MORAVSKÁ VYSOKÁ ŠKOLA OLOMOUC

Systémové inženýrství a informatika

Přístupy k automatizovanému testování GUI na platformě Android

BAKALÁŘSKÁ PRÁCE

Marta Kummerová

Vedoucí práce: PhDr. Jan Lavrinčík, Ph.D.

Olomouc 2021

PROHLÁŠENÍ

Prohlašuji, že jsem bakalářskou práci vypracovala samostatně a použila jen zdroje v seznamu literatury a použitých zdrojů.

Tištěná verze textu práce je shodná s textem práce na CD nosiči a elektronickou verzí vloženou do studijního systému IS/STAG.

V Olomouci dne 14. 4. 2021

Marta Kummerová

PODĚKOVÁNÍ

Děkuji svému vedoucímu doc. PhDr. Janu Lavrinčíkovi, Ph.D. za odborné vedení práce, za cenné rady a ochotu v průběhu zpracování této práce.

Obsah

Úvod

Pro zvolené téma bakalářské práce jsem se rozhodla, protože testování, konkrétně mobilních aplikací, bylo nedílnou součástí mé pracovní náplně. Převážně jsem testovala manuálně, proto jsem chtěla prohloubit své znalosti v oblasti automatického testování. Grafické uživatelské rozhraní na platformě Android jsem zvolila z toho důvodu, že aplikace pro Android jsou podle mého názoru více rozšířené mezi uživateli.

Testování softwaru je neodmyslitelnou a velmi důležitou součástí vývoje softwaru, která je často přehlížena. Při vývoji je nutné se s tímto tématem seznámit, protože kvalitní testování zlepší výsledný produkt a ušetří firmě čas i peníze, které by dali do náročnějšího vývoje softwaru.

Testování je velmi obsáhlý pojem, můžeme testovat desktopové aplikace, webové stránky a aplikace, mobilní zařízení. Tyto druhy se dají dále dělit. U testování webových stránek a aplikací je důležité mít nástroj, který podporuje testy na různých webových prohlížečích jako je Chrome, Safari, Edge, FireFox, Opera a Internet Explorer. U mobilních aplikací se zaměříme na platformu aplikace, nejpoužívanější jsou Android a iOS. Pro svou bakalářskou práci jsem se zaměřila na mobilní aplikace Android.

Dílčí cíle práce jsou definovat základní problematiku testování. Objasnit pojmy týkající se testování softwaru a testovacích prostředí. Nastínit historii a základy vývoje mobilních aplikací Android. Popsat, co obnáší automatické testování softwaru.

Hlavním cílem je ověřit přístupy automatického testování GUI na platformě Android pomocí různých nástrojů, které najdu po prozkoumání trhu, a porovnání jejich vlastností. Ze zmíněných nástrojů dále vyberu dva, které projdu detailněji a aplikuji na ně automatické testy. Zvolené nástroje vyberu podle množství uživatelů a podobných funkcí. Pro ukázku automatických testů vytvořím základní mobilní aplikaci o dvou obrazovkách, které budou obsahovat textové pole, validace a tlačítka.

1 Modely vývoje softwarového projektu

Začátky vývojových modelů se datují od 50. letech minulého století, kdy programy vznikaly modelem Napiš a oprav[1]. Princip spočíval v tom, že se napsal kód, aplikace se implementovala do provozu, a dále se ladily a opravovaly chyby. Zmíněný přístup vznikl velmi přirozeně, obsahoval mnoho chyb, a mohl tak sloužit jen pro jednoduché aplikace. Kvůli narůstající chybovosti a komplexnosti softwarů bylo jasné, že tento model není déle udržitelný.

1.1 Vodopádový model

Mezi první oficiální modely patří model vodopádový, který v roce 1970 představil doktor Winston Royce[2]. Je složen z po sobě jdoucích etap: Požadavky, Návrh, Vývoj, Implementace, Verifikace, Provoz a Údržba. Model je velmi jednoduchý a lze ho použít na projektu, který má podrobně promyšlený koncept. Samotný model má totiž jednu velkou nevýhodu, testovací etapa se nachází téměř na konci modelu, proto následná oprava chyb je velmi náročná a nákladná jak časově, tak finančně. V dnešní době je model neefektivní a nepoužívá se velmi často.

1.2 Spirálový model

Dalším přelomovým modelem je spirálový model, který zavedl Barry Boehmem v roce 1986[3]. Je založen na řízení rizik, to znamená analyzovat a eliminovat možné problémy, které se mohou s dalším vývojem vyskytnout. Každý jednotlivý cyklus je zakončen opakovanou analýzou rizik a možných problémů. Analýza ovlivňuje další fáze a rozvoje. Chyby a problémy se díky analýze rizik odhalí dříve, takže jejich následná oprava probíhá rychleji.

Spirálový model reaguje lépe než vodopádový na požadavky zákazníka, které se v průběhu vývoje mohou měnit, avšak změny požadavků se uskutečňují až po ukončení cyklu. Model je vhodné použít na složitých a rozsáhlých projektech.

[1] KADLEC, Václav. Agilní programování: metodiky efektivního vývoje softwaru. Brno: Computer Press, 2004. s. 33.
[2] KADLEC, Václav, ref. 1, s. 34.
[3] PATTON, Ron. Testování softwaru. Praha: Computer Press, 2002. s. 30

1.3 Přírůstkový model

Princip modelu je omezit rizika tím, že se projekt rozdělí na menší segmenty(přírůstky), a tak se zjednoduší zavedení změn do projektu. Provádí se série menších vodopádů. Každý z přírůstků je pak součástí evoluce celého vývoje projektu, který je také nazván jako evoluční.

1.4 Agilní metodiky

Agilní metodiky byly sepsány v manifestu roku 2001 s názvem The Agile Manifesto a vychází ze dvou základnách tezí[4]:

- Přijmout a umožnit změnu je efektivnější než se ji snažit zabránit,
- připraven reagovat na nepředvídatelné události.

Agilní přístup umožňuje rychlý vývoj softwaru a zároveň dokáže velmi dobře reagovat na změny požadavků od zákazníka v průběhu vývoje. Základní princip agilní metodiky je úzká spolupráce vývojového týmu a zákazníka. Software se průběžně předává zákazníkovi, který tak má možnost se podílet na dalším vývoji. Tato metodika je velmi rozšířená v IT firmách, protože v mnoha případech vede k větší spokojenosti zákazníka. Je to specifický přístup k řízení a zkoordinování celého projektového týmu a samotného projektu.

1.4.1 Metoda SCRAM

Jedna z nejrozšířenějších agilních metod je metoda *SCRAM*, která začala vznikat roku 1995[5]. Její představitelé jsou Ken Schwaber a Mike Beedl, kteří se snažili vyvinout flexibilnější metodiku vývoje softwaru, která lépe reaguje na změny požadavků a zlepší produktivitu vývoje.

V metodě SCRAM má projektový tým rozdělen role. Product owner hájí zájmy a požadavky zákazníka, zařizuje přenos informací mezi zákazníkem a vývojovým týmem, a tím koordinuje vývoj projektu. Vývojový team je tvořen jedinci na různých pozicích: analytik, grafik, programátor, tester. Scrum master odpovídá za spolupráci uvnitř týmu, rozhoduje o neshodách při vývoji softwaru a snaží se udržet pozitivní atmosféru celého týmu.

[4] PATTON, Ron. Testování softwaru. Praha: Computer Press, 2002. s. 30.

[5] KADLEC, Václav. Agilní programování: metodiky efektivního vývoje softwaru. Brno: Computer Press, 2004. s. 120.

Vývoj softwaru pomocí metody SCRAM se skládá ze Sprintů, což je základní jednotka vývoje scram[6]. Délka sprintu se ve firmách může lišit, někdy trvá týden, může však trvat i měsíc a opakuje se v pravidelných cyklech. Každý sprint začíná plánovací schůzí, kde si celý vývojového týmu určí, jaké backlogy se za daný sprint splní. Backlog je větší úkol, který je v rámci projektu nutno splnit. Backlog se rozpadne na menší části nazvané Tasky. Tasky jsou lépe splnitelné, obsahují popis úkolu, předpokládaný čas pracnosti. Po dokončení úkolu se task uzavře, doplní se popis řešení a reálně strávený čas pracnosti. Jakmile jsou všechny tasky uzavřené, na další plánovací schůzce se uzavře celý backlog.

Celý vývoj je doprovázen denními scram meetingy (stand-upy). Stand-up je rychlá denní schůzka, která se koná na stejném místě ve stejnou dobu, kde si členové týmu sdělí denní náplň práce. Neměla by trvat déle jak 10-15 minut.

