Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Srovnání testovacích nástrojů

Bakalářská práce

Autor: Michael Šeda
Studijní obor: Informační management

Vedoucí práce: doc. Mgr. Tomáš Kozel, Ph.D.
Odborný konzultant: Ing. Miroslav Ježek, Unicorn Systems

Hradec Králové Srpen, 2023

Prohlášení:

Prohlašuji, že jsem bakalářskou/diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 01.08.2023 Michael Šeda

Poděkování:

Děkuji vedoucímu bakalářské práce doc. Mgr. Tomášovi Kozlovi, Ph.D. za metodické vedení práce a Ing. Miroslavovi Ježkovi ze společnosti Unicorn Systems a.s. za odborný dohled a užitečné rady.

Anotace

Cílem této bakalářské práce je poskytnout detailní rozbor výhod a nevýhod programů určených pro automatické testování softwaru a programů souvisejících s automatickým testováním skrze editaci dat, zaznamenávání stavu testování a exportu výsledků testování. Výsledek této práce nebude využitelný pouze pro testery, ale může také posloužit manažerům projektů, jelikož bude zahrnovat i finanční srovnání. Při hodnocení programů bude kladen důraz především na intuitivnost programu a jeho náročnost ho používat, jelikož testeři často bývají nejvíce juniorní typ zaměstnanců v IT.

Hlavními objekty porovnání budou programy pro manuální testování, tvorbu automatických testů, textové editory a nástroje pro správu testování softwaru a následný export výsledků testování.

Annotation

Title: Test tool comparison

The aim of this bachelor thesis is to provide a detailed analysis of the advantages and disadvantages of programs designed for automated software testing and programs related to automated testing through data editing, recording test status and exporting test results. The result of this thesis will not only be useful for testers but can also serve project managers as it will include a financial comparison. In evaluating the programs, the focus will be on the intuitiveness of the program and how difficult it is to use, as testers are often the most junior type of employees in IT.

The main objects of comparison will be programs for manual testing, creation of automated tests, text editors, tools for test management and the subsequent export of test results.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Téma Srovnání testovacích nástrojů jsem si zvolil pro svou bakalářskou práci z důvodu, že se jedná o téma aktuální jak ve světě, tak i v mém kariérním životě, jelikož je testování jednou z hlavních náplní mé práce.

Existuje již mnoho testovacích nástrojů, ze kterých si uživatel může vybírat a mou snahou je tento proces rozhodování zjednodušit skrze srovnání programů na základě mých zkušeností.

2 Cíl práce

Cílem této práce je poskytnutí informací o testování softwaru, představení konkrétních programů pro testování, prezentaci jejich benefitů a záporů, jejich následné srovnání a doporučení. Dále je účelem demonstrovat využití těchto programů v praxi skrze simulaci pracovního dne testera během které bude nastíněno využití programů, které vyšly jako nejlepší ve srovnání.

2.1 Metodika zpracování hodnocení programů

Programy vybrané pro tuto bakalářskou práci, jsou rozšířené programy, které jsou využívány ve společnosti Unicorn Systems a.s., ve které pracuji, tudíž se s programy setkávám. Programy budou bodově ohodnoceny v rámci parametrů uvedených níže. Bodové hodnocení bude v tabulce. Počet bodů pro každý parametr hodnocení je 0 - 3.

2.1.1 Hodnocené parametry programů pro editaci dokumentů

- intuitivnost programu a snadnost jeho používání
- podpora - dohledatelnost návodů
- automatické doplňování
- možnost pluginů a doplňků do aplikace - například možnost přidání plugin pro formátování pro různé programovací jazyky
- integrovaný terminál/příkazový řádek

- zvýrazňování textu (dále označováno jako highlighting)
- hardwarová nenáročnost
- finanční ohodnocení - zda existuje plnohodnotná free verze a nebo jestli je nákup licence pro tým přijatelné a vyplatí se

Tabulka 1 Ukázka tabulky hodnocení programu pro editaci dokumentů

Název programu	Bodové ohodnocení
Automatické doplňování	0 - 3
Highlighting	0 - 3
Pluginy a doplňky	0 - 3
Příkazový řádek	0 - 3
Hardwarová nenáročnost	0 - 3
Intuitivnost	0 - 3
Podpora	0 - 3
Finanční ohodnocení	0 - 3

2.1.2 Hodnocené parametry programů pro manuální testování

- podpora protokolů
- typy testování - testování zabezpečení, výkonu, obsahu
- zprovoznění programu - instalace a nastavení prostředí
- hardwarová nenáročnost
- intuitivnost
- podpora
- finanční ohodnocení- zda existuje plnohodnotná bezplatná verze a nebo jestli je nákup licence pro tým přijatelné a vyplatí se

Tabulka 2 Ukázka tabulky hodnocení programu pro manuální testování

Název programu	Bodové ohodnocení
Podpora protokolů	0 - 3
Typy testování	0 - 3
Zprovoznění programu	0 - 3
Hardwarová nenáročnost	0 - 3
Intuitivnost	0 - 3
Podpora	0 - 3
Finanční ohodnocení	0 - 3

2.1.3 Hodnocené parametry programů pro tvorbu automatických testů

- intuitivnost programu a snadnost jeho používání
- podpora aserce - ověření hodnoty na daném místě
- podpora - dohledatelnost návodů
- reportování testů - jak snadno zřetelně lze sdílet výsledky testů
- možnost pluginů a doplňků do aplikace - například možnost přidání plugin pro formátování pro různé programovací jazyky
- znovupoužitelnost testů
- udržitelnost testů
- hardwarová nenáročnost
- testování výkonu
- finanční ohodnocení

Tabulka 3 Ukázka tabulky hodnocení programu pro tvorbu automatických testů

Název programu	Bodové ohodnocení
Reportování testů	0 - 3
Podpora aserce	0 - 3
Znovupoužitelnost testů	0 - 3
Pluginy a doplňky	0 - 3
Udržitelnost testů	0 - 3
Hardwarová nenáročnost	0 - 3
Testování výkonu	0 - 3
Intuitivnost	0 - 3
Podpora	0 - 3
Finanční ohodnocení	0 - 3

2.1.4 Hodnocené parametry test management tool programů

- intuitivnost programu a snadnost jeho používání
- plánování testů a organizace
- verzování a historie
- zabezpečení dat
- podpora - dohledatelnost návodů
- možnost pluginů a doplňků do aplikace - například možnost přidání plugin pro formátování pro různé programovací jazyky
- finanční ohodnocení

Tabulka 4 Ukázka tabulky hodnocení test management tool programu

Název programu	Bodové ohodnocení
Plánování a organizace testů	0 - 3
Verzování a historie	0 - 3
Zabezpeční dat	0 - 3
Pluginy a doplňky	0 - 3
Intuitivnost	0 - 3
Podpora	0 - 3
Finanční ohodnocení	0 - 3

3 Teoretická část

3.1 Úvod do teoretické části

Testování je jednou z nejzákladnějších částí vývoje softwaru. Hlavním účelem testování je poskytnutí informací vedení projektu o stavu aplikace, tj. které funkcionality jsou aktuálně otestované a zda-li jsou funkční či ne, dále zkontrolovat přítomnost, počet chyb a jejich závažnost [5].

Hlavními výhodami zakomponování testování do vývoje softwaru je zvýšená efektivita vývoje, předcházení chyb a jejich oprava. Samozřejmě při testování projekt potřebuje navíc testery a čas vykázaný pro jejich práci. Pomocí využití automatického testování lze redukovat potřebný čas. Automatické testy je však potřeba nadále udržovat a vyhodnocovat, viz kapitola 3. 4 Automatické testování.

Testování lze rozdělit podle míry přítomnosti testera na testování manuální a automatické. V manuálním testování tester osobně vykonává všechny testovací akce při každém průchodu testu, a to už od založení testovacího designu podle pročtení dokumentace požadavků a specifikací, založení testovacích případů na základě vytvořených testovacích scénářů, až po následný export výsledků testů. V automatickém testování jsou testy procházeny kontinuálně a automaticky skrze aplikace, ve kterých je automatický test vytvořen - tester deklaruje operace, které má systém vykonat, hodnoty, které má používat, a očekávané výsledky či reakce na výsledky neočekávané. Automatické testy pak mohou být nahrány na cloud, kde je může vidět celý tým a mohou zde být průběžně spouštěny, viz TeamCity v kapitole 3.4 Automatické testování.

Testování pak lze dále dělit na různé typy na základě jejich účelů [28] – jednotkové, integrační testování, funkční testing (testování funkcionality aplikace), systémové testování, akceptační testování, výkonové testování, bezpečnostní testování a smoke testing. Všechny tyto typy testování budou detailně popsány.

3.2 Modely fází cyklu vývoje softwaru

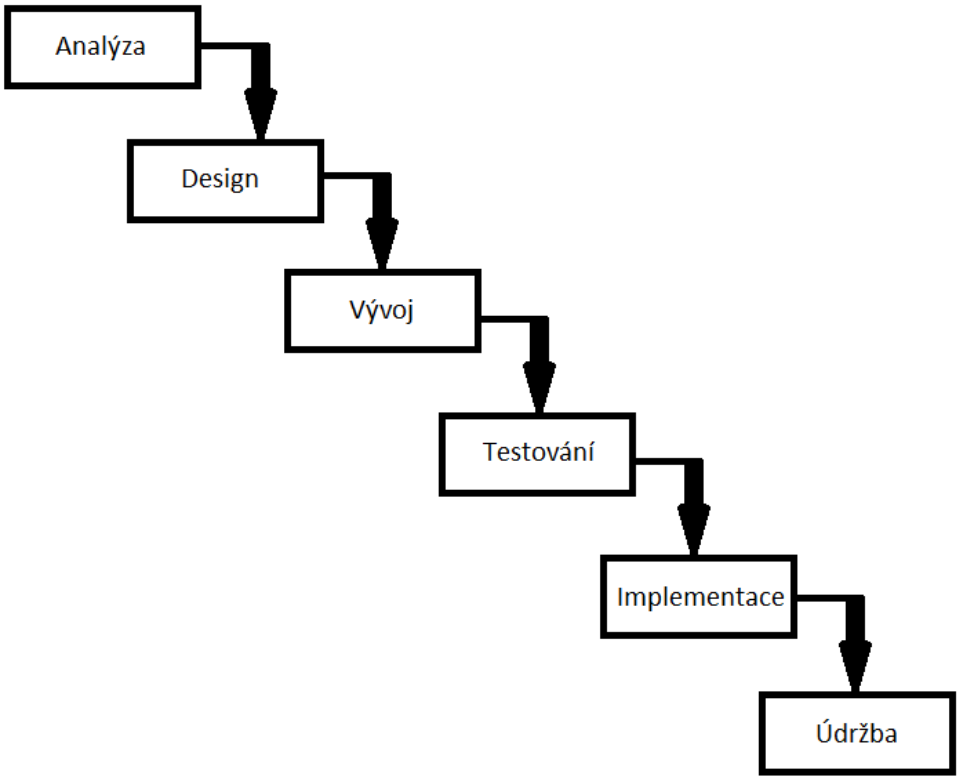
V následující části jsou vysvětleny fáze, které tvoří životní cyklus vývoje softwaru (SDLC = fáze vývoje informačního systému) a které zahrnují plánování, analýzu, návrh a implementaci. Byly vyvinuty modely životního cyklu vývoje softwaru (SDLC), jako

je vodopád, spirála, V-model, rychlé prototypování, inkrementální a synchronizace a stabilizace [16].

3.2.1 Vodopádový model

Při vodopádovém modelu je v první řadě potřeba definovat požadavky. Jakmile je kód kompletně napsán, následuje testování. Než se přejde k dalšímu kroku, je každý pracovní produkt nebo činnost dokončena [22].

