždy funkční a připravenou verzi softwaru k dispozici, kterou lze nasadit do produkčního provozu s minimálními riziky

Klíčové prvky:

Automatizace,pravidelnost, důvěryhodné a rychlé testování, škálovatelnost, konzistentní prostředí, monitorování a zpětná vazba

* Testy projdou (jdou do produkce /produkční server zíkazník)
* Posílá se po malých kouscích- malé části kódu, určitý úsek, úprava algoritmu

Mezitím Continuous Deployment – důkladnějí testy, automatizované

Test scenario: obecné info /high level co chci testovat. Konkrétní akce prováděná s určitou softwarovou komponentou a její očekávaný výsledek.

* Scénáře s důležitější funkcionalitou by měly mít vyšší prioritu, pomáhá sestavit testovací plán

Kvalitní scénáře jsou takové, které lze snadno následovat, provádět, zeefektivňují proces, šetří čas

Lepší kratší a jednodušší

Test case (recept) – návod , jak otestovat nějakou funkcionalitu

Popis kroků, popisujeme testovací scénáře

Podmínka, akce,výsledek

Přesná důslednost kroků, konkrétní. Provádíme instrukce napsané v testech. Zaznamenáváme výsledky testu Pass/Fail , OK/NOK. Pokud nalezneme chybu nahlasíme.

Jméno, heslo, email, vyplním a zmáčknu tlačítko

Nejvyšší priorita 1…..nejnižší 3

Test analýza -(business analýza (čím je část systému důležitější, tím větší prostor v testování by jí měl být věnován) – hlavní předmět aplikace, to co generuje zisk, mohou nést právní následky- nedostatečné zabezpečení citlivých dat, bezpečnost and functional- regresní testy ověřují opravenou funkcionalitu. Dokumenty vzniklé při analýze(funkční analýza – je na začátku procesu a funkční specifikace – vzniká později, jak fungovat v rámci aplikace). Výsledky komunikace s ostatními členy týmu.

Znalosti: čemu se více věnovat dle zkušeností testování jiných aplikací, obecné očekávané chování

Test analýza: Najít konkrétní funkcionality, které budeme testovat (happy path /unhappy path) v rámci jednotlivých fází

**User stories:** detailnější popis epicu

**Testovací podmínky:** co můžeme otestovat

**Testovací data :** potřebná pro testování, link na aplikaci

Které testovací techniky použijeme

**Analýza** požadavku,v jaké formě můžeme požadavky obdržet

S pecific

M easurable

A ttainable

R elevant

T ime based

V jaké formě můžeme požadavek obdržet: user story, epic, use case,personas,dokumentace,email

Use case = popisuje, co uživatel může dělat na stránkách či v systému

USER STORY = má v sobě několik UC

**Waterfall**

Vodopádové řízení projektu,jasně definovaný úsek, ukončený, jednodušší na plánování

Hotový výrobek retestujeme

Používáme, když je stanoven přesný termín dokončení/dodání např. v automobilkách, kde jsou složité procesy. Fixovaný scope, zadání stále stejné.

**Iterativní agilní řízení projketu**- v cyklech dodáme část produktu. Zasahujeme do aplikace, programujeme, najdeme bugy

**Skateholders:** zůčastněná strana, zákazník, zapojená strana, lidi venku, nepracují v rámci týmu.Analytici, experti, lidi ze strany zákazníka, investuji čas a peníze, mají vliv na požadavky v průběhu vývoje

**SCRUM** – jedna z agilních metodologií

Cross functional scrum team- zastupitelnost, developeři, testři, UX designer, můžou pracovat na stejné funkcionalitě, od začátku do konce ji vytvořit

Tvorba testů

Vycházíme z analýzy – high level testy, konkrétní testovací scénáře, zastupitelnost s kolegy, zapisujeme a uchováváme

Dokumentace -) analýzy-)TC-)test exekuce-) spouštíme testy

**Best paractices**

Důležté je jednoduchost, srozumitelnost a perspektiva- vcítit se do zákazníka

Pojmenování, popis testu, testovací podmínky – co potřebuji k vykonávání testu

Testovací kroky, testovací data

Závislosti ve vztahu k jinému testu - priority scénáře

Očekávané chování

Nejasná dokumentace: jít za tím, kdo tvořil

V týmu může upravit jeden druhému test na základě souhlasu

**Test** – **prolink na defekt.** Step nevalidní, ale komment doplnit proč.

Bug nalinkuji ke všem TC v cyklu?