1.5 Rational unified proces a Unified software development process

Rational unified process je rozsáhlá objektově orientovaná iterativní metodika vývoje softwaru[7]. Byla popsána v roce 1999 Ivarem Jacobsonem, Gradym Boochem a Jemesem Rumbaughem.

Metodika RUP zavádí 4 fáze vývoje a každá fáze se skládá z jedné nebo více iterací. První fází je Počáteční fáze (zahájení), kdy se stanoví cíle projektu, harmonogram projektu (plán iterací), odhad nákladu a definice rizik. Součástí této fáze může být návrh modelu nebo prototyp systému. Počáteční fáze končí rozhodnutím, zda je projekt za daných požadavků možno realizovat. V druhá fázi Rozpracování je definování architektury systému a vytipovaní komponent, které bude opakovaně potřeba. Zde již vzniká prototyp systému. V Konstrukční fázi je návrh, vývoj a testování systému. Poslední fáze je Nasazení, kde očekáváme funkční systém. Součástí této fáze je i školení uživatelů. Přechod mezi jednotlivými fázemi označujeme jako milníky.

Unified software development process zkráceně Unified proces (UP) je úzce propojena s jazykem UML a s metodou RUP má mnohé společné, je založena na stejných myšlenkách a principech[8]. Obsahuje stejné fáze, milníky i procesy. Jediná její odlišnost

[6] ŠOCHOVÁ, Zuzana a KUNCE, Eduard. Agilní metody řízení projektů. Brno: Computer Press, 2014. s. 43.

[7] BUCHALCEVOVÁ, Alena. Metodiky vývoje a údržby informačních systémů: kategorizace, agilní metodiky, vzory pro návrh metodiky. Praha: Grada, 2005. s. 36.

[8] KADLEC, Václav. Agilní programování: metodiky efektivního vývoje softwaru. Brno: Computer Press, 2004. s. 110.

spočívá v tom, že není propracovaná do stejné hloubky jako RUP a tvoří tak otevřený standard.

2 Mobilní aplikace

K mobilním aplikacím je možné vypracovat celou bakalářskou i diplomovou práci, proto zmíněnou kapitolu pojmu stručně a popíši důležité části, které se mobilních aplikací týkají. Zaměřím se na historie mobilních zařízení, historie systému Android, popis architektury systému Android. Dále objasním, jak se aplikace Android vyvíjí a z čeho se aplikace skládá. Nakonec

2.1 Historie mobilních telefonů

První bezdrátový telefon byl sice vynalezen v 19. století, ale první telefony pro veřejnost se začaly objevovat až v 50. letech dvacátého století v USA jako nová výbava automobilů. Plně přenosné telefony se objevily v průběhu 70. let a očekávala se nízká popularita. Kvůli poklesu cen tarifů a dostupnosti došlo k obrovskému nárůstu uživatelů. První mobilní telefon sestavila značka Motorola roku 1983.

2.2 Historie systému Android

Společnost Android Inc. Byla založena v Kalifornii v říjnu 2003 Andym Rubinem, Richem Minerem, Nickem Searsem a Chrisem Whitem. Roku 2005 byla Android Inc. odkoupena společností Google Inc., a tak se stala dceřinu společností. Po odkupu tým Googlu vyvinul platformu založenou na Linuxovém jádře a později získal několik patentů v oblasti mobilních technologií. V roce 2007 bylo vytvořeno uskupení Open Handset Alliance s cílem vyvinout standart pro mobilní zařízení. Ve stejný den byl ohlášený produkt Android[9].

2.3 Architektura systému

Architektura systému se skládá z několika důležitých vrstev, které detailněji rozeberu níže. Architekturu budu popisovat „zdola nahoru“ od nejnižší vrstvy po vnější. Nejnižší vrstvu tvoří jádro Linux, následují knihovny a Android Runtime, nad těmito

[9] UJBÁNYAI, Miroslav. Programujeme pro Android. Praha: Grada, 2012. s. 14.

vrstvy je Aplikační Framework a poslední horní vrstva je tvořena aplikací, kterou používají koncový uživatelé.

2.3.1 Linux

Základním pilířem, nejnižší vrstvou architektury Androidu, je upravené jádro operačního systému Linux. Úpravy se týkají redukce funkcí a jejich přizpůsobení možnostem mobilních zařízení. Jádro slouží k přímé interakci s hardwarem mobilního zařízení[10]. Zabezpečuje správu paměti, správu procesů, základní síťovou vrstvu a ovladače. Řízení procesů umožnuje, aby běželo více procesů současně. Na úrovni jádra je i zabezpečení systému, správa napájení, vstupní a výstupní operace a základní grafika. Podle hardwarové konfigurace jsou na této úrovni i moduly ovladačů pro Bluetooth, EDGE, 3G, Wi-fi, fotoaparát, GPS, kompas, akcelerátor, modul rozhlasového přijímače. Pro bezpečnější komunikaci a sdílení dat je na jádro implementován meziprocesorový ovladač Binder, který slouží jako zprostředkovatel komunikace. Součástí jádra je i správa napájení, která zabezpečuje, aby se energeticky náročnější moduly jako procesor a nabíječka, při delší nečinnosti vypínaly.

2.3.2 Knihovny a Android Runtime

Další vrstvou jsou knihovny, které poskytují přímý přístup aplikací ke komponentám systému Android. Jsou to nativní knihovny psané v jazyce C nebo C++. Tvoří tak mezivrstvu mezi linuxovým jádrem a komponenty vyšších vrstev.

Další vrstva Android Runtime do verze 4.3 obsahuje aplikační virtuální stroj Dalvik, který byl vyvíjen od roku 2005. Dalvik obsahoval licenční práva pro jazyk Java a změnu ve formě úspory energie a výkonu. Ve zmíněné Android Runtime vrstvě jsou tedy navíc knihovny programovacího jazyka Java.

Od verze 4.4 včetně se používá dopředni kompilace (překlad jazyka), a to z důvodů úspory energie a zrychlení aplikací. Tato verze přinesla prodloužení výdrže telefonu o 36 %.

Od verze 6.0 přichází změna v oblasti oprávnění aplikací. Systém Android umožňuje aplikace nainstalovat a spouštět jen s právy, které nezbytně potřebuje nebo instalaci zcela odmítnout. Aplikace si může v průběhu své aktivity dodatečně požádat o povolení práv, ale uživatel ji může odmítnout.

[10] LACKO, Ľuboslav. Vývoj aplikací pro Android. Brno: Computer Press, 2015. s. 59.

2.3.3 Application Framework a Aplikace

Vrstva Application Framework je pro vývojový tým nejdůležitější. Poskytuje přístup k základním službám systému a obsahuje například[11]:

- Packed Manager, což je modul pro správu balíčků (databáze), který udržuje seznam všech nainstalovaných aplikací v daném zařízení,
- Window Manager spravuje okna, která tvoří aplikaci. Z pravidla to bývá dvě a více oken současně,
- View Systém spravuje prvky grafického uživatelského rozhraní, jako jsou ikony, tlačítka a mnohé další,
- Activity Manager se stará o životní cyklus aplikace.

Poslední, a tedy nejvyšší úroveň architektury operačního systému Android, je tvořena ze samotných aplikací. Aplikační vrstvu využívají i koncový uživatelé.

2.4 Stručná historie verzí Android

V září roku 2008 v USA byla vydána první verze Android 1.0. Další jednotlivé verze systému se jmenují abecedně podle sladkostí (Aple pie, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, KitKat, Lollipop, Marshmallow, Nougat, Oreo a Pie)

2.5 Základní části aplikace Android

Aplikace pro Android jsou tvořené ze 4 základních komponent, a to jsou aktivity, services, content providers a broadcast reciever.

Activita je hlavní třída, která se zobrazí uživateli po spuštění aplikace. Aplikace má obvykle více activit a ty umožnují uživatelům přijímat informace přes grafické uživatelské prostředí. Přes aktivity uživatel aplikaci může ovládat.

Services neboli služby realizují operace a operace na pozadí. Obstarávají i spolupráci se vzdálenými procesy. Služby nejsou viditelné na uživatelském rozhraní a běží na jeho pozadí. Pokud je spuštěno více služeb, mohou probíhat i paralelně.

Content providers jsou poskytovatelé obsahu, kteří umožňují ukládat a sdílet data mezi více aplikacemi a procesy.

[11]LACKO, Ľuboslav. *Vývoj aplikací pro Android*. Brno: Computer Press, 2015. s. 60-61.