Jednotlivé fáze vývoje probíhají postupně bez překrývání. Každá fáze obsahuje plán úkolů, které mají být provedeny za určitý čas. Na závěr každé etapy se provádí dokumentace a testování, které pomáhají zachovat kvalitu projektu. Každý krok ve vodopádovém modelu se zastaví před krokem následujícím. Jinými slovy, požadavky jsou stanoveny před zahájením procesu návrhu a jakmile je stanoven návrh, začíná proces kódování atd. Přesto je práce testovacího týmu do té doby nesmírně časově a finančně náročná. Jelikož není testovací tým zapojen od začátku projektu, problém může být objeven až v extrémně pozdní fázi životního cyklu vývoje podle vodopádového modelu.



Obrázek 1: Vodopádový model

Zdroj obrázku: Vlastní zpracování na základě [22]

Výhody vodopádového modelu [22]:

- Požadavky jsou jasné před startem vývoje
- Každá fáze je časově vymezená
- Jednoduchá implementace

Nevýhody vodopádového modelu [22]:

- Jelikož jsou testeři přítomni až v průběhu fáze testování, nelze předčasně předcházet chybám v kódu a implementaci
- Nalezená chyba může ovlivňovat více komponent najednou a její oprava může být velmi komplexní

3.2.2 Agilní model

Při postupování skrze agilní přístup je využívána průběžná komunikace se zákazníkem. Projekt je rozdělen na časové úseky [22]. Každý úsek zahrnuje aktualizovanou dokumentaci s analýzou a požadavky s kontrolou od zákazníka. Během každého úseku je vytvořena jedna nebo více komponent a zároveň otestována. V agilním modelu jsou testeři přítomni od začátku.

Obrázek 2: Agilní model

Zdroj: [17]

Výhody agilního modelu [22]:

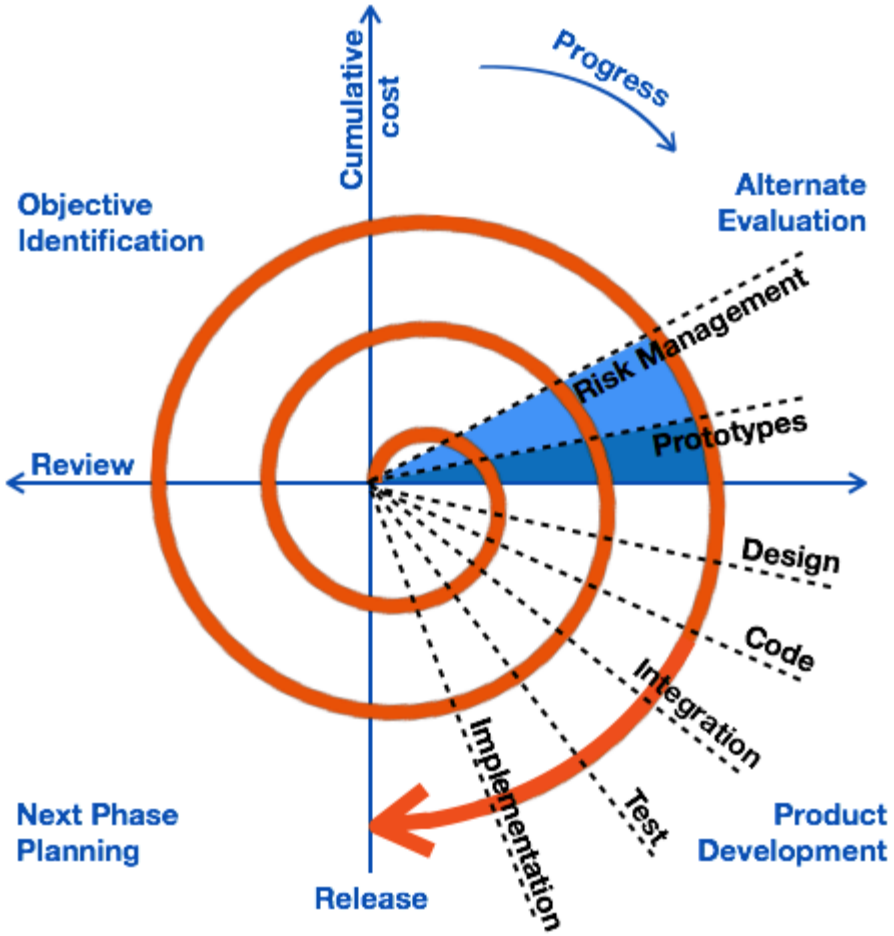
- průběžná komunikace se zákazníkem
- možnost průběžné aktualizace požadavků a specifikací
- zákazník může od první implementace využívat aplikaci, včetně následujících verzí, ve kterých jsou přidány nové funkcionality
- chyby jsou odhalovány průběžně

Nevýhody agilního modelu [22]:

- může být náročné pro velké projekty kvůli složitější koordinaci

3.2.3 Model spirály

Model spirály kombinuje model vodopádu a iterativní model (popsán níže). Je založen na fázích vývoje projektu, které se opakují pro každou novou verzi aplikace [23]. Hlavními fázemi jsou identifikace cílů, analýza risků, vývoj a závěrečná revize verze a plánování verze další.



Obrázek 3: Spirálový model

Zdroj: [20]

Výhody spirálového modelu [23]:

- Flexibilita: Během využití spirálového modelu lze průběžně měnit požadavky a specifikace projektu.

- Zpětná vazba: Spirální model využívá kontinuální komunikace se zákazníkem, díky čemuž je přítomná častá zpětná vazba pro tým od zákazníka.
- Brzká detekce chyb: Testerský tým je přítomný napříč celým vývojem, tudíž dochází k včasné identifikaci bugů.

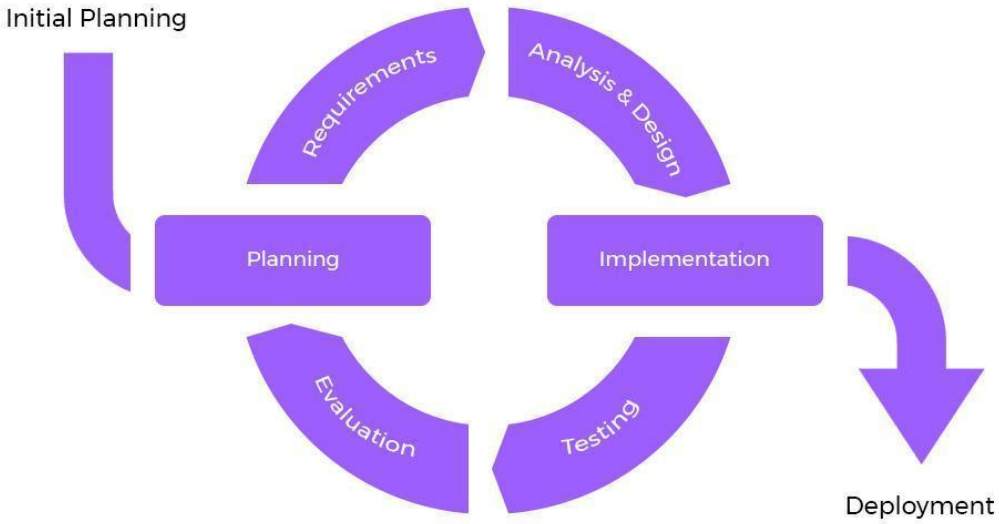
Nevýhody spirálního modelu [23]:

- Koordinačně náročný: Jelikož se jedná o komplexní model, jeho management je obtížnější než přímočařejší modely, jako je například model vodopádu.
- Vyšší požadované dovednosti: Spirální model vyžaduje využití týmu se zkušenostmi týkajících se identifikace risků.

3.2.4 Iterativní model

Iterativní model je založen na přírůstkovém vývoji softwaru. Od zákazníka přijdou základní požadavky, které management rozdělí na menší části, které jsou následně implementované. To vše je prováděno v krátkých, většinou týdenních intervalech.

Tento model je využíván například v situaci, kdy zákazník sám ještě netuší, jak by finální produkt měl vypadat nebo v situaci, kdy se očekává, že legislativa ovlivňující projekt se bude do budoucna měnit.



Obrázek 4: Iterativní model
Zdroj: [21]

Výhody iterativního modelu:

- Průběžné dodávání - jelikož je produkt rozdělen do menších částí podle verzí, které samy o sobě fungují, verze jsou průběžně dodávány zákazníkovi, který je už může využívat pro své účely. Aplikace má tedy hodnotu téměř od začátku vývoje.
- Flexibilita - vývoj rozdělený do samostatných částí umožňuje snadnější aktualizaci požadavků od zákazníka
- Zpětná vazba od zákazníka

Nevýhody iterativního modelu:

- Nekompletní definice finálního produktu - během vývoje produktu se může stát, že již existující požadavky a specifikace, které již byly do určité míry implementovány musí být zpětně upraveny

- Náročné pro management - modely rozdělené do menších celků bývají obecně složitější pro manažery projektu

3.3 Manuální testování

3.3.1 Box testing

Vyvíjenou aplikaci si můžeme představit jako objekt. Objekt lze pozorovat jako samostatný povrch, kdy neznáme jeho vnitřek, a tak popisujeme a pracujeme pouze s jeho zevnějškem (externí částí). V momentě, kdy známe i jeho obsah (interní strukturu kódu), můžeme pozorovat i jeho vnitřní prostředí.

Podle tohoto se testování dělí na black box testing, white box testing a grey box testing podle toho s jakou znalostí objekt testujeme.

Během **black box** testování [25] pracujeme pouze se vstupy a výstupy, bez toho abychom se zabývali interní strukturou aplikace (kódem).

Tester tedy navrhuje testovací případy podle požadavků a specifikace aplikace, se kterými se setká uživatel. Black box testing také nebývá tolik náročný, jelikož při něm většinou není potřeba znalost programování, pokud se tedy nejedná o automatický test. Tester pracuje pouze s vnější částí aplikace a pracuje prostřednictvím vkládání hodnot, interakcí s komponentami a sledování reakcí systému. Black box testing je tedy vhodný pro juniorní testery.

White box testování [26] naopak spočívá v práci s kódem, tester tedy musí znát programovací jazyk programu a jeho strukturu.

Při white box testování tester například zasílá požadavky a kontroluje odpovědi programu. Během tohoto může předčasně odhalit chyby v implementaci nových funkcionalit a může nalézt důvody sníženého výkonu (performance) programu.

Grey box testing [27] spočívá s kombinací obou black a white box testingu. Tester může například testovat tlačítko pro odeslání formuláře a zjistí, že tlačítko nereaguje na akci. Pokud je tester dostatečně informován o programovacím jazyce a struktuře programu, může přejít do části s kódem, upravovat ho a při tom synchronně sledovat změny v aplikaci.

Synchronní chování = uživatel čeká na reakci aplikace na request a poté může pokračovat (stisknutí tlačítka přihlásit)

Asynchronní chování = uživatel může dále provádět různé akce nezávisle na odeslaném requestu

3.3.2 Způsoby testování vstupů a výstupů

Equivalence class partitioning

Při testování stylem equivalence class partitioning se postupuje tak, že vstupní data jsou rozdělena do ekvivalentních skupin. Díky tomu je možné místo individuálního testování testovat skupiny dat. [2]

Díky tomuto odlišení lze rozlišit na data, která jsou schopna způsobit chybu a mají tedy větší význam pro testování a data která jsou méně schopna způsobit chybu a tudíž nemají takový význam pro testování.

Tento typ testování tedy zvyšuje účinnost testování a časovou efektivitu.

Boundary value analysis

Jak již z názvu vyplývá, při tomto typu testování jsou kontrolovány okrajové hodnoty. Tento styl testování se používá za účelem testování extrémních situací, ve kterých se pohybují vstupní hodnoty a parametry na okrajích definované množiny [24].

Pro představu dejme, že máme input pro telefonní číslo - budeme tedy testovat hodnoty, které jsou nižší jak 9 cifer a hodnoty, které jsou vyšší jak 9 cifer.