Nejsem tvůrcem bugu, ať vím (očičko) watch the ticket kdo tvůrce

**Persona:** Využívání person, abstraktní zástupce skupiny (MA,A,Rodič, student)

Co je chyba: odlišnost od očekávaného chování, není to tam, nefunguje to, funguje to jinak, funguje to pomalu a nepředvídatelně, neřeší to problém, potřebu

Může jít o

•neočekávané chování aplikace,nevhodný designový prvek,

•nepoužitelné nebo nepřístupné uživatelské rozhraní,

•překlep či gramatická chyba v textu,pomalé načítání aplikace,

•nesoulad s právními předpisy,bezpečnostní díru,

**Verifikace** = Kontrola,zda software dělá to, co popisuje zadání

Verifikace zajistí, že je software kvalitní, ale nezajistí jeho užitečnost

Ověřujeme, zda je vše správně na základě dokumentu (requierement)

**Validace =**Kontrola,zda software splňuje potřeby a očekávání uživatele

Obvykle přichází po verifikaci, když se řešení dostane k uživateli

* Zhodnocení, zda souhlasí s potřebami zákazníka (business potřeby)

**Bug tracking nástroje**

-Evidence chyb v rámci projektů, vyhledávání, filtrování

-Přidávání komentářů a příloh, Vazba chyb na požadavky

-Většinou vč. samotné správy požadavků

Co by měl bug report obsahovat:

Nesmí chybět:

Výstižný název chyby,Kroky k nasimulování

Skutečný a očekávaný výsledek (nejlépe s odůvodněním)

Hodí se:

Testovaná aplikace a její verze

Prostředí (testovací, předprodukční, produkční)

Vazba na testovací scénář, požadavek

Screenshot, záznam z logu aplikace

Další přílohy (např. když testujeme nahrání souboru)

Lze uzavřít storku, když má chyby? Musí se shodovat s Definition of Done

(Závažné US musíme zavřít. Ostatní nemusíme)

Musíme fixovat všechny bugy před releasem

* Na základě doporučení v závislosti na závažnosti bugu

**Spouštění testu** – automatizovaně, manuálně i kombinovaně

Můžeme testovat rovnou i podle zadání bez psaní testů

Session based

Exploratory testing

**Exekuce testu:** hlásíme chyby, najdeme při spouštění testů- zadáváme do bugtracking nástroje (ticketovací systém JIRA)

Jsou defekty nebezpečné ?

Zranění, lidské životy, finanční újma, nesplnění norem GDPR,legislativa

Dopad na termíny, ztráta zákazníka, ztráta reputace

Je to vůbec bug ?

Špatně pochopený TC, zlé prostředí, špatný vstup, zlý TC, nesplněné preconditions

Proč je dobrý report pro bug důlěžitý?

Jednoduše, spolehlivo a opakověně- někdy se k němu můžeme vracet, bude ho opravovat někdo jiný, aji retest může dělat někdo jiný

Krátký výstižný název = Report bug

Shrnutí

Dopad na zákazníka,Kroky jak chybu zreprodukovat (video , Gift, screenshot)

Očekávaný výsledek: aktuální výsledek

Nízka priorita : vysoká severita (vážnost)- určuje Test Lead, Project manager

**Timestamp** - datum a čas, kdy se vyskytla chyba

**Prostředí**, **prohlížeč, URL adresa, operační systém, verze aplikace**

Co nedělat?

Neitenpretovat jako chybu vývojáře, nedostatek informací

Nedá se retestovat

Ticketovací nástroj – například JIRA

Informační systém pro zadávání obecných požadavků a úkolů ticketů (issues)

Můžeme sledovat aktuální stav, jednotlivé kroky procesu, co můžeme dělat

Sledujeme role: zadavatel, řešitel

Definuje workflow- životní cyklus

Přehled a dohled nad kritickými parametry

**Hlídá u tiketů kritické parametry jako termíny, priority pro management**

Proč používat helpdesk- vhodné pro organizaci práce ve větších týmech

Zpracování a koordinace klientských dotazů

Udržuje historii u projektu, ulehčuje spolupráci

Nástroje **Bug tracking** - defekt , informuje Project managemenet. Indikuje,že se zaměřujeme na bugy . Častá integrace s dalšími nástroji pro vývoj software

( např. verzovací systém,CI- continous integration,CD- continous delivery, vývojová prostředí, pro správu požadavků a testů)

JIRA není open source projekt, nejrozšířenější bug tracking systém

Zadávání bugů, podpora řízení projektů v agilním prostředí

Propojenost systémů, reporty o automatizaci

**Test cykly:** skupina testů, které testujeme v rovnoceném prostředí (jazyce, prohlížeči aj.)

**Jira entities**

Issue types (bug, user story, epic, task, TC)

**Epic-** obsahově i z hlediska abstrakce největší část celku. Obsahuje dále US, tasks aj.

Issue types – jakýkoliv ticket,který používáme. Rozdělujeme na různé typy jako např. Bug nebo Task