Broadcast Receivers je komponenta sloužící k oznámení. Tato komponenta čeká na pozadí a podle určení reaguje, například oznámení o nízkém stavu baterie.

2.6 Vývoj mobilních aplikací

Oficiálně podporované vývojové prostředí pro aplikace Android je Eclipse. Eclipse je open-source vývojové prostředí (IDE – Integrated Development Enviroment) určený primárně k programování v jazyce Java. Jeho flexibilita však dovoluje nainstalovat doplňky i pro další programovací jazyky jako je PHP, Python, C++, Ruby a další[12]. V dnešní době je možné použít i jiné vývojové prostředí, které nabízí možnost vývoje mobilních aplikací.

2.6.1 Android SDK

Vývojářský balík android SDK obsahuje nástroje pro vývoj aplikací pro platformu Android. Je dostupný pro všechny hlavní operační systémy jako je Linux, Windows i macOS. Sada SDK je rozdělena na tři druhy a to základní, doporučená a plná konfigurace vývojového prostředí.

Základní konfigurace, která je nezbytná pro vývoj aplikací, obsahuje SDK Tools, SDK Platform-tool a Android SDK platforms. SDK Tools obsahuje nástroje pro debugging(ladění), testování aplikace, správu Android Virtual Devices (AVD), Android emulátor, analýzu grafického layoutu a další programy. SDK Platform-tool obsahuje další důležité nástroje pro vývoj aplikací, které jsou závislé na verzi platformy a aktualizují se při vydání nové verze SDK. Android SDK platform se skládá z knihoven, systémového obrazu, ukázkových kódů, vzhledů emulátoru a ostatních zdrojů.

Doporučená konfigurace SDK obsahuje USB Driver, příklady kódů a dokumentaci. USB Driver je komponenta, která je nutná při ladění a testování aplikace nainstalované na zařízení. Nutná je pouze pro platformu Windows. Příklady kódů jsou ukázkové kódy aplikací, které jsou aktuální pro každou platformu. Dokumentace obsahuje lokální kopii dokumentace pro aktuální Android Framework API. Dokumentace je využita i ve vývojovém prostředí Eclipse.

Plná konfigurace SDK obsahuje Google API a ostatní SDK platformy. Google API jsou knihovny, které zpřístupní rozhraní Google Maps.

[12] LACKO, Ľuboslav. Vývoj aplikací pro Android. Brno: Computer Press, 2015. s. 17.

2.6.2 Emulátor

Výhodou a zároveň nevýhodou Androidu je variabilita různých zařízení s různými verzemi systému a různým rozlišením displeje[13], proto je emulátor nezbytnou výbavou každého vývojáře. Většina aplikací se chová v emulátoru stejně jako na fyzickém zařízení. Existují ovšem výjimečné situace, které se virtualizovat nepodaří. Emulátor operačního systému je obsažen v balíčku Android SDK.

2.7 Distribuce aplikace

Primárně se aplikace s operačním systémem distribuují pomocí Google Play, což je služba určená pro stahování aplikací a her, kterou provozuje přímo Google. Pokud vývojáři chtějí nahrát aplikaci do služby Google Play, musejí dodržet podmínky Distribuční smlouvy pro vývojáře Google Play[14]. Android je otevřenou platformou, proto je možné na ni nahrávat aplikace i přímo např. z počítače (postupy jsou zdokumentované a podporuje je i samotné Android SDK).

3 Testování softwaru

Testování softwaru jde nazvat výzkumem kvality produktu. Cílem je zjistit nebo ověřit informace. Samotné testování jde zahrnout do oblasti ověřování a plánování kvality, proto má obrovské použití v celém vývojovém procesu aplikace. Testy se vždy tvoří na míru daného projektu, kdy se stanový vize a cíle testování. Velmi často se stává, že důležitost testování je ve firmách opomíjená. Avšak testování zlepšuje kvalitu vyvíjeného softwaru a ušetří firmě čas i peníze, které by museli investovat do pozdějších oprav chyb.

3.1 Základní pojmy

Podkapitola základních pojmů objasní pojmy týkající se problematiky testování softwaru. Například co znamená pojem software, co je to Framework, co obnáší pozice testera, k čemu slouží negativní chyby a proč vytvářet use casy. Pochopení zmíněných pojmu nám usnadní následné porozumění testovací problematiky.

[13] LACKO, Ľuboslav. Vývoj aplikací pro Android. Brno: Computer Press, 2015. s. 22.
[14] GOOGLE. Distribuční smlouva pro vývojáře Google Play. Citováno 22. 01. 2021.

Software je soubor počítačových programů, který provádí nějakou činnost[15]. Můžeme jej rozdělit na systémové nebo aplikační. Systémové programy zajišťují chod počítače. S aplikačními programy pracuje uživatel. Obecně lze software definovat jako vše, co tvoří fungující počítač a není hmatatelné. Většina softwarů jsou prospěšná, avšak jsou i škodlivé softwary, které napadají počítače, například v podobě virů.

Framework neboli česky aplikační rámec je softwarová struktura sloužící jako podpora při vývoji i testování softwarových projektů. Může obsahovat knihovny, návrhové vzory a další dokumentace. Framework usnadňuje vývoj a testování pomocí nadefinované struktury a základních komponent. Například .NET Framework pro jazyk C# nebo JUnit pro unit testy v jazyce Java.

Tester je člen vývojového týmu, který má na starosti testování produktu. Náplň této pozice většinou obnáší tvorbu testovací strategie, přípravu testovacích scénářů, návrh postupu testování a tvorbu testovacích scénářů. U větších projektů, které jsou podstatně komplexnější, je vyžadována na pozici Testera lepší znalost IT oblasti a programování, to však není pravidlem u těch menších.

Projekt je zjednodušeně zakázka. Na projektu se podílí celý vývojový tým. Každý projekt má životní cyklus, kdy se shromáždí požadavky, vytvoří se návrh, dále se produkt začne vyvíjet a testovat, poté se implementuje a předá zákazníkovi.

3.1.1 Chyba

Pro selhání softwaru je celá řada označení a jedna z nich se chyba[16]. Dá se říct, že celé testování stojí na chybách, a to na jejich odhalení nebo na ověření, že nejsou. Chyby mohou být jak softwarové, tak technické. Zjednodušeně lze nazvat softwarovou chybou veškeré chování aplikace, které se liší od analýzy projektu nebo neodpovídají testovacím scénářům.

Dalším pojmem jsou *pozitivní a negativní testy*. Pozitivní testy nebo testy splnění mají správné a korektní vstupy a očekávaný výstup je bezchybný průchod aplikací. Negativní testy též zvané jako testy selhání nebo testy vynucených chyb naopak mají

[15] SINGH, Yogesh. Software testing. New York: Cambridge University Press, 2012. s. 19.

[16] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. s. 149.

úmyslně chybný vstup a ověřují, zda systém vhodně reaguje. Vhodnou reakcí je například chybová hláška nebo upozornění na nevhodný vstup[17].

Use case neboli případ užití je popis postupu, jak uživatel aplikaci používá a jak aplikace reaguje. Podrobně sepsaný Use case je velkou výhodou v dalších fází projektu. Slouží k tvorbě test casů a k sepsání uživatelské příručky. Use case lze zpracovat textově nebo pomocí diagramu, nejčastěji se používá kombinace obou.

Grafické uživatelské rozhraní (GUI) je část aplikace, která se zobrazí uživateli na obrazovce nebo displeji. Uživatel pomocí aktivit, například kliknutí myší nebo vyplnění pole, ovládá samotný program. Grafické rozhraní lze pochopit jako komunikaci uživatele se softwarem.

Výsledek se zobrazí po provedení testu a nabývá hodnoty prošel nebo neprošel. Pokud test prošel výsledek je úspěšný, test nenašel žádnou chybu a testovaná část je v pořádku. Pokud test neprošel, v průběhu testu nastala chyba, to může mít více vysvětlení, které se odhalí při analýze chyb.

Report je nahlášení chyby, která byla nalezena v software, vývojovému týmu.

3.2 Dělení testů

Testy lze rozdělit podle více kritérií. Nejobecněji jde dělení popsat podle toho *co* testuji, *proč* testuji, s jakým *cílem* testuji a *jak* testuji. V části „co“ testujeme konkrétní věc jako kód a grafické uživatelské rozhraní. V „proč“ hledáme důvod, například jestli je aplikace připravená k dalšímu testování, komunikace mezi moduly funguje bezchybně a jestli aplikace funguje jako celek. V předposlední části testujeme s daným cílem, například zjištění spokojenosti zákazníka nebo ověření bezpečnosti aplikace. V poslední části „jak“ si zvolíme způsob, jakým budeme testovat, manuálně nebo automaticky.