Díky tomuto testování můžeme odhalit snadno přehlédnutelné chyby.

Tabulka 5 Boundary value analysis

Počet cifer	Výsledek
<9	E
9	S
>9	E

Legenda tabulky:
E - error (zobrazí se validační hláška - vložené hodnoty jsou invalidní)
S - success (zadané telefonní číslo je úspěšně uloženo)

V praxi bychom pak tedy měli 3 různé testy:
Test 1 (AF)*: Zadání hodnoty s počtem cifer menším jak 9 = error
Test 2 (HD)*: Zadání hodnoty s počtem cifer 9 = error
Test 3 (AF): Zadání hodnoty s počtem cifer větším jak 9 = error
*AF = alternative flow - označení pro alternativní test
*HD = happy day - označení pro test při, kterém je otestován správný chod

Decision table based testing

Decision table based testing se řadí mezi black box [4] (také popsáno v předchozích kapitolách) typ testování při kterém je aplikace testována bez nahlížení do jejích vnitřních částí.

Během decision table based testingu se definují všechny možné kombinace inputů a outputů a jejich výsledky. [3]

Příklad:

Máme modální okno pro přihlášení do systémů, které obsahuje dvě inputová pole - uživatelské jméno (username), heslo (password) a tlačítko přihlásit se.

Tabulka 6 Decision Table Testing

Podmínky	Podmínka 1	Podmínka 2	Podmínka 3	Podmínka 4
Username	F	T	F	T
Password	F	F	T	T
Output	E	E	E	S

Legenda tabulky:

T - true (vyplněnou validní hodnotou)

F - false (vyplnění invalidní hodnotou)

E - error (zobrazí se validační hláška - vložené hodnoty jsou invalidní)

S - success (uživatel je úspěšně přihlášen do aplikace)

Tester tedy vytvoří 4 různé testy:

Test 1 (AF)[1]: Zadání invalidní hodnoty uživatelského jména a hesla - error

Test 2 (AF): Zadání validního uživatelské jména a invalidního hesla - error

Test 3 (AF): Zadání invalidního uživatelské jména a validního hesla - error

Test 4 (HD)[2]: Zadání validního uživatelského jména a hesla - uživatel je úspěšně přihlášen

State transition

State transition testing, stejně jako předchozí typy manuálního testování spadá do black box testování. Tento typ testování je využíván za účelem kontroly úspěšného přechodu aplikace z jednoho stavu do druhého.

Jako první se definují stavy aplikace, dále se definuje, který stav může přejít do jiného a nakonec se definují akce nutné pro zahájení změny stavu aplikace.

Existující a používané typy stavu aplikace:

- non-running = stav při kterém aplikace není spuštěna a neběží

[1] Alternate flow - termín, který v testování označuje alternativní scénář

[2] Happy day - termín, který v testování označuje kompletně úspěšný scénář

- inactive = aplikace běží, ale po určitou dobu nepodléhá žádné interakci s uživatelem
- active = aplikace běží a podléhá aktivní interakci s uživatelem
- background = aplikace běží na pozadí a vykonává procesy
- suspended = aplikace běží na pozadí bez vykonávání procesů

Například budeme mít přihlášeného uživatele v aplikaci, jehož stav je active a může přejít do idle (inactive), pokud po určitou dobu nebude vykonávat akce v aplikaci. Máme nastavený odpočet například na 5 minut - jakmile uživatel nevykonal po dobu pěti minut žádnou akci - přejde do stavu idle.

Error guessing

Pojem error guessing označuje řešení vyskytnutého problému na základě osobních zkušeností testera z aktuálního projektu či předešlých projektů [5]. Postup při testování na základě předešlých zkušeností je těžko popsatelný, jelikož funguje na subjektivních postupech testera a je využíván ad hoc.

3.3.3 Fáze testování

Testování lze rozdělit do několika fází. Těmito fázemi je dle [6] statické testování, dále také jednotkové testování [7], integrační testování [8] a systémové testování [9], [10]. Všechny tyto fáze jsou popsány detailněji v kapitolách níže.

Statické testování

První fází testování je takzvaný static testing, během kterého tester předběžně kontroluje dokumentaci na requirement analysis, aby se zajistil jejich správnou funkčnost a jejich následnou integraci do systému [6]. Tester také průběžně komunikuje s analytiky v případě, že se domnívá, že by měla být aktualizována dokumentace. V této fázi tedy ještě neprobíhá testování přímo programu. Tato fáze je přítomná i během dalších, jelikož nastávají situace, kdy se dokumentace musí zpětně upravovat, jelikož může nastat změna ad hoc.

Jednotkové testování

Další, druhou fází, je takzvaný unit testing. Testování jednotek je technika testování softwaru, která se zaměřuje na testování jednotlivých jednotek nebo komponent softwarového systému izolovaně od zbytku systému. Cílem testování jednotek je zajistit, aby každá jednotka nebo modul softwaru fungoval podle očekávání a splňoval požadavky systému. [7]

Jedná se o nezbytnou část vývoje softwaru, protože pomáhá zachytit chyby v rané fázi vývojového cyklu. Díky izolovanému testování jednotlivých jednotek kódu mohou vývojáři rychle identifikovat a opravit problémy dříve, než se rozšíří do dalších částí systému. To vede k rychlejším vývojovým cyklům, kvalitnějšímu kódu a spolehlivějším softwarovým systémům.

Rámce (frameworky)[3] pro testování jednotek, jako je JUnit pro Javu nebo NUnit pro .NET, poskytují vývojářům nástroje pro efektivní vytváření a spouštění testů jednotek. Tyto rámce se obvykle integrují s vývojovými prostředími, jako je Eclipse nebo Visual Studio, a poskytují funkce, jako je správa testovacích případů, vytváření zpráv a ladění.

Tester podle dokumentace připraví test designy a test casy, které nejdříve projde vývojář při implementaci komponenty a poté po její nasazení je znovu otestuje tester a označí jako úspěšné či neúspěšné.

Unit testing by měl být prováděn často v průběhu celého procesu vývoje. V agilním vývojovém prostředí se jednotkové testy obvykle spouštějí po každé změně kódu, aby se zajistilo, že se kód stále chová podle očekávání. Unit testy lze také spouštět automaticky v rámci procesu kontinuální integrace, který zahrnuje automatické sestavení, testování a nasazení kódu při každé jeho změně.

Integration testing

Třetí fází testování je integration testing, který se dělá na dva typy:

- Integrační testing mezi různými systémy - testování kolaborace vyvíjené aplikace s jiným systémem či službou. Pro představu může být vyvíjená aplikace pro e-shop, která bude využívat externí službu platební brány.

[3] více rozvedeno v 3. 7 Frameworky pro Unit Testing

- Integrační testing komponent aplikace - při kterém se testuje “kolaborace” celkového systému aplikace s novou implementací [8].

Integrační testování se obvykle provádí po jednotkovém testování, kdy se jednotlivé jednotky nebo moduly testují izolovaně.

Způsoby integračního testování

Integrační testování lze dělit do těchto přístupů:

- Top-down: Při tomto přístupu se začíná testováním od modulů nejvyšší úrovně až po moduly úrovně nejnižší. Díky této metodě lze předčasně identifikovat vzniklé chyby a následné je izolovat a opravit [13].
- Bottom-up: Opakem testování metodou top-down je způsob bottom-up. Tímto způsobem začíná identifikace chyb u modulů nejnižších až po moduly nejvyšší
- Big-bang integration testing: Riskantní přístup k testování, který testuje celý systém najednou. Nevýhodou může být náročná izolace chyb [14].
- Hybrid: Hybridní přístup využívá kombinace metody top-down i metody bottom-up. Některé části aplikace jsou testování od nejnižších komponent po nejvyšší a obráceně.

Pro představu můžeme mít dvě existující komponenty - komponentu pro nahrání zprávy a nově přidáme komponentu pro registr zpráv, ve kterém jsou vidět všechny příchozí či odchozí zprávy. Pokud nahrajeme zprávu v původní komponentně, zpráva by měla být úspěšně viditelná v nové komponentě registru zpráv.

System testing

Čtvrtou fází je system testing, ve kterém je poprvé testována aplikace jako celek. Tato fáze by se dala označit jako nejdůležitější, jelikož testuje komplexně finální produkt [9]. System testing obsahuje více částí zaměřených na určité oblasti:

Functional testing během kterého jsou testovány základní funkčnosti aplikace především v očekávaných případech chování uživatele, tedy především happy day testy.

Performance testing, který kontroluje rychlost systému, reakce a využití zdrojů. Tím může být například rychlost odpovědi systému na stisknutí tlačítka, nahrání či stáhnutí zprávy, start aplikace, přihlášení a další.

Security testing jehož cílem je otestovat zabezpečení dat a systému - kdo má přístup k datům, kde jsou uschována a podobně. Na co by si měl vývojářský tým dát pozor může být například využívání externích databází pro ukládání snímků obrazovky. Ty by neměly být uloženy externě, ale měly by být nahrány jako soubor přímo do test management toolu. Podobně je tomu i u dokumentace o aplikaci, která by neměla být po stažení otevírána na cloudu ale na disku.

Usability testing je založen na testování intuitivnosti systému a snadnost jeho používání. Tento typ testování je prováděn skrze sledování postupu uživatele a vymezení komponent a objektů, které mohou vzbuzovat zmatení v uživateli [10]. Výsledkem tohoto testování je pak tedy esenciální zpětná vazba od uživatele.

Compatibility testing, které slouží pro testování aplikace na různých platformách či prohlížečích. Pokud není aplikace specificky určena pouze pro předem určené platformy, předpokládá se, že by měla být použitelná na všech platformách či prohlížečích. Toto je žádoucí, jelikož použitelnost na více platformách a prohlížečích umožňuje větší škálu použitelnosti a tedy i více potenciálních zákazníků.

Moment kdy aplikace funguje například na Google Chrome a nefunguje na Safari také vzbuzuje v zákazníkovi negativní dojem.

Pro cross platform testování lze používat programy pro vyzkoušení responzivity na různých zařízeních. Jedním z nich je například Browser Stack, který umožňuje otestovat aplikaci na různých zařízeních a operačních systémech [11].

3.4 Automatické testování

Automatické testování se rozlišuje od manuálního testování tím, že je prováděno s minimální potřebou přítomnosti testera. Tester je vyžadován pouze pro vytvoření automatických testů a jejich průběžnou údržbu, kontrolu výsledků a update na základě nových změn v aplikaci. Během času, kdy běží automatické testy, může tester věnovat svou pozornost vytváření nových automatických testů nebo testování zajímavějších testovacích případů. Benefity automatického testování je pravidelná kontrola systému, pokrytí většiny funkcionalit systému.

Automatické testování se dělí na stejné typy jako manuální, viz kapitola 1.3.2 Typy testování. Další výhodou automatických testů je také to, že celkové výsledky jsou lehce přístupné celému týmu a nejen testerům, pokud jsou automatické testy spouštěny pravidelně v programu jako je například TeamCity. Do tohoto programu mívá přístup celý tým, jelikož lze skrze TeamCity uskutečňovat sestavení nových verzí a jejich následné nasazení a lze zde sledovat reálný stav těchto procesů, stejně tak jsou zde i pravidelně spouštěny testy [12].

3.5 Programy pro testování

Testování softwarových aplikací lze zjednodušit a zefektivnit pomocí automatických testovacích programů. Pomocí těchto technologií mohou vývojáři efektivněji vyhledávat chyby nebo nesrovnalosti díky automatizaci zdlouhavých operací, spouštění několika testů najednou a provádění mnoha testů. Vývojáři mohou využitím automatizačních funkcí zaručit spolehlivost a výkonnost svých softwarových řešení.