**User story** – popis izolované části určité funkcionality

**Story points** -na týmové úrovni, kolik času nám to zabere

Většinou se vztahuje k implementaci konkrétního US,Tasku

Numerické vyjádření komplexity (1 nižší úroveň, 3- vyšší úroveň)

Zohledňuje komplexitu daného úkolu a případná rizika

**Kapacita:** kolik je tým za dané období schopen dodat. Kdo to dělá z týmu tester junior či senior

Porovnávání s relativně podobným již vyřešeným problémem, o kterém víme kolik času zabral, jak byl náročný

Sprint

Časově omezený úsek v rámci agilního vývoje

Tým na konci doručí dohodnutou funkcionalitu ze sprint planningu

Na konci sprintu je potencionálně releasovatelný produkt

Trvá 1 až 4 týdny

Je zakončen sprint review and retrospektive

Sprint vytváříme během sprint planning meetingu. Můžeme naplánovat víc, ale jen jeden lze v daném čase spustit

**Experience based testing**- už máme zkušenosti, na kterých stavíme

Personas ( story naprofilované na uživatele) – rodič testuje na tabletu, mobilu či komplu

Test Pyramid

Kategorizuje typy testů dle úrovní

Jak robustní testování na jaké úrovni je, definuje odlišné úrovně

Každá úroveň má vlastní specifikace (co testuje, jak, cíle testování a typické bugy)

UNIT/Component testy – píší vývojáři na úrovni kódu, nejvíc testů a nejrychlejší. Soustředují se na testování jedne unit či komponenty

Testování UI/grafické testování. Dřívější testování je levnější

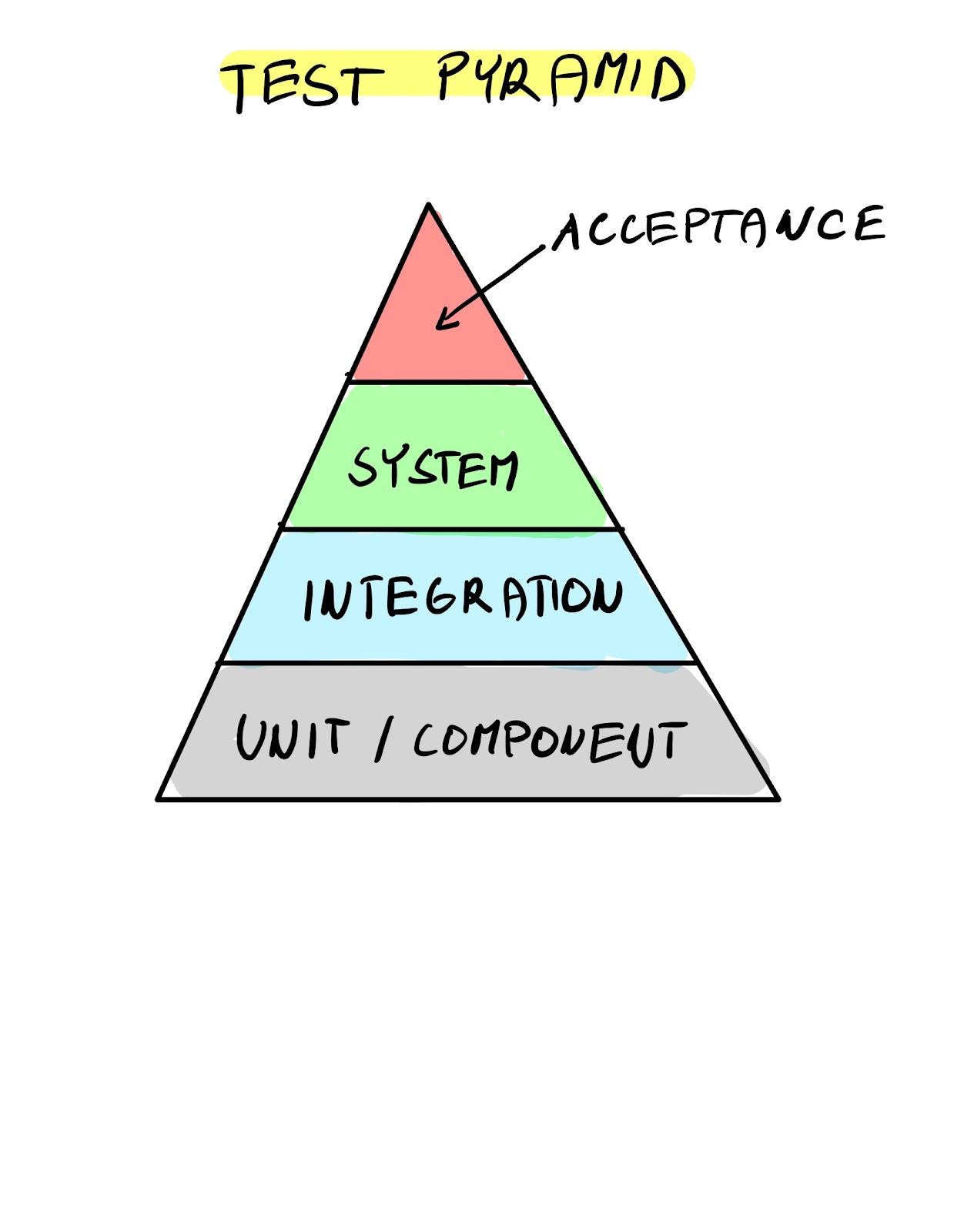
Příklady: testují se základní stavební kameny/jednotky – tlačítko (invoice), vytvořená platební metoda aj. Zda není problém s daty, špatná logika kódu