3.2.1 Unit testy

Unit testy také zvané jednotkové testy jsou tvořeny programátory v průběhu vývoje softwaru a používají metodu testování bílé skřínky[18], kterou detailněji rozeberu v další kapitole. Jednotkou testu se nazývá samostatná testovaná část programu, která je nezávislá na ostatních. Testy ověřují systém na nižší úrovni, na úrovni kódu. Výhoda unit testů

[17] HEROUT, Pavel. Testování pro programátory. České Budějovice: Kopp, 2016. s. 27.
[18] GOUSSET, Mickey a POKORNÝ, Jan. Řízení životního cyklu aplikací ve Visual Studiu 2010. Brno: Zoner Press, 2010. s. 126.

spočívá v odhalení chyb v kódu již při psaní testů, to usnadňuje pozdější opravy. Unit testy mohou sloužit jako dokumentace kódu, takže není problém projekt předat jinému programátorovi. Při dalším rozvoji softwaru a zásahu do kódu lze testy použít jako regresní.

3.2.2 UI testy

User interface tests lze přeložit do češtiny jako testy uživatelského rozhraní jinak také známé jako GUI testy, testy grafického uživatelského rozhraní. Zníněné testy jsou velmi často používané pro testování mobilních aplikací. Kontrolují hlavně grafickou stránku aplikace, která je pro uživatele velmi zásadní. Simuluje uživatelské aktivity jako je kliknutí na tlačítko nebo vyplnění textového pole[19]. Lze je ovšem použít i pro kontrolu správné funkcionality softwaru. Například mohu zkontrolovat nejen jestli se grafický prvek nachází na obrazovce, ale i jestli má správnou hodnotu.

3.2.3 Smoke testy

Smoke testy ověří, zda je software, aplikace připravena na testování. Aplikace musí jít nainstalovat, spustit a je nakonfigurovaná pro konkrétního zákazníka. Poté se aplikace předá k dalšímu testování. Testy často používají v následujících situacích[20]:

- po odstávce systému,
- po nasazení systému do produkce,
- před prezentací zákazníkovi nebo veřejnosti,
- po změně testovacího prostředí,
- pro průběžnou kontrolu testovacího prostředí.

3.2.4 Integrační testy

U integračních testů můžeme rozlišovat integraci vnitřní a vnější. Testování vnitřní integrace ověřuje správné propojení částí aplikace nebo jednotlivých modulů. Vnější integrační testování kontroluje správnou komunikaci jednotlivých modulů, které jsou propojené do jednoho celku a tvoří tak aplikaci. Je mnoho strategií, jakým způsobem tvořit

[19] AMMANN, Paul a Jeff OFFUTT. Introduction to software testing. New York: Cambridge University Press, 2008. s. 261.

[20] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. s. 175.

testy, pro ukázku popíšu dva nejznámější, a to integraci pomocí stromového diagramu, kdy od hlavního uzlu procházím stromem až po koncové listy a naopak. Oba způsoby mají své výhody a nevýhody, avšak způsob od uzlu k listům bývá oblíbenější[21].

3.2.5 Systémové testy

Systémové testy kontrolují chování aplikace jako celku. Zaměřují se i na smysl systému, zda aplikace plní požadavky, pro které byla vytvořena. Systémové testy jsou situovány do pohledu zákazníka a tvoří se na základě testovacích scénářů. Většinou probíhají ve více kolech, kdy se nalezené chyby odstraní a testuje se znovu. Systémové testy slouží jako výstupní kontrola před předání softwaru zákazníkovi.

3.2.6 Akceptační testy

Akceptační testy probíhají na straně zákazníka. Pokud se během vývoje a interního testování neobjevily nebo opravily větší nedostatky, aplikace se může předat zákazníkovi. Zákazník testuje, zdali jsou splněné všechny domluvené požadavky. Samotné testování probíhá na testovacím prostředí u zákazníka a nedostatky se reportují zpátky na vývojový tým. Protože report provádí zákazník, musejí být nedostatky opraveny v nejkratším možném čase.

3.2.7 Regresivní testy

Regresní testování je typ testování softwaru, který ověřuje, že software, který byl dříve vyvinut a testován, stále funguje stejným způsobem i po jeho změně nebo propojení s jiným softwarem[22]. Změny mohou zahrnovat vylepšení softwaru, opravy, změny apod. Během testování regrese mohou být odhaleny nové softwarové chyby nebo regrese. Někdy se provádí analýza dopadu změn softwaru, která určuje, které oblasti by mohly být navrhovanými změnami ovlivněny.

3.2.8 Bezpečnostní testy

Zabezpečení dat je nedílnou součástí vývoje aplikace. Je velmi důležité ochránit citlivé informace před hrozbami, hlavně pokud aplikace komunikuje přes internetové připojení[23]. U bezpečnostních testů se často používá modelování hrozeb, kdy se vyhodnotí

[21] SINGH, Yogesh. Software testing. New York: Cambridge University Press, 2012. s. 372-273
[22] SINGH, Yogesh. ref. 21 s. 336.
[23] SINGH, Yogesh, ref. 21, s. 471.

architektura systému a vytipují riziková místa. Déle se provedou takzvané penetrační testy, kterými se simulují možné útoky a prověří se, do jaké míry je systém zabezpečený. Pro provádění testování musí mít tester potřebné znalosti z oblasti bezpečnosti softwaru.

3.2.9 Výkonnostní testy

Výkonnostní testování zjišťuje, jak aplikace v průběhu procesu pracuje a jak je zatížená, proto do oblasti výkonnostních testů spadají testy zátěžové. Často se výkonnostní testy měří pomocí času, to znamená za jak dlouho proběhl daný proces. Zmíněný proces se nazývá test pomocí stopek a vznikl před mnoha lety, kdy testeři opravdu seděli před obrazovkami se stopkami v ruce[24]. Cílem výkonnostního testu je určení významných a důležitých „úzkých hrdel“, kdy aplikace nemá výkon, jaký je očekáván.

3.2.10 Zátěžové testy

Do výkonnostního testování spadají i zátěžové testy, kdy se testuje, jak se aplikace chová při normálním a extrémním zatížení. Zjednodušeně řečeno se snaží nalézt slabá místa aplikace. Zátěžové testování je velmi komplexní pojem, skrývá se pod ním mnoho druhů, jak systém zatížit[25].

Obecně zátěžové testování staví na principu, že se aplikace extrémně zatíží a odhalí chyby, které se v normální situaci neprojeví.

Test maximální zátěže simuluje situaci během špičky provozu, dále se může zatížení zvyšovat, dokud se neodhalí hraniční hodnota zátěže, při které aplikace už nepracuje správně.

Měření střední doby poruchy zjišťuje průměrný čas mezi nalezením chyby nebo pádu aplikace.

Testování nedostatku zdrojů analyzuje, co se děje při pádu aplikace. Zjišťuje například co se stane s daty, pokud aplikace spadne a není dostatek místa na jejich uložení.

Testování kapacit slouží k otestování serverů a určení, kolik uživatelů dokáže server obsloužit.

Testování opakovaných požadavků je metoda, kdy ve smyčce opakují a spouštějí určité funkce, dokud není dosaženo určitého počtu opakování nebo dokud se neobjeví nežádoucí efekt.

[24] PAGE, Alan, JOHNSTON, Ken a ROLLISON, Bj. Jak testuje software Microsoft. Brno: Computer Press, 2009. s. 246.

[25] PAGE, Alan, JOHNSTON, Ken a ROLLISON, Bj. ref. 24. s. 249.

3.2.11 Manuální a automatické testování

Manuální testy jsou ručně prováděné testy, kdy uživatel prochází software podle příkladů užití. Tyto testy jsou vhodné pro menší projekty nebo pro jednotlivé rozvoje. Při manuálním testování většího a složitějšího softwaru, může dojít k přehlédnutí chyby, avšak může nastat situace, která se nedá pokrýt jinými testy. Jako každý způsob testování má své výhody a nevýhody, proto nejlepší možnost je zkombinovat manuální a automatické testování.

Automatické testy jsou testy užití přepsané do kódu, které se spustí a probíhají automaticky bez přítomnosti uživatele. Testy se dají aplikovat na desktopové, webové i mobilní aplikace. Podrobněji si automatické testování rozebereme v další kapitole.