Výhody technologií automatizovaného testování [29] jsou následující:

Úspora času: Díky tomu, že automatizace umožňuje současné provádění několika testovacích případů, výrazně snižuje čas potřebný k provedení testů. Výsledkem je, že zatímco testy běží, mohou se členové vývojového týmu soustředit na jiné úkoly a postupy související s projektem.

Vyšší přesnost: Manuální testování je náchylné k lidským chybám. Automatické nástroje pro testování softwaru pomáhají tento faktor eliminovat a zajišťují konzistentní a přesné výsledky testů.

Znovupoužitelnost automatických testů: Automatizované testovací skripty lze používat opakovaně napříč projektů, což redukuje potřebu vytvářet nové testy od nuly. To z dlouhodobého hlediska šetří nejen čas, ale i lidské a finanční zdroje.

Větší pokrytí testů: Automatizované testovací nástroje mohou rychle provést velké množství testovacích případů, což umožňuje širší rozsah testování. Tím je zajištěno, že je otestováno více aspektů softwaru, což může vést k vyšší kvalitě aplikací.

(CI/CD)[4]: Automatické nástroje pro testování softwaru lze snadno integrovat do pipeline[5] CI/CD, což vývojářům umožňuje průběžně testovat kód a častěji a spolehlivěji nasazovat aktualizace softwaru.

Rychlejší zpětná vazba: Automatické testy poskytují rychlou zpětnou vazbu o výkonu softwaru a umožňují vývojářům identifikovat a řešit problémy rychleji než při manuálním testování. Automatické testy probíhají několikanásobně rychleji než manuální testování [15]

Nákladově efektivní: V dlouhodobém horizontu může ušetřit peníze, protože zkrátí čas a úsilí potřebné pro manuální testování.

Nevýhody automatického testování [30]:

Dovednosti a zkušenosti testera: Tvorba automatických testů je závislá na schopnostech testera. Při využití automatického testování se musí tester umět orientovat v programech pro tvorbu automatických testů a musí mít základní programovací znalost. V komplexních testovacích případech musí mít všestranné dovednosti.

Nekompletní pokrytí: Některé testovací případy vyžadují lidský náhled na řešení pro lepší simulaci uživatele.

Údržba: Automatické testy jsou potřeba pravidelně kontrolovat a upravovat na základě provedených změn v aplikaci.

[4] Kontinuální integrace a kontinuální dodávání
[5] Pipeline = řetězec procesů

Falešné výsledky: Při nesprávném setupu automatických testů může dojít k chybné identifikaci problémů či k přehlédnutí bugu.

3.6 Problematika výběru programů pro testování

V současnosti, kdy je testování softwaru nedílnou částí procesu vývoje, již existuje nespočet různých testovacích aplikací s odlišnými funkčnostmi a využitím, tudíž může nastat náročná situace, kdy tester či zbytek vývojářské týmu musí vůbec zvolit ty nejoptimálnější programy, které k budou pro konkrétní projekt využity na základě parametrů funkčností, doplňků, intuitivnosti a finančního hlediska. Všechny tyto parametry budou v této práci detailně zkoumány.

3.6.1 Programy pro editaci dokumentů

Při testování je mnohdy zapotřebí editace textových dokumentů za účelem úpravy testovacích dat, například při úpravě hodnot v xml, json a podobných formátech dokumentů.

Jako programy pro editaci dokumentů v této práci jsem zvolil Notepad++, Atom, Visual Studio Code a Sublime Text, jelikož se dle elektronického článku [1] jedná o jedny z nejvíce populárních textových editorů na trhu.

3.6.2 Nástroje pro práci s manuálními testy

Programy pro manuální testy jsou potřeba především při zasílání požadavků do aplikace. Uživatel si v programu může zkoušet zasílat requesty s různými vstupy a sledovat odpověď programu.

3.6.3 Programy pro tvorbu automatických testů

Tyto programy slouží k vytváření automatických testů a jejich následné aktualizaci v závislosti na změně požadavků. Zároveň by měly poskytovat přehled výsledku každého testovacího případu a to včetně samotných výsledků jednotlivých kroků v testovacím případu. Pro představu - existuje testovací případ, který má otestovat, že odeslaná zpráva je úspěšně doručena do systému. Program by nám měl ukázat finální výsledek -

že je zpráva přítomna v systému, ale dále by měl být schopen ukázat výsledku ověření obsahu zprávy, pokud je testovací případ obsahuje.

3.6.4 Management tool programy

Účelem management tool programů je poskytnout správu a organizaci testovacích scénářů a případů. Správný management tool program by měl disponovat i možností exportu graficky přehledného výsledku testovacích případů.

3.7 Frameworky pro Unit Testing

Rámce (frameworky) pro testování jednotek jsou pro vývojáře zásadními nástroji, protože jim umožňují vytvářet a spouštět testy, které ověřují funkčnost určitých jednotek nebo komponent softwarové aplikace. Tyto rámce poskytují disciplinovanou a standardizovanou metodu testování jednotek a zajišťují, že stavební prvky softwaru fungují tak, jak mají, a jsou spolehlivé.

Frameworky pro testování jednotek umožňují vývojářům izolovat konkrétní softwarové jednotky nebo komponenty, což usnadňuje vyhledávání a opravu problémů, aniž by byly ovlivněny ostatní komponenty aplikace. Jednotkové testovací rámce podporují automatizaci tím, že umožňují vývojářům provádět testy rychle a konzistentně. Aby bylo zajištěno časté testování aplikace, lze automatizované testy snadno integrovat do potrubí kontinuální integrace a kontinuálního dodávání (CI/CD).

Rámce pro testování jednotek obsahují funkce assertion[6], které vývojářům pomáhají specifikovat zamýšlené chování testované jednotky. Kromě toho vytvářejí zprávy o výsledcích testů, které vývojářům usnadňují hledání a opravu případných problémů.

Některé frameworky pro testování jednotek spolupracují s nástroji pro pokrytí kódu a umožňují vývojářům vypočítat podíl kódu, který testy pokryly. To pomáhá identifikovat oblasti, které vyžadují další testování, a pomáhá zaručit, že aplikace je dostatečně otestována.

Dalšími frameworky pro unit testing jsou takzvané Support TDD (Test-driven development), usnadňují vývojářům používání metody TDD, při které vývojáři píší

[6] aserce - ověření

testy před vývojem skutečného kódu. Při použití této techniky je vytvořený kód spolehlivější a organizovanější, což vede k nižšímu riziku chyb.

4 Praktická část

4.1 Testing toolbox

4.2 Definice testing toolboxu

Souhrn všech programů, které tester využívá pro svou práci se označuje jako testing toolbox.

Jedná se tedy nejen o aplikace určené přímo pro automatické testování, ale i aplikace pro test management (test management tools), které slouží pro správu, organizaci a export testů.

4.3 Ukázka využití testing toolboxu v praxi

4.4 Testerovo zadání

V rámci tvorby aplikace přibyla nová komponenta s registrem zpráv a je nutno ji pokrýt testy.

4.5 Řešení zadání

1. Tester vyhledá dokumentaci popisující požadavky a specifikace nové komponenty.
2. Tester v jednom z test management tool programu založí testovací scénáře, které přiřadí na zákazníka pro schválení.
3. Na základě schválení testovacích scénářů, tester vytvoří testovací případy. Testovací případy budou obsahovat především odesílání a přijímání zpráv a kontrolu jejich úspěšného zobrazení v registru zpráv.
4. Skrz program pro manuální testování tester otestuje manipulaci se zprávami, které si připraví v preferovaném nástroji pro editaci dokumentů.
5. Po úspěšném manuálním testování tester začne vytvářet automatické testy na novou komponentu pro pravidelné pokrytí této funkcionality.

   Tester zvolí program pro tvorbu automatických testů, ve kterém připraví požadavky pro metody využívané v rámci funkčnosti komponenty a

vytvoří ověřovací metody, ve kterých definuje očekávané stavy zpráv a jejich obsah.

6. Po vytvoření automatických testů vytvoří tester šablonu pro report výsledků automatických testů pro nové verze.
7. V moment kdy je nasazena nová verze, tester spustí automatické testy a exportuje výsledky do test management tool programu.

4.6 Aplikace pro editaci dokumentů

4.6.1 Notepad++

Notepad ++ je open source[7] editor textu, je velmi rozšířený a oblíbený po celém světě. Notepad ++ založil Don Ho v roce 2003. Od té doby se stal základním nástrojem pro uživatele pracující s textovými soubory, webovým vývojem nebo programovacími jazyky.

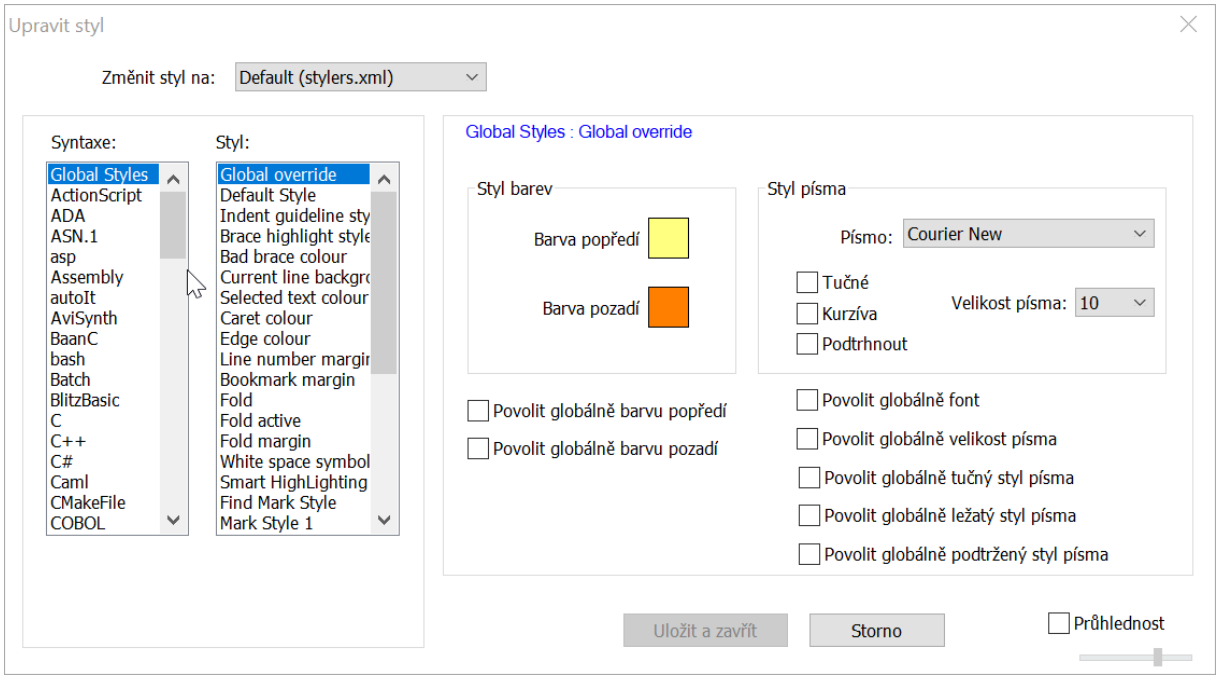
Program Notepad ++ byl navržen se zaměřením na produktivitu a efektivitu. Uživatelům nabízí velké množství funkcí a možností přizpůsobení, které uživatelům pomáhají lépe a spolehlivěji v práci.

Mezi užitečné funkce patří například transformace do různých syntaxí a jejich zvýrazňování a skládání, automatické dokončování a možnost editací více souborů najednou. Notepad++ disponuje vysoce přizpůsobitelným uživatelským rozhraním.