Jsou to testy na nízké úrovni, zaměřejí se na malé části kódu

Čím více se stoupá v pyramidě vzhůru, méně testů ,ale obsáhlejší, které zabírají více času

Component testy, testují jednu vrstvu

Bez databáze, kontrola testu



Integrační testy: typ softwerového testování

Rychlé testy, v počtu vteřin, nenavázáno s databází

Testují 2 vrstvy

Ověření, kontrola vzahu mezi komponenty v systému, zda fungují správně

Komponenty systému se testují izolovaně

Komponenty se testují izolovaně až po té dohromady a sleduje se jak moduly softwaru interagují, spolupracují spolu a zda spolupracují správně. Nejdřívě komponenta A a potom komponenta B.

Druhá komponenta při testování lze nahradit MOCK A či MOCK B/ falešný/ náhradní/simulátor

Příklad: Vyvíjíte webovou aplikaci, která komunikuje s externím emailovým serverem pro odesílání emailů. Chcete provést IT, ale nechcete zatěžovat reálný externí emailový server, a proto nasimulujete komponentu, která bude imitovat chování emailového serveru.

**Stubs** jsou falešné implementace komponent nebo modulů softwaru, které jsou vytvořeny, aby nahradily komponenty, které ještě nebyly vyvinuty nebo jsou nedostupné pro testování

Stubs – stejná architektura (není databáze) – nahradíme json soubory, bude nám databázi emulovat

**Drivers** jsou speciální komponenty nebo moduly vytvořené pro testování nižší vrstvy software, když vyšší vrstva ještě není dostupná nebo není implementována.

Driver je Postman, když není UI, testujeme server pomocí Postmana

System testing

* Funkční testing, User perspective (UI) - Z pohledu uživatele
* Mostly black box
* Non functional testing
* Vše musí být na svém místě
* Chcemě vědět, jak to funguje jako celek

Acceptance/Accepting testing

* Kontrola, zda aplikace splňuje požadavky zákazníka
* Ready pro produkci, check list/confirmation
* E2E =End to end testy- reálný uživatel si projde, jednotlivými kroky aplikace od zaregistrování přes vytvoření přihlášky a její stáhnutí a zaplacení, testování na všech vrstvách
* Simuluje koncového uživatele, testování na UI vrstvě
* USER Interface testy- zapojení reálných uživatelů

CHECK list =

Například USER CAN Login on to application PASS/FAIL

Nalezení bugu větší pravděpodobnost, ušetří čas

Např.Smoke testy

Release Check list

**UX** je User eXperience, jak zákazník vnímá vaši webovou stránku nebo mobilní aplikaci

**UI** = uživatelské rozhraní

**Návrháři UI/UX** obvykle kreslí design ve Photoshopu a Figmě, říkají vývojářům, jak umístit tlačítka na obrazovky, jaké akce by měl zákazník udělat, aby provedl konkrétní akci

Flow chart: Vývojový diagram (anglicky Flow chart je grafické znázornění procesu, sekvence kroků, postupu nebo algoritmu. Cílem je znázornit tok kroků procesu od začátku do konce grafickým způsobem, který může být lépe pochopitelný než jen pouhý slovní popis.

**Front – end** vývojář: vytváří kód uživatelského rozhraní pro webové stránky a aplikace

Frontend vs backend

**Frontend je něco jako “přední konec”, neboli to, co je vepředu, to, co je vidět. Backend, tedy “zadní konec” je pak to úúúplně vzadu.**  věci, které jsou viditelné, a nebo se pomocí nich definuje vnitřní chování nějaké aplikace.