3.3 Metody testování

Metody, jak přistupovat k testování softwaru, se mohou lišit v závislostech na fázi projektu a dalších okolnostech vývoje a jeho prostředí. Již dříve jsem zmínila, že je mnoho druhů testů například Unit testy, Integrační testy a další, taktéž je i více metod testování.

Metody nám umožňují pohlížet na aplikaci z více úhlů, a tak odhalit řadu různých chyb.

3.3.1 Statické a dynamické testování

Statické a dynamické testování se liší v tom, zda je software spuštěný nebo ne. Při statickém testování se chyby hledají v kódu a aplikace není spuštěná, tato metoda se často používá při úpravě a revidování kódu. Dynamické testování je dá se říct typické testování, kdy se software spustí a hledají se chyby v běžícím procesu. Automatické UI testy pracují principu dynamického testování.

3.3.2 Černá, bílá a šedá skříňka

Metody černé, bílé a šedé skříňky se zakládají na znalostech, které tester má ohledně testovaného softwaru. Testování černé schránky je bez jakýchkoli znalostí kódu a vnitřního fungovaní softwaru. Je to způsob, jakým testuje koncový zákazník a simuluje tak reálné způsob užití aplikace. Proti tomu testování bíle schránky je se znalostí kódu a funkcionalit, testování je tak důkladnější a otestují se tak i okrajové části aplikace, které neznalý uživatel může přehlédnout.

Šedá někdy nazvaná skleněná skříňka je přístup testování, kde jsou testy navrhnuty podle požadavků zákazníka a dále je aplikovaný postup testování bílé skříňky[26].

4 Prostředí vývoje softwaru a testů

Prostředí, ve kterém se software nachází, je ovlivněno fází vývoje softwaru. Každé prostředí má svůj název, zkratku, funkce, výkonnost, správu prostředí a integrovatelnost.[27]

Název je důležitou jedinečnou identifikací prostředí a vystihuje činnost, pro kterou je prostředí určeno. Sjednocení názvosloví je nezbytné pro komunikaci nejen v týmu, ale napříč celým oddělením a firmou.

Zkratka je taktéž důležitá pro komunikaci, ve většině případů je ve formě tří až čtyř znaků (písmen).

Funkce označuje aktivitu, kterou v daném prostředí provádíme. Často se stává, že jedno prostředí má i více funkcí.

Výkonnost je hodnota požadovaného výkonu prostředí. Produkční prostředí může být například nachystané na zátěž tisíců uživatelů, ale prostředí pro systémové testy bude pouze pro desítky uživatelů.

Správa prostředí obstarává instalace nových verzí aplikace a jejich provoz.

Integrovanost popisuje nutnost integrace s dalšími aplikacemi.

Vývojové a testovací prostředí se dělí na několik typů podle toho k čemu slouží a podle testů, které do nich spadají.

4.1 Pískoviště a vývojové prostředí

Pískoviště lze použít pro tvoření prototypu aplikace a ke zkoušení testovacích nástrojů[28]. K takovému prostředí poslouží vyřazený nevyužitý hardware nebo cloud.

Vývojové prostředí zkráceně DEV, z anglického slova development, slouží vývojovému týmu a zahrnuje nástroje pro automatické sestavení verzí, uložiště kódu s podporou verzování a ostatní nástroje pro vývoj aplikace[29]. Ve vývojovém prostředí

[26] PAGE, Alan, JOHNSTON, Ken a ROLLISON, Bj. Jak testuje software Microsoft. Brno: Computer Press, 2009. s. 249.

[27] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. s. 123.

[28] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. ref. 27. s. 175.

[29] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. ref. 27. s. 125.

z pravidla probíhají unit testy nebo jednotkové integrační testy. Výkonnost v tomto prostředí nehraje roli, protože k němu přistupuje jen omezený počet lidí.

4.2 Testovací prostředí pro systémové testy

Zkráceně nazvané System test (SYS). Do systémového prostředí se nasazují již hotové funkčnosti, které prošli jednotkovými testy[30]. Prostředí SYS plní dvě zcela odlišné funkce. Slouží pro finální systémové testy, které provádějí vývojový testeři. Opět pro zmíněné prostředí není podstatná výkonnost.

4.3 Testovací prostředí pro integrační testy

Pro akceptaci funkčnosti a integrační testy slouží integrační prostředí (INT). Zde se nasazují již hotové verze aplikace, které prošly systémovými testy[31], a následně se provádí integrační testy a uživatelské akceptační testy. Jedná se o fungující systém, který může obsahovat vazby na jiné moduly, proto se očekává od prostředí integrovanost. Výkonnost v integračním prostředí není vyžadovaná.

4.4 Testovací prostředí pro podporu produkce

Zkratkou PRS z anglického názvu production support má tři hlavní účely[32]:

- simulování chyb nalezených v produkčním prostředí,
- ověření a otestování nalezených chyb,
- testování drobných změn.

Zmíněné prostředí musí být integrované, protože odhaluje chyby i z jiných částí aplikace. Občas se prostředí pro podporu produkce spojuje do jednoho s prostředím předprodukčním. V tomto prostředí se již klade určitý důraz na výkonnost.

4.5 Předprodukční testovací prostředí

Právě v tomto prostředí se provádějí výkonnostní, zátěžové a bezpečnostní testy. Zkratka je z PRE ze slova preproduction. Integrovanost je plně vyžadována jak na úrovni

[30] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016, s. 125.

[31] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ, ref. 30, s. 125-126.

[32] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ, ref. 30, s. 126.

softwaru, tak hardwaru[33]. Předprodukční prostředí by se mělo co nejvíce podobat produkčnímu prostředí. Toto je finální prostředí před nasazení aplikace do produkce.

4.6 Produkční a záložní testovací prostředí

Toto prostředí simuluje finální aplikaci, která bude nasazena u zákazníka. S produkčním prostředím vzniká i záložní prostředí, které převezme jeho funkci v případě poruchy[34]. Zkratku má produkční prostředí PRO z anglického slova production a záložní prostředí BAC z anglického slova backup (záloha).

4.7 Školící prostředí

Podle názvu prostředí lze snadno poznat, že školící prostředí slouží ke školení uživatelů. Prostředí nese zkratku EDU od slova education (vzdělání). Výkonnost zde není vyžadována[35].

5 Automatizované testování

Účel automatických testů je usnadnit a zefektivni provádění manuálních testů. Zcela automatické testy prováděné bez zásahu lidského faktoru výrazně snižují možnost provedení chybného postupu při testován softwaru a tím také zvyšují pravděpodobnost bezporuchového provozu softwaru.

Automatické testy vznikají na základě kvalitně vypracovaných testovacích scénářů.

5.1 Rozdíl mezi manuálním a automatickým testování

Rozdílů, krom hlavního, že program simuluje uživatelský postup, je plná řada. Pro příklad uvedu několik z nich[36]:

- Automatické testy probíhají zadarmo a rychleji, takže se dají častěji provádět,
- postup prováděný automaticky je vždy stejný, postupuje podle kódu a nikdy neudělá chybu,

[33] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. s. 126.

[34] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. ref. 33. s. 126-127.

[35] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. ref. 33. s. 127.

[36] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. ref. 33. s. 178-179.

- neintuitivní chování, pokud se v aplikaci něco pozmění, automatický test situaci neumí vyřešit a tester tak musí test upravit,
- u automatického testu je omezená možnost rozboru chyby, propíše se jen řádek kódu, kde se test zastavil a proč se zastavil,
- pokud se test vyhodnotil jako chybný, tester musí postup projít znovu a chybu konkrétně definovat,
- při vytváření automatického testu vlastně test provádíme, takže se vyplatí následně pro regresivní testy,
- pro automatické testy GUI je potřeba funkční software,
- kvalifikace testera se velmi liší.

5.2 Výhody a nevýhody automatizovaných testů

Výhody a nevýhody automatických testů jsou vždy vhodné vztáhnout k danému projektu, konkrétně k dané oblasti. Každý projekt má své specifikace, proto se benefity mohou lišit na základě projektu.

Obecně největší výhodou automatických testů je časová úspora. Když se testy spouštějí a vyhodnocují automaticky, ušetří se čas pracovníka, který se tak může věnovat jiné aktivitě. Pokud se jedná o plnou automatizaci, testy se mohou spouště přes noc a nenarušovat tak denní činnost pracovníků.

Jako nevýhodu lze považovat neustálou údržbu automatických testů. Může nastat situace, kdy je do aplikace zanesena změna, a testy se tak musí ihned aktualizovat. Automatické testy ovšem nemusí odhalit všechny skryté chyby, proto se může stát, že se nějaká chyba dostane do provozu.