Výhody Notepad++

- upravitelné uživatelské prostředí - Notepad++ umožňuje rychle a jednoduše nastavit syntaxi a styl vzhledu programu

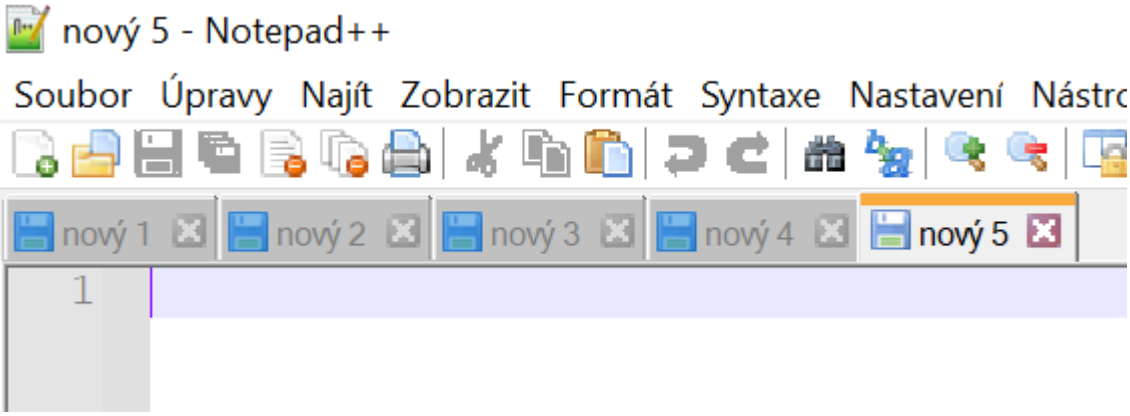
[7] volně dostupný a editovatelný program



Obrázek 5: Upravení stylu a syntaxe v Notepad++

Zdroj: Vlastní zpracování

- Hardwarová nenáročnost - Notepad++ je pouze lehce náročný pro operační systémy a zařízení
- Možnost editace více dokumentů najednou



Obrázek 6: Možnost editace více dokumentů najednou v Notepad++

Zdroj: Vlastní zpracování

- Použitelnost na všech platformách - přestože byl vyvíjen Notepad++ především pro Windows, je zcela použitelný i na macOS a Linuxu

- Robustní plugin support - oficiální pluginy lze nainstalovat přímo přes “Správu pluginů” v programu, existuje však možnost využití dodatečných pluginů instalací ze složky
    - plugin velmi využívaný a užitečný testery je plugin “Compare”, který umožňuje porovnání mezi dvěma soubory a nalezne všechny rozdíly, to lze velmi využít při testování zasílání a přijímání zpráv a mnoha dalších případech
- Nulové finanční náklady - oficiální Notepad++ je zdarma ke stažení v plné verzi
- Automatické doplňování - lze nainstalovat pluginy pro automatické dokončování a nabízení kódu - což může být velmi užitečné pro juniorní uživatele
- Funkce najít a nahradit: Notepad++ v základním stavu disponuje funkcí najít a nahradit, umožňující testerovi například změnit více hodnot najednou

Nevýhody Notepad++

- Práce s velkými soubory: při práci se soubory většími nad například 100MB probíhají úkony velmi pomalu a může dojít k poškození souboru
- Absence cloudového úložiště: Notepad++ nezahrnuje integrovaný systém pro práci s cloudovým úložištěm, což může být nepříjemné pro uživatele využívající více různých zařízení

Tabulka 7 Hodnocení Notepad++

Notepad++	Bodové ohodnocení
Automatické doplňování	2
Highlighting	2
Pluginy a doplňky	2
Příkazový řádek	3
Hardwarová nenáročnost	3
Intuitivnost	1
Podpora	2
Finanční ohodnocení	3

4.6.2 Atom

Atom je dalším z open source textových editorů, vyvinut GitHubem. Atom disponuje podporu pro kódování v mnoha jazycích jako například Python, C++, JavaScript a mnoha dalších.

Jelikož byl vyvinut GitHubem, je velmi užitečný při kolaboraci s Gitem. Je velmi adaptivní a lze dobře upravit pro spokojenost uživatele [18].

Výhody Atomu

- Open-source: Atom je bezplatný a svobodně upravitelný pro uživatelské využití, navíc je průběžně aktualizován skrze otevřenou komunitu uživatelů
- Použitelnost na všech platformách - Atom lze využívat na většině platforem, včetně těch hlavních jako je Windows, macOS a Linux
- Práce s více soubory: Podobně jako již zmíněný Notepad++, Atom umožňuje práci s několika soubory najednou, s možností “split view” (=textové soubory jsou otevřený v menším rozlišením vedle sebe)

- Integrovaná Git podpora: Díky integrované Git podpoře se uživatelům snáze využívá propojení s Gitem, tester si tak může jednoduše otevřít více souborů s různými verzemi podle Gitu
- Automatické doplňování: Funkce velmi užitečná juniorním testerům, jelikož automaticky nabízí a doplňuje části kódu
- Podpora více programovacích jazyků: V základním stavu Atom disponuje podporou více programovacích jazyků jako například Python, JavaScript a další

Nevýhody Atomu

- Systémová náročnost: Atom narozdíl od Notepad++ není lightweight[8] program, tudíž nastartování a samotný chod Atomu bývá pomalejší na méně výkonných zařízeních, což se často vztahuje na juniorní testery, jelikož nemají svou firmou přiděleny více výkonná zařízení
- Závislost na balíčcích: Jelikož jsou některé funkce Atomu založeny na externích balíčcích, může dojít k inkompatibilitě mezi nimi a uživatel musí svou pozornost a čas věnovat troubleshootingu[9]
- Strmá křivka používání: Atom je na první pohled uživatelsky přívětivý. Při využívání komplexnějších funkčností musí uživatel často řešit problémy s balíčky a musí se naučit ovládat dané funkcionality, což není ideální pro juniorní testery
- Příkazový řádek není integrovaný v programu a je potřeba dodatečně nainstalovat
- Zvýrazňování syntaxí není integrováno v programu a je nutné dodatečně nainstalovat

[8] program, který je nenáročný pro zařízení
[9] odstraňování problému, termín používaný v IT

Tabulka 8 Hodnocení Atomu

Atom	Bodové ohodnocení
Automatické doplňování	3
Highlighting	1
Pluginy a doplňky	2
Příkazový řádek	1
Hardwarová nenáročnost	1
Intuitivnost	2
Podpora	2
Finanční ohodnocení	3

4.6.3 Visual Studio Code

Vyvinut společností Microsoft v roce 2015, open source editor Visual Studio Code je jedním z nejrozšířenějších a nejvíce používaných editorů kódu.

V základním stavu zahrnuje podporu pro programovací jazyky JavaScript, C, C Sharp, kaskádové styly a TypeScript a je použitelný na většině platforem [19].

Výhody Visual Studio Code

- Dostupnost na všech platformách - VSCode je dostupný na hlavních platformách jako je Windows, MacOS a Linux
- Rozšiřitelnost - VSCode má integrovanou knihovnu pro stažení a instalaci nejpopulárnějších pluginů a doplňků
- Integrovaná Git podpora - VSCode umožňuje snadné používání Git operací jako například klonování repozitáře, aplikace patchů[10], zobrazení souboru v různých verzích, aplikování změn a další

[10] patch - soubor změn, které aktualizují verzi

- Automatické doplňování - VSCode umožňuje nastavení automatického nabízení a doplňování částí kódů - usnadnění a zrychlení práce
- Integrovaný debugger[11]
- Cena - jakožto open source program, VSCode je bezplatný
- Integrovaný terminál - VSCode obsahuje terminál pro využití pokynů bez toho, aby uživatel musel opustit program

Nevýhody Visual Studio Code

- Klávesové zkratky - pro plné využití benefitů Visual Studio Code se musí uživatel naučit klávesové zkratky pro různé funkcionality
- Nekvalitní pluginy - jelikož uživatelé mohou přispívat svými vlastními pluginy, nemusí být vždy kvalitní a uživatel musí více bádat při výběru pluginů
- Práce s nastavením - sekce nastavení Visual Studio Code je velmi detailní a některé problémy je náročné vyřešit kvůli orientaci v nastavení

Tabulka 9 Hodnocení Visual Studio Code

Visual Studio Code	Bodové ohodnocení
Automatické doplňování	3
Highlighting	3
Pluginy a doplňky	2
Příkazový řádek	3
Hardwarová nenáročnost	2
Intuitivnost	2
Podpora	3
Finanční ohodnocení	3

[11] nástroj pro hledání chyb v kódu a jeho následné doladění

4.7 Aplikace pro manuální testování

4.7.1 Insomnia

Insomnia je open source testovací program, ve kterém tester může jednodušše odesílat požadavky a zobrazovat jejich odpovědi. Insomnia disponuje velmi intuitivním uživatelským prostředím. Umožňuje identifikaci bugů a vytvářet vlastní požadavky. Insomnia je dostupná na většině platforem. Jedná se o velmi nový program, jelikož vznikla v roce 2016.

Výhody Insomnia

- intuitivní uživatelské prostředí - Insomnia disponuje přehledným a intuitivním uživatelským rozhraní s možnostmi dodatečných úprav jejího vzhledu
- práce s proměnnými - Insomnia umožňuje deklarování a definování pro různá prostředí
- kvalitní podpora protokolů mezi něž patří SOAP[12], REST, GraphQL
- integrovaný support pro různé typy požadavků - v Insomnii lze snadno využít základní typy požadavků jako je například GET, POST, PUT, DELETE, LIST, PATCH a mnoho dalších
- request chaining = funkcionalita Insomnie, která umožňuje zrychlit testování pomocí využití odpovědi jednoho požadavku jako vstup pro další požadavek
- import/export - Insomnia umožňuje export do různých formátů jako je JSON, YAML a další, což umožňuje snadnější kolaboraci s dalšími programy jako je například Postman
- rozšiřitelnost - Základní funkcionality Insomnie lze obohatit o dostupné pluginy, které je možné získat a nainstalovat přímo skrze Insomnii

Nevýhody Insomnia

- absence mobilní aplikace - Insomnie neexistuje na mobilních zařízeních, tudíž není možné ji využívat pro testování mimo desktopové zařízení
- výkonnostní náročnost - při manipulaci s velkým souborem dat často dochází ke zpomalení zařízení při používání Insomnie

[12] Simple Object Access Protocol - manipulace se zprávami ve formátu xml

Tabulka 10 Hodnocení Insomnie

Insomnia	Bodové ohodnocení
Podpora protokolů	3
Typy testování	2
Zprovoznění programu	3
Hardwarová nenáročnost	3
Intuitivnost	3
Podpora	3
Finanční ohodnocení	3

4.7.2 SoapUI

Soap UI je program vyvinutý společností SmartBear Software za účelem testování API[13] služeb. Umožňuje zasílání požadavků různých protokolů, díky čemuž lze identifikovat chyby v požadavcích a odpovědích systému.