HTML je značkovací jazyk, pomocí kterého udáváte strukturu a formu webové stránky. Jazyky, které fungují na klientovi tedy v prohlížeči

Technologie beckendu běží na serveru (programovací jazyky: Java,Python). Pomocí nich naprogramujeme, co bude aplikace dělat

Beckend vše připraví, aby fungovalo (na serveru) a potom to odešle na frontend (na klienta), kde se už jen finalizuje, jak to vše bude ve výsledku vypadat.

Základy webových prohlížeču:

Browser = webový prohlížeč, počítačový program, který má konkrétní úkol -) prohlížení webových aplikací WWW – world wide web

Google Chrome, Mozilla Firefox, Safari,Samsung Internet, Edge,Opera

Kde browsery využíváme: PC, mobily, tablety

Browser komunikuje s webovým serverem ( dle pokynů uživatele, který klikne na odkaz či zadá URL adresu)za pomocí značek, stylů aj. se zobrazí stránky

Pro zobrazení některých zvláštních součástí stránky (např. Flash animace) je třeba prohlížeč doplnit o specializované zásuvné moduly (plug – iny)

Stránky nebo aplikace se mohou v různých prohlížečích zobrazovat různě. Při testování testovat pohyb bez myši (šipkami) a myší, důležité zvážit verzi prohlížeče

Pokud je v prohlížeči opravena chyba nebo je přidána nová fukcionalita, dojde k aktualizaci

Kde najít nahoře v google chrome- Menu:3 tečky(pravý horní roh)- help- návod pro ostatní prohlížeče

URL adresa – adresa, end point

Uniform Resource Locator

Sobour znaků, k identifikaci přesného umístění informací v systému

Definuje doménovou adresu serveru, umístění zdroje na serveru a protokol

IP adresa

Kombinace číslic , Internet Protocol address

Identifikuje zařízení

2 formáty: starší Ipv4 – přestává se používat (omezená kapacita)

4 čísla oddělená tečkami

Ipv6- komunikuje přes číslice a písmena

Kombinace čísel a písmen oddělená dvojtečkami

Pomocí příkazového řádku (CMD) zjistíme IP adresu..napíši IP config

[www.mojeip.cz](http://www.mojeip.cz) nebo v komplu whats my IP

Typy adres

Statická- přidělená na pevno a nemění se. Potřebujeme ji mít na 1 místě (servery) stejná adresa dnes a zítra

Dynamická: přiřazená dočasně, každý den je jiná adresa

VPN: místo reálné, do internetu zobrazuje jinou adresu, simuluje, chrání uživatele, šifruje

**DNS**

Domain name Systém

IP adresy jsou pro člověka téměř nezapamatovatelné, proto DNS, který složí jako telefonní seznam pro IP adresy

* Přeloží jména našich stránek na IP adresu

13 kořenových DNS serveru, vyrovnávací servery (kde se ukládají informace, které nejčastěji navštěvuji)

**Cookies**

Má každá jiný token, dobu expirace, zjistíme, kdo se přihlásil – proto používáme cookies = osobní preferece

Slouží k ukládání preferencí uživatele

Uložení: v prohlížeči uživatele či na serveru. Cache jsou ukládány jen v prohlížeči uživatele.

**Cache** je úložiště pro data, aby stránka fungovala co nejrychleji, změna funkcionality se projeví až po aktualizaci v daném browseru

Cache z pohledu testování – nezrušíš, nevymažeš cache, nevíš, zda vidíš aktuální stránku

V Network/Application,Local Storage,Session Storage

Nesmí být vidět heslo, číslo kreditky-to je bug?

Cache CTRL F5 – obnovení nové stránky v browseru

Cache na různé věci na browseru, musí se vymazat, aby se nová verze znovuobnovila

Cookies mají dočasnou, trvalou nebo automaticko expiraci. Cookies zabírá méně místa v porovnání s mezipamětí Cache

Odsouhlasování technického a funkčního řazení

Malé množství dat

Data posílá nazpět serveru, může vás identifikovat dle vašich preferencí, odlišit jednotlivé uživatele

**K čemu slouží:**

**Pomáhají stránkám odlišit jednotlivé uživatele**

**Uložení potvrzení, že je uživatel přihlášený**

**Usnadňuje personalizaci**

Analytické- jak dlouho na dané stránce vydržíte, odkud se uživatel na stránku proklikl

Typy cookies

Session cookie: pouze k identifikaci v rámci jedné konkrétní návštevy webu. Pak se smaže

Persistent cookie – uloží se na disk a zůstane tam i po zavření prohlížeče. Mají nastavené datum expirace, po datu se vymažou

Secure cookie:ukládají se v rámci šifrované informace komunikace

Typy cookies dle původu scriptu

Uložena pomocí skriptu běžícího na doméně z adresního řádku

Jsou považovány za bezpečnější

Third party cookies

Vytvořeny na jiné doméně, než na které se nacházíte

Stejnou cookie proto lze načítat na různých místech internetu a uživatele tím sledovat napříc doménami