5.3 Výběr vhodných testů pro automatizaci

Ne všechny testy a testovací scénáře jsou vhodné k automatizaci, proto je důležité umět správně vybrat ty scénáře, které se pro automatizaci hodí. U výběru je možné se zaměřit na určité prvky jako například:

- testovací scénáře se často opakují,
- scénáře jde snadno automatizovat,
- jestli mají ekonomický přínos,
- neobsahuje příliš mnoho manuální přerušení,
- systém, na kterém chceme testy provádět, je funkční a stabilní,

- v testované části nejsou v budoucnu plánované větší změny kódu.

5.4 Test case

Test case neboli testovací scénáře[37], jsou sepsané postupy, vstupní data a podmínky, jak procházet aplikací při uživatelském testování. Postupně tak vzniká soubor průchodů, testují se i chybné průchody, kde čekáme od aplikace chybu. Testovací scénáře se zaměřují nejen na grafickou část aplikace, ale i na její funkcionalitu. Testovací scénáře mají pokrýt veškeré možnosti a situace, do kterých se aplikace může dostat.

U psaní scénářů je vhodné dodržet jednotnou strukturu a až následně tvořit automatické testy. Příklad struktury může být tabulka, kde máme název testu, vstupní data, data v procesu, popis průchodu (jednotlivé kroky), očekávaný výsledek.

5.5 Kostra automatizovaného testu

Kostra automatického testu odpovídá struktuře test casu a má dvě hlavní části. Inicializační část, ve které se test spustí a mohou se i provést prvotní kroky, které se opakují u všech testovacích případů a tělo testu. V těle testu je konkrétní případ, který testujeme s rozdílnými vstupy pro ověření případu. Tělo obsahuje podmínku, kterou musí test splnit, aby byl označený za správný. Na jednu inicializační část lze většinou připojit více testů najednou, avšak je vhodné mít propojené testy jednoho testovaného prvku. Tímto stylem si samotné testy kategorizujeme a jsou poté přehlednější.

6 Přehled nástrojů pro testování

6.1 Visual Studio 2019 a Xamarin.UItest cross-platform test

Je vývojové prostředí (IDE) od americké akciové společnosti Microsoft. Slouží pro vývoj konzolových aplikací a aplikací s grafickým rozhraní. Xamarin.UItest je testovací Framework v jazyce C#. Slouží pro testování grafického uživatelského rozhraní mobilních aplikací.

[37] SINGH, Yogesh. Software testing. New York: Cambridge University Press, 2012. s. 336.

6.2 Visual Studio 2019 a Test Studio

Test Studio je nástroj pro testování softwaru na bázi Windows pro testování funkcí webu a desktopů, testování výkonu softwaru, testování zátěže a testování mobilních aplikací vyvinutý společností Telerik. Test studio je samostatný program, který má plug-in pro Visual Studio[38], takže pokud je vývojový tým zvyklí na programování ve Visual studiu, může použít k testování rozšíření od Teleriku. Plug-in je podporovaný ve verzích 2012 a vyšší. Test studio mobile bohužel tento rok ukončil podporu testování mobilních aplikací a zaměřuje se více na webové aplikace.

6.3 TestComplete

TestCopmlete je program pro automatické testování vyvinutý společností SmartBear Software. Poprvé byl vydán v roce 1999 společností AutomatedQA. Poskytuje speciální funkce pro automatizaci testovacích akcí, vytváření testů, definování základních dat, spouštění testů a protokolování výsledků testů.[39] Obsahuje i funkci záznamu testů, kdy se spustí záznam a po provedení potřebných kroků se záznam převede na test. Pro uživatele je dobře ovladatelný. Navíc má funkci, která automaticky pořídí snímek obrazovky při objevení chyby. Lze využít k funkčnímu, regresivnímu, zátěžovému a unit testování. Dále testuje také webové a desktopové aplikace. Podporuje mobilní systémy Android i iOS a z webových prohlížečů podporuje Internet Explorer, Mozilla Firefox a Google Chrome.

6.4 Silk Test

Silk Test je nástroj pro testování od společnosti Segue Soft. Od roku 2009 patří pod společnost Micro Focus International. Různé verze Silk Testu podporují různé jazyky a vývoje, například Silk4Net podporuje C#, Silk Test Workbench automatizuje testy na vizuální úrovni a Silk4J umožňuje v prostředí Eclipse použít jazyk Java. Silk test umožnuje automatické testy mobilních pro Android i iOS a webových aplikací[40] na prohlížečích Firefox, Chrome, Edge, Safari a Internet Explorer.

[38] PROGRESS SOFTWARE CORPORATION. GUI Testing in Visual Studio [online]. 2021. [cit. 30.3.2021].

[39] SMARTBEAR SOFTWARE. Automated Testing. [online]. 2020. [cit. 30.3.2021].

[40] MICRO FOCUS. Silk Test 21.0: Testing Mobile Applications. [online]. 2020 [cit. 30.3.2021].

6.5 Ranorex

Ranorex je Framework poskytující GUI automatické testy od společnosti Ranorex GmbH. Ranorex slouží k testování desktopových, webových a mobilních aplikací. Zníněný program nabízí testovací záznam a následné vytvoření testu. Uživatelsky je program velmi přívětivý a přehledný. Pro psaní testů v kódu nebo ze záznamu lze použít Ranorex studio[41], které nabízí různé druhy testů.

6.6 Eclipse a Jubula

Jubula poskytuje automatické testy GUI pro více druhů aplikací[42]. Jubula je projekt vývojového prostředí Eclipse. Lze ho použít i pokud nemáte zkušenosti s programováním. Jubula používá metodu černé skříňky, takže není nutné mít k dispozici kód aplikace. Jubula reportujte výsledky testů do systémů pro správu životního cyklu aplikací a při nalezení chyby pořídí automaticky snímek obrazovky.

6.7 Ascential Test

AscentialTest je podnikový systém správy testů, který zahrnuje plánování testů, vývoj, správu dat, hledání a nahlášení defektů[43], vyvinutý společností Zeenyx Software. Manuální a automatické testy jsou tvořeny pomocí před chystaných komponent, které se seřazují postupně podle test casů ve virtuální testovacím editoru.

6.8 Watir a SikuliX

Watir je open-source knihoven Ruby pro testování webových aplikací. Bohužel nepodporují testování mobilních aplikací[44]. Byl vytvořen Bretem Pettichordem a Paulem Rogersem.

SikuliX je nástroj, který testuje pomocí snímku obrazovky, automatizuje vše, co na obrazovce vidí. Lze využít i na rozpoznání textu na obrázku[45]. Podporuje nejen webové aplikace, ale i mobilní.

[41] RANOREX GMBH. Easy-to-use tools for beginners and experts. [online]. 2021. [cit. 30.3.2021].
[42] ECLIPSE FOUNDATION. Jubula. [online]. [cit. 30.3.2021].
[43] ZEENYX. AscentialTest® Features. [online]. [cit 30.3.2021].
[44] WATIR. Watir is…. [online]. 2021. [cit. 30.3.2021].
[45] RAIMUND HOCKE. What is SikuliX? [online]. 2017. [cit. 30.3.2021].

6.9 EggPlant

EggPlant mobilní testovaní vzniklo v roce 2002 společností RedStone, kterou v roce 2008 získala firma EggPlant. Nástroj EggPlant testuje mobilní aplikace se systémem iOS, Android, Windows Phone a BlackBerry. EggPlant testuje mobilní aplikace pomocí mobilních brán (Mobile Gateways), které umožňují přímé a automatické ovládaní zařízení[46].

6.10 Appium

Je volně přístupný testovací Framework, který si zakládá na tom, že není nutné mít kód aplikace k dispozici a automatizuje testy z jakéhokoli jazyka[47]. Vhodné pro mobilní nebo webové aplikace. Appium bylo vyvinuto v roce 2011 a od roku 2016 spadá pod společnost JS Foundation.

6.11 T-plan Robot

Je nástroj od společnosti T-Plan využívající testovací metodu černé skříňky pro automatizaci mobilních, webových a desktopových aplikací[48]. T-plan Robot je založený na jazyce Java, ale lze použít aplikace i v jiných jazycích jako C# nebo C++. Nástroj T-plan Robot používá technologii rozpoznání obrazu. Taktéž využívá záznam testů pro jejich automatizaci, a tak simuluje uživatelskou interakci s aplikací.

6.12 MonkeyTalk

Je volně dostupný nástroj pro automatické testování mobilních aplikací Android a iOS[49]. Je založený na jazyce Java a lze použít jako Plug-in do prostředí Eclipse. Navíc podporuje záznam a přehrání testu, což usnadňuje uživateli práci s programem. Navzdory velkému zájmu byla podpora aplikace ukončena.