Výhody SoapUI

- uživatelsky přívětivé rozhraní
- podpora mnoha protokolů jako je například SOAP, REST, JDBC
- open source - existuje nezpoplatněná verze
- dobrá dokumentace - lze snadno dohledat pomoc
- umožňuje komplexní testování
- upravitelné skrze skripty

Nevýhody SoapUI

- středně obtížná příprava používání
- strmější křivka náročnosti používání
- bezplatná verze je limitovaná - neobsahuje některé užitečné funkcionality
- při obsáhlejších projektech bývá pomalejší

[13] API - rozhraní pro programování aplikací

Tabulka 11 Hodnocení SoapUI

SoapUI	Bodové ohodnocení
Podpora protokolů	3
Typy testování	3
Zprovoznění programu	2
Hardwarová nenáročnost	2
Intuitivnost	2
Podpora	3
Finanční ohodnocení	2

4.8 Aplikace pro tvorbu automatických testů pro backend

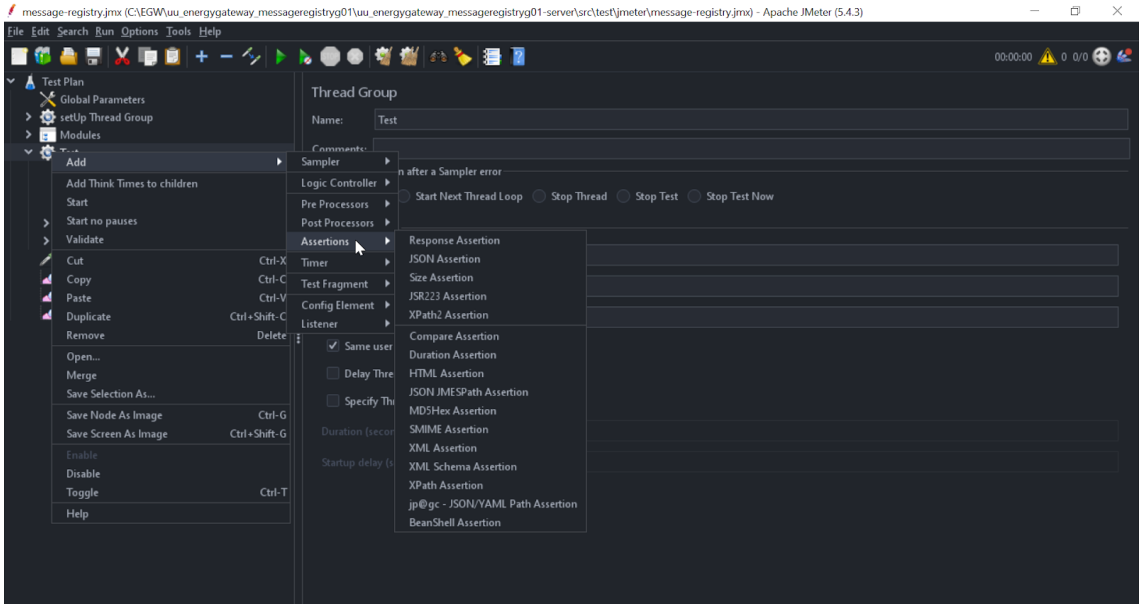
4.8.1 JMeter

Apache JMeter je open source software založený na Javě, který byl vytvořen za cílem testovat výkony aplikací, webové služby a síťové protokoly. Má nespočet funkčností a je velmi versatilní. Byl vyvinut Stefanem Mazzocchim v roce 1998 a je nadále rozvíjen komunitou vývojářů pracujících na projektu Apache Jakarta Project.

Výhody JMeteru

- Open-source - JMeter je zdarma k používání v plné verzi, tudíž je z finančního hlediska JMeter skvělá volba
- Využitelnost na různých platformách - pokud je platforma kompatibilní s Java Virtual Machine, lze na ní bez problému JMeter využívat. Windows, MacOS i Linux jsou všechny s Java Virtual Machine kompatibilní, tudíž na nich lze JMeter využívat
- Rozšiřitelnost - JMeter podporuje širokou škálu rozšíření a pluginů

- Support protokolů - JMeter umožňuje využití mnoha protokolů, jako například http/https, FTP[14], SOAP, JDBC[15]
- Integrované testovací elementy



Obrázek 7: Asserty v JMeteru

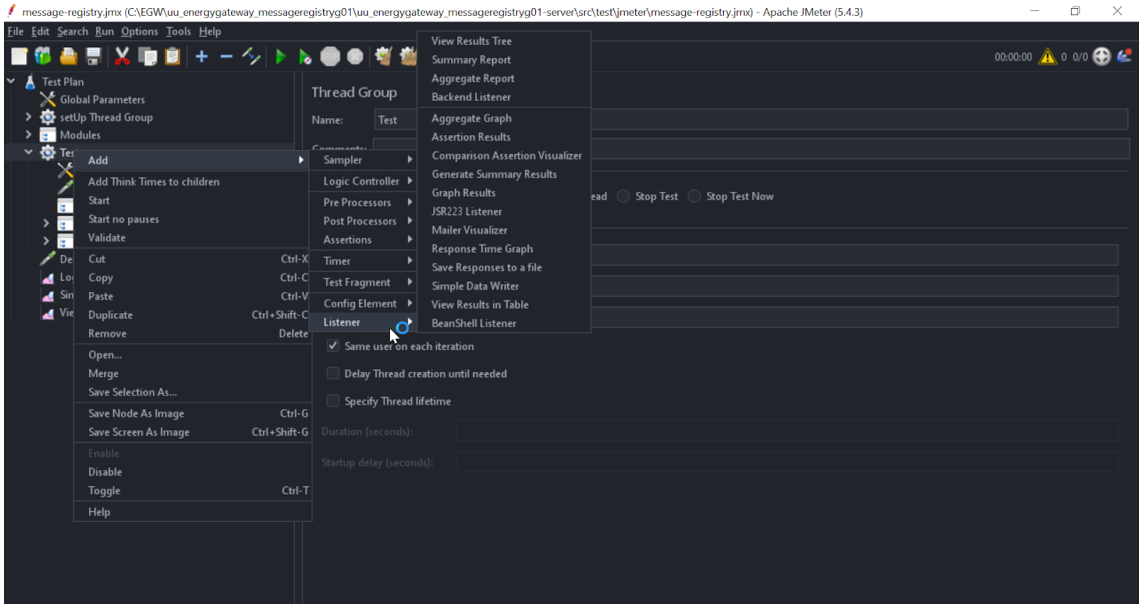
Zdroj: Vlastní zpracování

Na obrázku 7 je ukázka integrovaných asercí. Skrze Response Assertion jde definovat úseky v response, u kterých chce tester ověřit, že jsou přítomné, zároveň lze lehce ověřit i opak - nepřítomnost subjektu v odpovědi, to lze uskutečnit pomocí parametru “not”.

Dále lze vytvořit aserce na základě použitého formátu odpovědi, jako například JSON assertion, ve které tester definuje JSON path (hierarchii a cestu k testovanému parametru) a to samé pro XML a další formáty.

[14] FTP - standardní komunikační protokol pro přenos souborů mezi počítači pomocí počítačové sítě

[15] Java Database Connectivity - umožňuje kolaboraci s relačními databázemi



Obrázek 8: Listeners v JMeteru

Zdroj: Vlastní zpracování

Tester dále může využít zabudované funkčnosti “Listener”, ve které může tester zvolit View Results Tree, kde vidí výsledky testovaných modulů s detaily - aserce, které dopadly úspěšně či neúspěšně s detailním důvodem proč neuspěly. V praxi můžeme mít asserci na přítomnost identifikátoru odesílané zprávy. Pokud identifikátor nebude přítomno v odpovědi a má být, výsledkem aserce bude, že identifikátor není přítomen nebo že se neshoduje s očekávanou hodnotou.

Další užitečnou funkčností JMeteru jsou takzvané extraktory, které umožňují vyextrahovat hodnotu parametru z jedné odpovědi a dále ji ověřit v dalších odpovědích. To v první řadě šetří čas a dále zvyšuje efektivitu testování.

- Integrovaná možnost exportu výsledků - při využití View Results Tree, tester může ve zvoleném formátu vyexportovat detailní výsledky testů, tyto vyexportované výsledky jde následně zpětně vložit do View Results Tree, kde se výsledky zobrazí přímo v JMeteru. Pokud tedy pracuje na jedné funkčnosti více testerů, mohou si navzájem lehce a rychle sdílet informace o testech nebo výsledky poskytnout vývojářům pro lepší porozumění bugu, výsledky jde zobrazit ve více formách - v grafech, chartech či tabulkách a podobně
- Testování výkonu - JMeter má v sobě integrovanou možnost pro testování výkonu aplikace, jejíž součástí je také možnost deklarovat počet zaslaných zpráv a jejich velikost pro snadné a efektivní testování výkonu

- Versatilita - JMeter lze využít prakticky testování všech částí vyvíjené aplikace
- Práce s prostředím - JMeter umožňuje deklarovat a definovat parametry pro různá testovací prostředí. Tester může deklarovat a definovat různé proměnné pro každé prostředí samostatně
- Ukládání proměnných - tester může snadno deklarovat a definovat proměnné a dále s nimi pracovat
- Integrovaná kvalitní podpora - JMeter obsahuje v základu návody pro používání - *"Nástroj má svoji nápovědu, která je celkem dobrá a uvádí i příklady, jak jednotlivé obrazovky (samplery) využívat, samozřejmě je potřeba i googlit, protože některé specifické problémy, momenty nápověda neřeší. Zde spíš pomáhají uživatelé, kteří se s podobnými situacemi setkali."* - test inženýrka Bc. Monika Mizerová, citováno 1. 7. 2023

Nevýhody JMeteru

- Intuitivnost - Ze začátku může pro juniorního testera JMeter působit nepřehledně a obtížně, ale není tomu tak. Jakmile tester pochytí základy orientace v JMeteru, práce se nadále stává intuitivní, navíc existuje spousta dokumentace zahrnující postup v JMeteru a návody, jak od vývojářů JMeter tak od jeho komunity
- Systémová náročnost - Při komplexním testování může, především při testování akcí více uživatelů najednou JMeter poměrně zatížit zařízení, které tester využívá

Tabulka 12 Hodnocení JMeteru

JMeter	Bodové ohodnocení
Reportování testů	2
Podpora aserce	3
Znovupoužitelnost testů	3
Pluginy a doplňky	2
Udržitelnost testů	3
Hardwarová nenáročnost	2
Testování výkonu	3
Intuitivnost	1
Podpora	2
Finanční ohodnocení	3

4.8.2 Postman

Postman je API testovací nástroj, který poskytuje možnost konstrukce, testování a dokumentaci API. Je použitelný na platformách Windows, Linux a MacOS. Postman je poměrně nový oproti ostatním zmíněným programům - byl uveden na trh v roce 2012. Existuje jeho bezplatná verze, ale plná verze je placená.

Výhody Postmanu

- Intuitivní uživatelské rozhraní - Postman má intuitivně navržené uživatelské prostředí, což urychluje a usnadňuje jeho využití pro uživatele - "*Uživatelské rozhraní Postmanu je velmi podobné Insomnii, která má poměrně intuitivní uživatelské rozhraní"* - Kateřina Smrčková, testerka, Gen Digital, citováno 1. 7. 2023

- Kolaborace a sdílení - Postman umožňuje vzájemné sdílení prostředí, kolekcí a dokumentací mezi dalšími testery a vývojáři
- Podpora API protokolů - přítomnost podpory pro SOAP, REST[16], GraphQL a dalších
- Komunitní dokumentace: k snadnější orientaci v programu existuje obsáhlý návod, do kterého přispívá i komunita

Nevýhody Postmanu

- Cena - přestože existuje verze Postmanu zdarma, pro důkladné testování je třeba zakoupit obsáhlejší verze.
    - Basic verze: po zaokrouhlení od 280,- do 330,- za jednoho uživatele na měsíc
    - Professional verze: po zaokrouhlení od 660,- do 790,- za jednoho uživatele na měsíc
    - Enterprise: po zaokrouhlení vychází na 2180,- za jednoho uživatele na měsíc
- Limitovaná podpora pro výkonnostní testy: Postman disponuje především možnostmi pro funkcionální testování
- Náročné využití pokročilých vlastností - při komplexním testování tester potřebuje vyšší znalost programování či strávit delší dobu hledáním podpory - "*I po třech letech používání se mi občas stane, že mě překvapí, že něco nelze nebo hledám věci jinde, než jsou.*" - Kateřina Smrčková, testerka, Gen Digital, citováno 1. 7. 2023

[16] representational state transfer - cesta, jak jednoduše vytvořit, číst, aktualizovat nebo smazat informace ze serveru pomocí jednoduchých HTTP volání

Tabulka 13 Hodnocení Postmanu

Postman	Bodové ohodnocení
Reportování testů	2
Podpora aserce	2
Znovupoužitelnost testů	2
Pluginy a doplňky	2
Udržitelnost testů	2
Hardwarová nenáročnost	2
Testování výkonu	3
Intuitivnost	2
Podpora	2
Finanční ohodnocení	1

4.9 Test management tools

4.9.1 Testlink

Testlink je dalším z open source aplikací. Jedná se o webový test management tool, jehož hlavními funkcionalitami je vytváření testovacích plánů, testovacích případů, designů, jejich správu a export výsledků testování. Testlink byl vyvinut v roce 2004 společností TeamTest a je dostupný na většině platforem.