* Cílené na marketing, nabízejí weby dle našich priorit a nabízených stránek
* **Zapamtuje si daná cookies - Nastavení jazyka, zboží v košíku,přihlášení**

Jsou cookies bezpečné

Pro počítač nepředstavují nebezpečí, z pohledu ochrany soukromí,ale může být situace odlišná

Nepředstavují bezprostřední nebezpečí, je těžké být ve virtuálním světe anonymní

Kde nastavení nalezneme

* Například v Google Chrome: Menu (tři tečky v pravém horním rohu) > Settings > Cookies and other site data

Máme možnost blokovat cookies třetích stran nebo smazat cookies poté, co zavřeme prohlížeč

Cookies testování

Pokud nechci ukládat cookies, můžu vypnout ukládání a nastavení prohlížeče

Použít incognito mode a po testování cookies vymazat

**Cache**

Mezipamět, vyrovnávací pamět

Týká se dat:obrázky, styly

Moderni browsery používají vlastní **cache** . Mezipamět nastavení, co se týče promazávání

Lze ručně mazat, může způsobit drobné nedostatky (nové logo)- neuvidím nejaktuálnější verzi stránek – dále rozhozeno formátování, rozbité skripty, nesprávné obrázky

* Jak vymazat cache Například v Google Chrome: CTRL + SHIFT + DEL > zaškrtnout Cached images and files > Clear data

CTRL plus F5 reload – vypnutí refresh serveru

Extensions a ADD - ONS: upravují a rozšiřují funkci web.prohlížečů

Jaká rozšíření používají – zeptat se senior testerů v nov práci

Např.AddBlock- blokovat reklamy na internetu

Ne všchny bezpečné, škodlivé, jistá forma malweru

**Cache**

Slouží k ukládání informaci o HTML,CSS, obrázku do prohlížeče, aby se zlepšila rychlost načítání stránek. Doba expirace nastavena ručně nebo řízena serverem.

Developer tools

Sada nástrojů umožňující více prozkoumat stránky z pohledu kódu, síťového provozu

Smoke testy- zahořovací

Po každé změně kódu

Nový built, co má naznačit, zda aplikace funguje správně (vypadá schopně či ne) = běžné vyhodnocení, fungují nejzákladnější funkcionality

Pokrýváme všechny části aplikace do úzké úrovně

Hodnocení rizika (kam patří?) – pravděpodobnost a míra dopadu. Opakované Testování určujě pravděpodobnost výskytu rizika. Testování sleduje procesní, projektová a produktová rizika.

Sanity testy - lokální check

Podrobnější, část testu běží, kde došlo k ubgraedu (ne na celém systému), měří funkcionality (konkrétní změny), nedokumentují se , nejsou na to TC

* Cílené na některé části systému

**E2E- testování napříč celou službou produktu**

Simuluje chování reálného uživatele, nejméně testů, nevhodné k automatizaci

**USER Acceptance testing ?**

Na konci beta testování, zákazník zkusí aplikaci, předáno zákazníkovi, mělo by to už dobře fungovat

**Stress testing** – co se stane, když překročíte hranice možného testování (MAX 100 uživatelů), bude 150, server bude mít stres

Testuje zátěž – najít hranici, limity, kde systém spadne

Zjistit, zda se dá vrátit do normálu

**Load testing** – pod hranicí

**Performance testy** – měříme výkon

**Jiný pohled**

Cookoliv existuje, předpoklad, že by se mohlo chovat jinak a potřebuje přetestování, aktualizované úseky, testing starších věcí, které se dříve netestovaly, testování jiným způsobem

**Test strategy**

* Živá dohoda o tom, proč, jak a kdy budeme všichni společně přistupovat k testování a často nejen k testování, ale I ke kvalitativní analýze

1. **Analytické**- rozebíráme, jdeme do detailu (požadavku, rizika)
2. **Model** – based – sleduje přechody stavů
3. Methodical – tvoření systémů, definujeme dopředu set testovacíh podmínek
4. **Process-Compliant**- vývoj specifických vzorků
5. **Directed** – rada, průvodce nebo instrukce pro zůčastněné strany
6. **Regression** – averse – vytvářet testy, které budeme moci znovu a efektivně využívat
7. **Reactive**- spoléhající se na to, že se budou vyvíjet v rámci testování /exploratory testing

Testovací strategie má dlouhodobou životnost, strategie firmy, společnosti

**3 perspektivy testovací strategie**

**Rozhodnutí:** jak budu/podle jakého klíče vyhodnocovat severitu bugu

Definice bugu, definice procesu

Set praktik – které bude zajišťovat předcházení duplicit

Co budeme používat a co ne, rozdělení složek a odsouhlasení best practices, jaká sada se bude automatizovat, co vybere do regresního testování, testovací strategie