7 Srovnání nástrojů

Ve srovnávací tabulce srovnávám základní prvky, které nástroje nabízejí, a to technologii a podporu, zpracovanou dokumentaci, dostupnost nástroje a jejich licence a poslední vydanou stabilní verzi.

[46] EGGPLANT. Mobile Gateways. [online]. 2020 [cit. 30.3.2021].
[47] THE JS FOUNDATION. Představujeme Appium. [online]. [cit. 30.3.2021].
[48] T-PLAN LIMITED. Overview. [online]. 2021. [cit. 30.3.2021].
[49] VASANTH K. How to do mobile app automation testing using MonkeyTalk tool? [online]. 2016. [cit. 30.3.2021].

Název	Technologie a podpora	Dokumentace	Dostupnost a licence	Nejnověj ší verze
Visual Studio, Xamarin. UITest	Veškerá podpora a doplňky Visual Studia, Android, iOS.	Ano, velmi dobře zpracovaná a dostupná na oficiálních stránkách docs.microsoft.com.	Licencovaná, trial verze zdarma.	Březen 2021
Telerik, Test Studio	HTML, AJAX, Silverlight, ASP.NET MVC, JavaScript, WPF, Java, Visual Studio, C#, Android, iOS.	Ano, velmi dobře zpracovaná a dostupná na oficiálních stránkách docs.telarik.com.	Licencovaná, trial verze zdarma.	Března 2021
Test Complete	Web, Windows, Android, iOS, WPF, HTML5, Flash, Flex, Silverlight. .NET, VCL, Java, Android, iOS.	Ano, velmi dobře zpracovaná a dostupná na oficiálních stránkách support.smartbear.com.	Licencovaná, trial licencovaná verze zdarma.	Březen 2021
Silk Test	.NET, WPF, Java, DOM, IE, Web, SAP Windows GU, Android, iOS.	Ano, velmi dobře zpracovaná a dostupná na oficiálních stránkách supportline.microfocus.com.	Licencovaná, Trial není k dispozici, pouze demo video.	Prosinec 2020
Ranorex	HTML, HTML5, JavaScript, .NET, WPF, Java, Android, iOS.	Ano, velmi dobře zpracovaná a dostupná na oficiálních stránkách ranorex.com/Documentation.	Licencovaná, trial verze zdarma.	Únor 2021
Jubula	Java, HTML, Android, iOS.	Ano, velmi dobře zpracovaná a dostupná na oficiálních stránkách eclipse.org nebo wiki.eclipse.org	Licencovaná. Základní verze bez podpory zdarma, s podporou placené.	Červenec 2020
Ascential Test	MS Windows, Internet Explorer, Firefox, Chrome, Java, .Net.	Na oficiálních stránkách není k dispozici.	Licencovaná, trial verze zdarma.	Duben 2020
Watir	Web, Windows, Internet Explorer, Firefox, Chrome.	Ano, velmi dobře zpracovaná a dostupná na oficiálních stránkách rubydoc.info/gems/watir	Volně dostupná	Duben 2021
SikuliX	Phyton, Ruby, JavaScript.	Ano, zpracovaná na oficiálních stránkách sikulix-2014.readthedocs.io	Volně dostupná	Březen 2021

EggPlant	Web, Mac, Windows, Linux, Android, iOS.	Ano, zpracovaná na oficiálních stránkách http://docs.eggplantsoftware.com/	Licencovaná bez trial verze	Březen 2021
Appium	Android, iOS, Selenium WebDriver, Web, Windows, Linux.	Ano, zpracovaná na oficiálních stránkách https://appium.io/docs/en/about-appium/intro/	Volně dostupná	Březen 2020
T-plan Robot	Windows, Mac, Linux, Unix, Solaris, Android, iOS.	Ano, zpracovaná na oficiálních stránkách docs.t-plan.com	Licencovaná, je nutné poslat žádost o trial verzi zdarma	2021

Tabulka 1: Srovnávací tabulka nástrojů[50]

8 Detailní seznámení s vybranými nástroji

Nástroje jsem vybrala na základě uživatelské základny, podobných funkcí a dostupnosti softwaru. Zvolila jsem Visual Studio 2019 s Xamarin.UITest a Ranorex studio. V kapitole popíši práci s nástrojem, jeho dostupnost a instalaci, prostředí softwaru, zakládání projektu a tvoření a správu automatických testů. Následně vyhodnotím výhody a nevýhody obou nástrojů.

8.1 Visual Studio 2019

Jak už jsem zmínila, je to vývojové prostředí pro konzolové aplikace a aplikace s grafickým rozhraní. Visual Studio obsahuje editor kódu podporující IntelliSense (viz. kapitola 8.1.1) a refaktorování. Integrovaný debugger pracuje jak na úrovni kódu, tak na úrovni hardwaru. Další nástroje zahrnují designer formulářů pro tvorbu aplikací s GUI, designer webový stránek, tříd a databázových schémat. Do Visual Studia lze přidávat rozšíření, což vylepšuje funkčnost celého nástroje. Dále podporuje jazyky prostřednictvím jazykových služeb, takže je možné programovat v jakém jazyce si zvolíte. Defaultní jazyky jsou C, C++, Visual Basic a C#, ostatní je nutno doinstalovat v jazykových službách.

[50] Zdroj vlastní

8.1.1 IntelliSense

IntelliSense je podpora dokončování kódu, která zahrnuje několik funkcí: seznam členů, informace o parametrech, rychlé informace a dokončování slov. Zmíněné funkce vám pomůžou získat další informace o kódu, který používáte, sledovat parametry, které píšete, a přidávat volání vlastností a metod pouze několika klávesami[51].

8.1.2 Historie Visual studia

Visual Studio bylo poprvé vydáno v roce 1997 společností Microsoft. Pokusili se tak složit více svých nástrojů dohromady a vytvořit jednotné prostředí pro vývoj. Visual Studio 97 bylo vydáno ve dvou edicích, Professional a Enterprise. Visual Studio 97 byl první pokus Microsoftu použít jedno vývojové prostředí pro více jazyků. V průběhů let vzniklo mnoho verzí Visual studia a to 6.0, 2002, 2003, 2005, 2008, 2010, 2012, 2013, 2015, 2017 Visual Studio 2019 bylo Microsoftem uvolněno 2. dubna 2019. Poslední vydání stabilní verze 16.7.3 je z 8. září 2020.

8.1.3 Architektura

Visual Studio nepodporuje žádný programovací jazyk nebo nástroj sám o sobě. Místo toho je možno přidat různá rozšíření funkčnosti. Každá funkčnost je zabalena do balíčku VSPackage. Když je nainstalována, je dostupná jako služba. IDE poskytuje tři služby. Služba *SVsSolution* umožňuje očíslovat projekty a sestavy. Druhá služba *SVsUIShell* poskytuje rozdělování na okna a UI funkce (jako panely, nástrojové lišty a okna nástrojů). Poslední služba *SVsShell* se stará o registraci balíčků VSPackage.

8.2 Xamarin.UItest cross-platform test

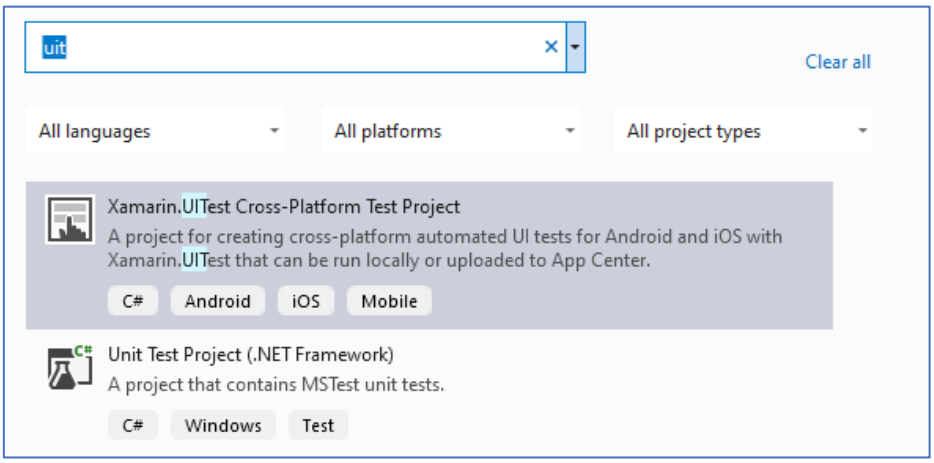
Xamarin.UITest je testovací Framework v jazyce C# využívající NUnit pro testy přijatelnosti uživatelského rozhraní v aplikacích pro iOS a Android. Xamarin.UITest je knihovna automatizace, která umožňuje provádění testů NUnit na zařízeních Android a iOS. Testy interagují s uživatelským rozhraním jako uživatel: zadávání textu, klepání na tlačítka a gesta – například tahy prstem[52]. Xamarin.UITest poskytuje

[51] MICROSOFT. IntelliSense v aplikaci Visual Studio. [online]. 2021. [Cit. 22. 01. 2021].
[52] MICROSOFT. Xamarin.UITest. [online]. 2021. [Cit. 02. 03. 2021].

read-eval-print-loop (REPL), který umožňuje vývojářům a testerům komunikovat s obrazovkou za běhu aplikace a zjednodušuje vytváření dotazů.