Výhody Testlinku

- Open-source: Jakožto open-source program je Testlink k užití bez platby
- Centralizovaný management: Testlink má centralizovaný management pro testování - to znamená, že testovací případy, testovací plány, projekty, požadavky a verze aplikací jsou na jednom místě což umožňuje snadnou orientaci

- Propojení požadavků: Požadavky lze předem deklarovat pro určené celky a následně je propojit například s testovacími případy
- Správa verzí: V Testlinku lze snadno vytvářet nové verze bez opakovaného vytváření nových testovacích případů. Tester může mít již připravené testovací případy týkající se minulých verzí a pokud je potřeba například regresní testování těchto případů, tester založí novou verzi a následně do ní pomocí checkboxu přidá již existující testovací případy.
- Reportování: Testlink má integrovanou možnost exportu výsledků testovacích případů, včetně grafů jako například procentuální koláčový graf rozdělený na úspěšně a neúspěšné testy
- Uživatelsky přívětivý: Vzhled uživatelského prostředí Testlinku je sice zastaralý (více popsáno v nevýhodách), ale umožňuje snadnou orientaci ve většině funkcionalit až na pár, které jsou zmíněny v nevýhodách

Nevýhody Testlinku

- Zastaralý vzhled: Testlink má oproti ostatním zmíněným test management nástrojům velice zastaralý vzhled, což většině uživatelů snižuje požitek z práce s Testlinkem
- Absence testlinku na mobilních zařízeních: To může negativně ovlivnit manažery, kteří často potřebují mít přístup ke stavu aplikace i mimo kancelář
- Limitované doplňky: Testlink disponuje podporou pro pár základních nástrojů jako je například Jira, veškeré dodatečné doplňky jsou však buďto nevyužitelné nebo je velmi náročné je zprovoznit
- Sekce ve formě obrázků: Tlačítka na přechod do některých sekcí jsou formou miniaturních obrázku bez popisu při hoveru[17], což zbytečně snižuje orientaci v Testlinku

[17] hover je označení pro najetí kurzorem na element

Tabulka 14 Hodnocení Testlinku

Testlink	Bodové ohodnocení
Plánování a organizace testů	1
Verzování a historie	2
Zabezpeční dat	1
Pluginy a doplňky	1
Intuitivnost	1
Podpora	2
Finanční ohodnocení	3

4.9.2 PractiTest

PractiTest je placený test management tool s cloudovým úložištěm, založený roku 2008, který umožňuje správu testování od úplného začátku (definování specifikací a požadavků) až po předání zákazníkovi.

Výhody PractiTestu

- Centralizovaná správa - Practitest poskytuje správu většiny procesů vývoje aplikace, od managementu projektu, po tvorbu testovacích scénářů a případů
- Integrovaná komunikace s ostatními nástroji - Practitest umožňuje snadnou komunikaci s ostatními nástroji pro správu testů či jejich spouštění jako jsou například Jira, Selenium a mnoho dalších, výsledkem je usnadnění zacházení s programem a úspora času při práci
- Přístupnost v týmu - Practitest umožňuje snadný přístup k popisu existujících chyb nejen pro testery, ale i pro vývojáře jejichž úkolem je nalezené chyby opravit, díky přístupu pro všechny členy týmu se snižuje pravděpodobnost nedorozumění

- Intuitivní uživatelské rozhraní - Practitest má moderní vzhled a jeho rozložení je snadné pro juniorní uživatele
- Cloudové úložiště - Practitest umožňuje ukládání projektu na cloud, tudíž lze snadno projekt rozšiřovat a je lehce dostupný
- Zabezpečení - Practitest disponuje možností šifrování dat, správou přístupů a průběžné bezpečnostní kontroly
- Silná podpora - Tým pro podporu Practitestu je velmi reaktivní a poskytuje mnoho návodu přímo v aplikaci či v její dokumentaci online, navíc existuje mnoho návodů od samotných uživatelů včetně videí

Nevýhody PractiTestu

- Cena - přestože existuje bezplatná verze Practitestu, pro větší projekty využívající pokročilejší funkcionality je tato verze často nedostatečná
    - Týmová verze s licencí pro 5 uživatelů stojí po zaokrouhlení 1 050,- na měsíc
    - Korporátní verze začínající na 10ti licencích je poskytnuta skrze domluvu s týmem Practitestu
- Dokumentace - některé části dokumentace Practitestu nejsou psané dostatečně přívětivě pro uživatele a ten musí občas hledat jednodušší vysvětlení v alternativních zdrojích
- Cloudové úložiště - Cloudové úložiště se jeví pro většinu projektů jako benefit. Neexistuje zde však možnost ukládání dat interně, což nemusí vyhovovat projektů se striktními požadavky na zabezpečení

Tabulka 15 Hodnocení Practitestu

Practitest	Bodové ohodnocení
Plánování a organizace testů	3
Verzování a historie	3
Zabezpeční dat	3
Pluginy a doplňky	2
Intuitivnost	2
Podpora	2
Finanční ohodnocení	1

4.9.3 Jira

Jira je velmi rozšířený placený management tool, který slouží pro vytváření a trackování úkolů, chyb a mnoha dalších na celém projektu. Podporuje různé typy přístupů jako například Scrum[18], agilní přístup a Kanban[19]. Jira je použitelná nejen v odvětví IT a byla založená v roce 2002 společností Atlassian.

Výhody Jiry

- Transparentní tabule: Jira umožňuje vytváření tabulí, u kterých lze snadně upravovat, kdo všechno je může upravovat a vidět, což poskytuje skvělý přehled o stavu projektu v reálném čase, protože obsah tabulí se automaticky mění na základě stavu tiketů obsažených v tabulích. Je zde také možnost používání různých grafů pro dodatečnou grafickou reprezentaci stavu projektu.
- Spolupráce s Gitem a TeamCity: Velmi užitečnou vlastnost, kterou vysvětlím v praxi:

  Máme například tiket, který má parametr fix version. Pokud je správně nastavené propojení s Gitem nebo TeamCity, fix verze je nastavena jako

[18] iterativní a inkrementální metoda organizace vývoje softwaru
[19] metoda pro vytváření cedulí s rozpisem úkoly a jejich pořadí ve frontě na základě jejich priority

číslo verze - next - komponenta. Jakmile je změna opravující chybu aktivní v repozitářích, fix version se sám přepíše na aktuální stav verze a tester ví, že je oprava aktivní a nemusí kontrolovat s Gitem

- Rozšiřitelnost: Jira je vhodná pro malé projekty i pro korporátní obsáhlé projekty
- Integrované vykázání času: Jira umožňuje zaznamenávání odpracovaného času na jednotlivých úkolech, což je přehlednější jak pro zaměstnance tak pro management
- Prioritizace: Jira umožňuje nastavení priority každého tiketu, což pomáhá efektivnímu řešení problému
- Podporuje agilní vývoj: Jira umožňuje využití metod jako je Scrum, Kanban a další
- Zabezpečení: Jira disponuje správou přístupu, šifrováním dat a kvalitním zabezpečením a body obnovy
- Zmínky: V tiketech lze zmínit ostatní pracovníky v komentářích. To odešle notifikaci o zmínění v daném tiketu
- Robustní filtr: V Jiře lze filtrovat skrze robustní základní filtr, přes který uživatel vybírá parametry pro filtrování. Dále také uživatel může využít pokročilý filtr, kdy filtruje pomocí psaní řetězce

Nevýhody Jiry

- Cena: Jira může být poměrně nákladná pro obsáhlejší týmy. Využití některých pluginů je také zpoplatněné
- Závislost na pluginech: Pro uspokojivé fungování Jiry je třeba zakoupení pluginů a doplňků, což opět zvyšuje finanční náročnost
- Nevhodná pro neagilní projekty: Pro projekty využívající například Waterfall model není Jira vhodná
- Závislost na Jiře: Po dlouhodobějším využívání Jiry je velmi náročné pro projekt migrovat na jinou platformu z důvodu její komplexnosti
- Pomalá reakce týmu podpory

Tabulka 16 Hodnocení Jiry

Jira	Bodové ohodnocení
Plánování a organizace testů	3
Verzování a historie	3
Zabezpeční dat	3
Pluginy a doplňky	1
Intuitivnost	2
Podpora	2
Finanční ohodnocení	1

5 Shrnutí výsledků

Shrnutí výsledků bude znázorněno prostřednictvím tabulky s hodnocením.

5.1 Vyhodnocení nástrojů pro editaci dokumentů

Tabulka 17 Vyhodnocení nástrojů pro editaci dokumentů

	Notepad++	Atom	Visual Studio Code
Automatické doplňování	2	3	3
Highlighting	2	1	3
Pluginy a doplňky	2	2	2
Příkazový řádek	3	1	3
Hardwarová nenáročnost	3	1	2
Intuitivnost	1	2	2
Podpora	2	2	3
Finanční ohodnocení	3	3	3
Celkový počet bodů	18	15	21

5.1.1 Výsledek vyhodnocení nástrojů pro editaci dokumentů

Největší bodové ohodnocení obdrželo Visual Studio Code, především na základě uživatelského prožitku. Visual Studio Code disponuje mnoha integrovanými funkcionalitami jako je příkazový řádek, zvýraznění syntaxe, automatické doplňování a formátování. Navíc jakákoliv dodatečná instalace pluginů je velmi snadná přímo skrze samotný program.

5.2 Vyhodnocení programů pro manuální testování

Tabulka 18 Vyhodnocení programů pro manuální testování

	Insomnia	SoapUI
Podpora protokolů	3	3
Typy testování	2	3
Zprovoznění programu	3	2
Hardwarová nenáročnost	3	2
Intuitivnost	3	2
Podpora	3	3
Finanční ohodnocení	3	2
Celkový počet bodů	20	17

5.2.1 Výsledek vyhodnocení programů pro manuální testování

Insomnia získává vyšší bodové ohodnocení především díky vyšší intuitivnosti, obsáhlejší bezplatné verze aplikace a náročností zprovoznění programu, čímž může být označena jako vhodnější pro juniorního testera v menší až střední společnosti.

SoapUI však nabízí oproti Insomnii robustnější testování, které může lépe využít zkušenější tester.

5.3 Vyhodnocení programů pro tvorbu automatických testů

Tabulka 19 Vyhodnocení programů pro tvorbu automatických testů

	JMeter	Postman
Reportování testů	2	2
Podpora aserce	3	2
Znovupoužitelnost testů	3	2
Pluginy a doplňky	2	2
Udržitelnost testů	3	2
Hardwarová nenáročnost	2	2
Testování výkonu	3	3
Intuitivnost	1	2
Podpora	2	2
Finanční ohodnocení	3	1
Celkový počet bodů	24	20

5.3.1 Výsledek vyhodnocení programů pro tvorbu automatických testů

JMeter se jeví jako lepší program pro tvorbu automatických testů především z pohledu finančního ohodnocení. To hraje roli především pro juniorní testery, kteří mohou JMeter využít i ve své volné chvíli pro zlepšení svých dovedností.

5.4 Vyhodnocení test management tool programů

Tabulka 20 Vyhodnocení test management tool programů

	Testlink	Practitest	Jira
Plánování a organizace testů	1	3	3
Verzování a historie	2	3	3
Zabezpeční dat	1	3	3
Pluginy a doplňky	1	2	1
Intuitivnost	1	2	2
Podpora	2	2	2
Finanční ohodnocení	3	1	1
Celkový počet bodů	11	16	15

5.4.1 Výsledek vyhodnocení test management tool programů

Practitest je vyhodnocen bodově nejlépe, jelikož oproti Jiře je více založen přímo na testování. Jeho jedinou nevýhodou je, že se jedná o placený program.