Komunikační strategie

* Ptejte se na otázky, na které znáte odpovědi, aby k odpovědi došli I ostatní, potvrzení vašeho chápání

Testovací strategie – širší, high level, více informací, na začítku procesu testování

-oblast myšlení

Nástroje, kde se budou shromažďovat požadavky, zda automatizovat

* Postup, zaměření, technicky
* Soubor pravidel, doporučení – co budeme testovat, druhy testování a zda se vyplatí automatizace

Test plán= Logistic

Co budeme testovat a jak

Přehled, hloubka, architektura aplikace

Reporting (metriky) – Reporty poskytuje například Test Manager/Test Lead

Reporty obsahují: počet nalezených bugů, test coverage – kolik funkcionalit pokryto testy, automatizované testy v %

-specifičtější, cíle (high level), živý dokument, prostředí

- design, které testy automatizovat a které ne

Slouží k identifikaci základních milníků v testování

Vztahuje se ke konkrétnímu projektu

Plánování je cesta, ne úkol

Je to závazek a předpoklad = živá dohoda

Plán/ model nebyl dobrý – není to ale realita, ale poučení, užitečné

Road mapa

Př. Zahrnuje Security testy, integrační testy, E2E testy

Reporting

* Poskytujeme informace
* Reporty o průběhu
* Postupu testování

**Summary report** – podáváme informace na konci sprintu Test led/Delivery manager (komu předává zákazníkovi)

**Summary report pro Stakeholdery**

Metriky – jak dlouho trvá exekuce automatizovaných testů, kolik bugů

Rizika, která se mohou objevit. Další kroky pro příští sprint. DOPORUČENÍ. Info o tom, co se dělo.

Každý report je připravován pro konkrétního stakeholdera, doručujeme očekávanou informaci, nejefektivnějším způsobem, adresujeme emoci a očekáváme pocity, s cílem spoušťet akci

Předáváme Počty a Context testování

Popisují očekávané výsledky Expected/Efficient/emoce (pozitivní I negativní) a očekáváme akci – lepší plánování, výsledky a dopad.

Míra důležitosti defektu, informuje, zda se problém řeší či ne.

I-Doručit očekávanou informaci – definovaná metrika

II -jakou formou budeme metriky efektivně popisovat, prezentovat(graf, tabulka, časová osa)

III- emoce

IV – Impact, odhadovaný, měřitelný, hodnotitelný

Co je to Defekt priority v analýze

Příčina- analýza kořenové příčiny

K čemu slouží 5 proč : WHY

Formal post mortem/formální analýza – incident summary, k čemu vedlo a co bylo za problém

Samostatná složka: fault, timeline, rootcause, incident summary

Cloud je síť vzájemně propojených vzdálených serverů, která slouží převážně k úschově a sdílení dat nebo užívání aplikací a softwaru. Na rozdíl od klasického pevného disku ve vašem počítači nejsou data fyzicky uložena v přístroji, nýbrž právě v cloudu

**API=Application Programming Interface**

* Api funguje správně, jen když máme všechna potřebná data pro podání požadavku
* Browser dostal všechny potřebné informace, frontend to pak správně zobrazí na stránce
* Někdy může být chyba na frontendu (nepošle nám zpět správný soubor informaci) – bug, kontaktovat developera, např. na UI je 8 místo 10 receptů
* Rozhraní pro komunikaci se softwerem, testujeme když není UI
* Testujeme jen beckend přes API
* Zadat dotaz ve správném formátu, dostaneme přesně definovanou odpověď

Proč testovat na úrovni API- nepotřebujeme k němu funkční UI

Můžeme zkusit více možností testování, zlepšit security stránek

* Způsob, jak zjistit, zda problém na frontendu či beckendu
* Investigace= vyšetření bugu, na jaké vrstvě je

SOAP API- verze před Rest API, starší verze, moc se nepoužívá

REST API = Represantional State Transfer (speciální druh API – podmnožina)

Co je **JavaScript**- je objektově orientovaný scriptovací jazyk, figuruje na straně klienta v prohlížeči

* Vytváří logiku stránky a tvoří příjemnější, uživatelské rozhraní
* Programy psané v javascriptu se nazývají skripty a zapisují se přímo do HTML kódu
* Je populární pro dněšní weby díky javascriptovým frameworkům (Angular,React)
* JavaScript – do HTML nebo webové stránky přidává nějaké chování – co se stane když se klikne na tlačítko, když uživatel vyplní špatně formulář
* Díky JS dáváme aplikaci dynamičnost, pohyb a funkci
* Dělá logiku stránky, validace, kontrola dat, hlášky programovací jazyk - pomocí něhož stránky komunikují s (backendem, databází. ...) dokáže hlídat obsah vkládaných dat (validace hesla, mailu...)

**Výhody:** svižnost – server není zbytečně zatěžován

Př. Klient (uživatel) odešle požadavek na server, server zpět zašle HTML a skript, který je následně zpracován prohlížečem. PHP, kdy stránka přichází stažená již ze serveru

Plná integrace s HTML a CSS

Rychlejší interakce a dynamické načítání stránky

Př.vyhledávání v prohlížeči omužnujě právě Javascript

-různorodost frameworků

-databíze API – která umožnujě do aplikace interagovat funkcionalitu nástrojů třetích stran

**Nevýhoda:** potencionálně možné zneužití kódu, který se běžně zobrazuje v prohlížeči vedle HTML – nízké riziko, existují techniky na ošetření,aby se to nestávalo

* Riziko nekompatibility nebo spíš zákazu Javasriptu v prohlížeči, v důsledku čehož by se stránka nezobrazovala správně – nízké riziko
* -absence možnosti přístupu a přepisu jiných dat, než jsou [cookies](https://www.rascasone.com/cs/blog/co-jsou-cookie-soubory-lista).
* Zablokování zobrazování JavaSriptu uživatelem v prohlížeči /Ř: Navrhnout stránku tak aby mohla fungovat i bez použití Javasriptu

Auto generated headers dělá postman sám od sebe, vyplní údaje do payload

Headers pomůže pochopit serveru, co on dostal a v jakém formátu to je (velikost, co je uvnitř)

Důležité pro nás vědět, kdy se vyplňuje Authorization (Bearer token) – token

Když není Authorizace, v headers chybí.

Co je v Headers – když něco chybí v headers a je v dokumentaci, tak doplnit

Body/payload obsahuje headers níže

1.**Content type** -Json,Xml

2.**Content length** – jak dlouhý řetezec , množství

3.**Authorization** – kdo jste MA/A,rodič

4.**Host** –adresát zásilky

5.**User Agent** – Postman či jiná použitá aplikace použita na request

6.**Accept –** v jakém formátu přijímáš zprávu Json, Xml

7.**Accept Encoding** - nedůležité

8.**Connection** – nedůležité

**Request** součástí je body u get metody ne, enpoind,metoda,

V jakém formátu posíláme data? V request body – většinou jason 99%, někdy XML – starší verze Api SOAP

**Response** součastí – Status code součást, ale nemusí být body, pokud body tak ve formátu jason nebo xml

* **REST API Architektura pro webové API**
* Umožňuje jednotný přístup ke zdrojům, datům

**Výhoda:** nezáleží na programovacím jazyce, jak je napsán server, nemusíme vědět, jakým způsobem kontaktuje server, použitelné v jakékoliv aplikaci, důležité je znát strukturu API

**Postman =** aplikace, která slouží k interakci s API

* Slouží jako klient vytvářející requesty na API
* Umožňuje ukládat requesty a organizovat je
* Nejčastěji se používá v REST api jason format, umí jakýkoliv jazyk
* API testování = grey box (vidíme částečně pod kapotu, jak to funguje uprostřed)
* Pokud API není dobrá dokumentace, je to bug

**GIT = Version control systém (VCS)**

Systém, který nám pomáhá sledovat a řídit změny v souborech, nepotřebujeme internet, beží na našem počítači

GIT HUB = jednoduší na práci mezi lidmi, umísťujeme GIT do Cloudu, musíte se přihlásit, sdílí práci vytvořenou v gitu

**Alfa testing** – server v práci, na který nahrajeme aplikaci, Poskytneme heslo/jméno zákazníkovi. Podepíše nám papír, že dál nebude šířit. Zákazník nebo někdo z firmy/zástupce zákazníka dá po sprintu zpětnou vazbu.

**Beta testování**- blíž k zákazníkovi, nahraje si aplikaci na svůj server/telefon

Lepší kvalita aplikace, těsně před releasem

SQL- potřebuje najít data v databázi, aby správně pochopit data

Př.White box testing

Výhody oproti excelu – lepší propojení, více lidí zárověň, lepší se čistí data, přehlednost, rozah, rychlost, integrita, bezpečnost

Jaké datové typy: string, integar, - kolem 10ky, graf

Vazby mezi tabulkami: 1:N zákazník má několik objednávek,Objednávka má jen jednoho zákazníka

M:N – objednávka versus produkt. V jedné objednávce, může být více produktů.Jeden produkt může být ve více objednávkách

Primární klíč – jednoznačně určuje unikátní řádek v tabulce

Cizí klíč- odkazuje na primární klíč jiné tabulky