Pokud si uživatel nemůže dovolit za program platit tak může využít bezplatný Testlink, který je sice ohodnocen nejméně body, ale stále se jedná o použitelný program.

6 Závěry a doporučení

Účelem této bakalářské práce bylo obeznámit čtenáře s testováním softwaru, představit životní cykly vývoje aplikací a jejich fáze, popsat co je to testing toolbox, představit jaké kategorie programů do něho patří a tyto programy vzájemně porovnat a vyhodnotit, který z programů lze nejvíce doporučit pro juniorního testera.

Jako první byla představena metodika hodnocení porovnávaných programů na základě kategorií do které programy patří. Většina parametrů byla unikátní pro danou kategorii, ale zároveň byl součástí hodnocení všech programů uživatelský prožitek, tím jest intuitivnost programu a možnost dohledání podpory. Každý program také obdržel body za finanční ohodnocení.

Dále byly představeny modely vývoje softwaru včetně představení jejich výhod a nevýhod. Jednotlivé modely byly popsaný skrze pořadí jejich fází a jejich podstatnost. Z této části lze získat představu o tom, jak jsou představené modely komplexní a jak vhodné jsou pro menší či větší společnosti vyvíjející software.

Další bylo obeznámí s manuálním a automatickým testováním - typy na které se manuální testování dělí a na způsoby, kterými se manuálně testuje - především typy testování vstupů a výstupů. Účel zakomponování automatického testování byl ukázán skrze představení jeho výhod a nevýhod.

Problematika výběru programů pro testování nastínila, že juniorní tester může být přehlcen možnostmi programů na trhu a představila otázku, jaké programy lze pro juniorního testera doporučit. Každá kategorie obsahovala úvod, ve kterém bylo vysvětleno, čím je daná kategorie specifická a proč jsou tyto programy součástí testing toolboxu.

V praktické části je demonstrován ukázkový den testera - prezentace zadaného úkolu a jeho řešení rozděleno do kroků, během kterých tester využije programy ze svého testing toolboxu.

Hodnocení jednotlivých programů bylo uskutečněno skrze prezentaci funkcionalit, vlastností a dalších parametrů vybraných programů. Pro snadnější orientaci byly vypsány výhody a nevýhody každého programu a jeho bodové ohodnocení.

Toto ohodnocení pak bylo graficky znázorněno pomocí tabulek, ve kterých bylo finální bodové hodnocení programů z dané kategorie a následné doporučení.

Věřím, že jsem ve své práci vysvětlil dostatečně veškerou terminologii dané oblasti pomocí poznámek pod čarou.

Ve své práci vidím jedno omezení, konkrétně v parametrech uživatelského prožitku z programu, jelikož tento parametr je do určité míry subjektivní a může se pro každého uživatele lišit.

Pro další bakalářské práce na toto téma bych mohl doporučit porovnat další programy, které jsem nezmínil ve své práci nebo detailněji ohodnotit některé parametry hodnocení.

Jako přínos své bakalářské práce bych označil nastínění ukázkového dne testera a především poskytnutí detailního srovnání vybraných programů a jejich doporučení, ze kterých lze poskládat kompletní testing toolbox.

7 Seznam použité literatury

[1] ATHOW, Desire. Best text editors of 2023. Techradar [online]. 2022, 1 [cit. 2023-02-26]. Dostupné z: https://www.techradar.com/best/best-text-editors

[2] ARYANDANA, I G S, A E PERMANASARI a T B ADJI. Comparing method equivalence class partitioning and boundary value analysis with study case add medicine module. IOP science [online]. 2020, 9 [cit. 2023-03-02]. Dostupné z: https://iopscience.iop.org/article/10.1088/1757-899X/732/1/012072/meta

[3] P. J. H. King, Decision tables, The Computer Journal, Volume 10, Issue 2, August 1967, Pages 135–142, [cit. 2023-03-02]. Dostupné z: https://doi.org/10.1093/comjnl/10.2.135

[4] Khan, Mohd. Ehmer, Different Approaches to Black Box Testing Technique for Finding Errors (July 21, 2021). International Journal of Software Engineering & Applications (IJSEA), Vol.2, No.4, October 2011, [cit. 2023-03-02] Available at SSRN: https://ssrn.com/abstract=3890672

[5] In: Guide to the Software Engineering Body of Knowledge [online]. United States of America: Angela Burgess, 2001, s. 219 [cit. 2023-03-11]. Dostupné z: https://cmapspublic.ihmc.us/rid=1K1K7VJ2J-1QMZQ0M-2SXJ/SWEBOK.pdf#page=88

[6] W. Zhong-Min, H. Yi-Wei and L. Chen, "Design and Implementation of a Static Test Approach for Embedded Software," 2012 Third World Congress on Software Engineering, Wuhan, China, 2012, pp. 125-127, [cit. 2023-03-02]. Dostupné z: doi: 10.1109/WCSE.2012.30.

[7] S. Bauersfeld, T. E. J. Vos, K. Lakhotia, S. Poulding and N. Condori, "Unit Testing Tool Competition," 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops, Luxembourg, Luxembourg, 2013, pp. 414-420, [cit. 2023-03-02]. Dostupné z: doi: 10.1109/ICSTW.2013.55.

[8] S. Tahvili, M. Saadatmand, S. Larsson, W. Afzal, M. Bohlin and D. Sundmark, "Dynamic Integration Test Selection Based on Test Case Dependencies," 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Chicago, IL, USA, 2016, pp. 277-286, [cit. 2023-03-02]. Dostupné z: doi: 10.1109/ICSTW.2016.14.

[9] M. Lindvall, A. Porter, G. Magnusson and C. Schulze, "Metamorphic Model-Based Testing of Autonomous Systems," 2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET), Buenos Aires, Argentina, 2017, pp. 35-41, [cit. 2023-03-02]. Dostupné z: doi: 10.1109/MET.2017.6.

[10] F. Dias and A. C. R. Paiva, "Pattern-Based Usability Testing," 2017 IEEE International Conference on Software Testing, Verification and Validation

Workshops (ICSTW), Tokyo, Japan, 2017, pp. 366-371, [cit. 2023-03-02]. Dostupné z: doi: 10.1109/ICSTW.2017.65.

[11] BrowserStack [online]. [cit. 2023-03-12]. Dostupné z: https://www.browserstack.com/cross-browser-testing

[12] TeamCity [online]. online: Vývojář JetBrains, 2006 [cit. 2023-03-12]. Dostupné z: https://www.jetbrains.com/teamcity/

[13] Agile software development and UX design: A case study of integration by mutual adjustment. Elsevier [online]. 2022, 10 [cit. 2023-03-19]. Dostupné z: doi:https://doi.org/10.1016/j.infsof.2022.107059

[14] Sebastian Vöst. 2015. Vehicle level continuous integration in the automotive industry. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015). Association for Computing Machinery, New York, NY, USA, 1026–1029. [cit. 2023-03-02]. Dostupné z: https://doi.org/10.1145/2786805.2803193

[15] Testování softwaru: Kapitola 3: Nástroje pro testování - Rychlost [online]. Hradec Králové, 2017 [cit. 2023-03-19]. Dostupné z: https://theses.cz/id/dsbtq2/STAG87242.pdf. Bakalářská práce. Univerzita Hradec Králové.

[16] WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC. International Journal of Information Technology and Business Management [online]. 2012, 5 [cit. 2023-03-19]. Dostupné z: https://mediaweb.saintleo.edu/Courses/COM430/M2Readings/WATEERFALLVs%20V-MODEL%20Vs%20AGILE%20A%20COMPARATIVE%20STUDY%20ON%20SDLC.pdf

[17] Agilní model řízení projektu [online]. In: . [cit. 2023-03-19]. Dostupné z: https://fesordata.cz/cs/jak-pracujeme/

[18] Atom [online]. GitHub, 2015 [cit. 2023-03-21]. Dostupné z: https://github.com/atom/atom

[19] Visual Studio Code [online]. Microsoft, 2015 [cit. 2023-03-21]. Dostupné z: https://code.visualstudio.com/

[20] SDLC - Spiral Model. In: Tutorialspoint [online]. tutorialspoint [cit. 2023-03-26]. Dostupné z: https://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm#

[21] KOCHAR, Aneesha. Iterative Development: A Starter's Guide. In: DistantJob [online]. Tech Insights, 2021 [cit. 2023-03-26]. Dostupné z: https://distantjob.com/blog/iterative-development/

[22] ICSSP '20: Proceedings of the International Conference on Software and System ProcessesJune 2020 Pages 185–188 [cit. 2023-03-02]. Dostupné z: https://doi.org/10.1145/3379177.3390304

[23] ACM SIGSOFT Software Engineering NotesVolume 12Issue 1Jan. 1987 pp 35–37 [cit. 2023-03-02]. Dostupné z: https://doi.org/10.1145/24574.24576

[24] AST 2014: Proceedings of the 9th International Workshop on Automation of Software TestMay 2014 Pages 36–42 [cit. 2023-03-02]. Dostupné z: https://doi.org/10.1145/2593501.2593507

[25] Linmin Yang, Zhe Dang, and Thomas R. Fischer. 2011. Information gain of black-box testing. Form. Asp. Comput. 23, 4 (Jul 2011), 513–539. [cit. 2023-03-02]. Dostupné z: https://doi.org/10.1007/s00165-011-0175-6

[26] Frank Lammermann and Stefan Wappler. 2005. Benefits of software measures for evolutionary white-box testing. In Proceedings of the 7th annual conference on Genetic and evolutionary computation (GECCO '05). Association for Computing Machinery, New York, NY, USA, 1083–1084. [cit. 2023-03-02]. Dostupné z: https://doi.org/10.1145/1068009.1068193

[27] Ali Khalili, Massimo Narizzano, Armando Tacchella, and Enrico Giunchiglia. 2015. Automatic test-pattern generation for grey-box programs. In Proceedings of the 10th International Workshop on Automation of Software Test (AST '15). IEEE Press, 33–37 [cit. 2023-03-02].

[28] DOSHI, Kalpesh. Types of Testing: Different Types of Software Testing in Detail [online]. In: . Browsterstack, 2023, s. 1 [cit. 2023-08-01]. Dostupné z: https://www.browserstack.com/guide/types-of-testing

[29] Dudekula Mohammad Rafi, Katam Reddy Kiran Moses, Kai Petersen, and Mika V. Mäntylä. 2012. Benefits and limitations of automated software testing: systematic literature review and practitioner survey. In Proceedings of the 7th International Workshop on Automation of Software Test (AST '12). IEEE Press, 36–42 [cit. 2023-03-02].

[30] ALMYASHEV, Sergey. Test automation advantages and disadvantages [online]. In: . s. 1 [cit. 2023-08-06]. Dostupné z: https://zapple.tech/blog/test-automation-frameworks/test-automation-advantages-and-disadvantages/

Přílohy

1) Oskenované zadání práce

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Michael Šeda**
Studium:	I1900647
Studijní program:	B0688A140001 Informační management
Studijní obor:	Informační management
Název bakalářské práce:	**Srovnání testovacích nástrojů**
Název bakalářské práce AJ:	Test tool comparison

Cíl, metody, literatura, předpoklady:

Cíl: Porovnat testovací nástroje a doporučit, které používat

Osnova:

1. Úvod
2. Obecná problematika testování
3. Testing Toolbox
4. Představení vybraných nástrojů a jejich využití v praxi
5. Shrnutí výsledků a doporučení
6. Závěr

-

Zadávající pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	doc. Mgr. Tomáš Kozel, Ph.D.
Datum zadání závěrečné práce:	19.5.2022