8.2.1 Vytvoření projektu Xamarin.UITest

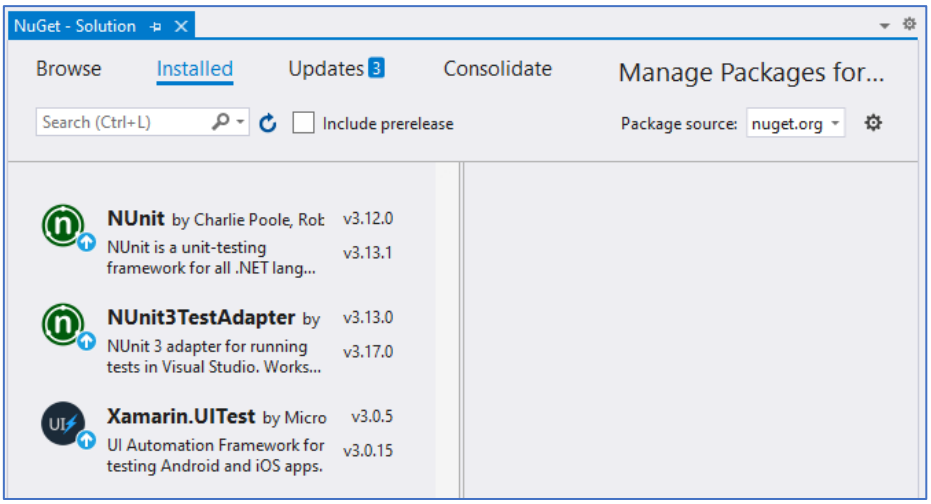
Při spuštění programu Visual Studio 2019 se zobrazí nabídka možností, vybrala jsem vytvořit nový projekt. V nabídce jsem vyhledala vhodný projekt na testování. Pro psaní automatického testu GUI ve Visual studiu jsem vybrala typ projektu Xamarin.UITest Cross-Platform Test Projekt.



Obrázek 1: Založení projektu[53]

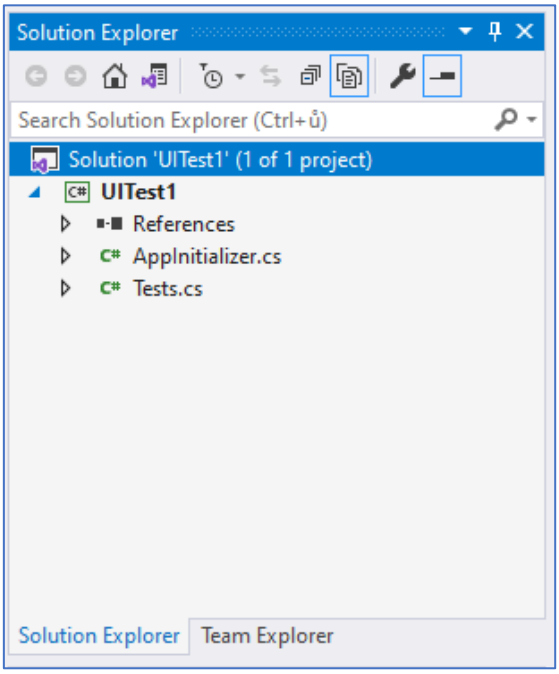
Správný chod testu zajišťují balíčky, které je nutné přidat k projektu, konkrétně balíčky NuGet s názvem NUnit, Xamarin.UITest a NUnitTestAdapter, bez zmíněných doplňků nepůjdou testy spustit.

[53] Zdroj vlastní.



Obrázek 2: NuGet nastavení[54]

Nový projekt má dvě třídy. AppInitializer obsahuje kód, který pomáhá inicializovat a nastavit testy. Druhá třída, Testy, obsahuje standardní kód, tedy automatické testy samotné.



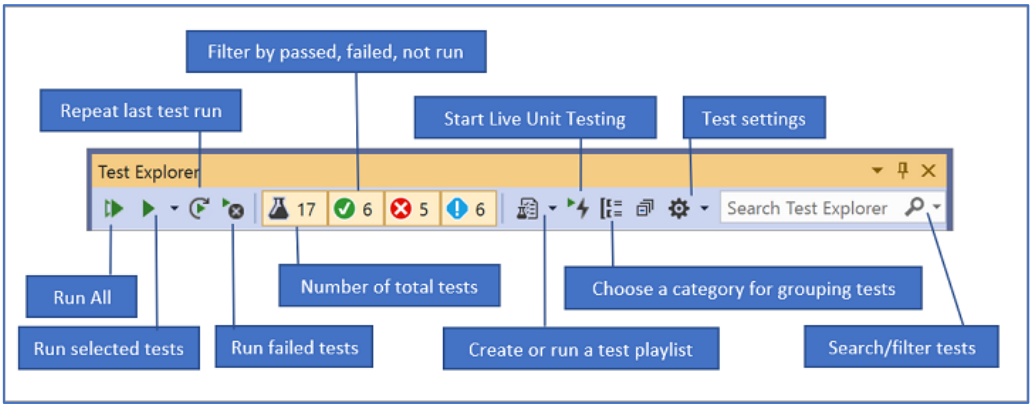
Obrázek 3: Solution Explorer[55]

V projektu je vhodné zakládat složka a až poté ve složkách vytvořit testovací třídy. Testy jsou díky tomu roztříděné a přehledné.

[54] Zdroj vlastní.
[55] Zdroj vlastní.

8.2.2 Správa testů

Pomocí Test Explorer (Průzkumníka testů) lze spustit testy jak jednotlivě, tak naráz. Na panelu můžeme vidět kolik testů celkem kód obsahuje, kolik proběhlo úspěšně, špatně nebo vůbec neproběhlo. Dále je možné testy seskupit podle vybraných kritérií. Výchozím seskupením testů je název projektu a třídy.



Obrázek 4: Průzkumník testů[56]

V okně pod Test Explorerem vidíme detailní shrnutí průběhu testu jako je název testu, číslo řádku, kde se test nachází, stav testu a uplynulý čas, po který test probíhal.

Pokud test skončí neúspěšně, pod oknem se zobrazí podrobnosti chybného testu s popisem chyby, která se vyskytla. Popis obsahuje název a čas trvání, plus zprávu o tom, jaká chyba nastala a trasování kroků před objevení chyby.

8.2.3 Spuštění testu

Testy spouštíme pomocí Test Exploreru, můžeme je spustit všechny najednou, celé sady nebo jednotlivě. Všechny testy spustíme pomocí funkce Run All (spustit vše), sadu testů ikonou spuštění a vybráním kategorie. Jednotlivé testy nejprve vybereme stisknutí klávesové zkratky CTRL + klikem na testy v seznamu, které chci spustit. Po označení vybraných testů zvolíme možnost Spustit vybrané testy.

8.2.4 Možnosti testování a pořizovací cena nástroje

Visual Studio nabízí pomocí doinstalování doplňků všechny druhy testovacích projektů, a to přes jednotkové, grafické testy až po výkonnostní a bezpečnostní testy.

[56] MICROSOFT. Spouštění testů částí pomocí Průzkumníka testů. [online]. 2021. [Cit. 15.3.2021].

Pořizovací cena nástroje se samozřejmě odvíjí od účelu, ke kterému bude sloužit a od počtu licencí. V ceně se samozřejmě promítá případná podpora softwaru A může se pohybovat od 2 000 Kč až po statisíce pro velké firmy.

8.3 Ranorex a její historie

Ranorex je Framework poskytující GUI automatické testy od společnosti Ranorex GmbH. Ranorex slouží také k testování desktopových, webových a mobilních aplikací. Tento Program nabízí testovací záznam a následné vytvoření testu. Pro psaní testů v kódu lze použít Ranorex studio[57], které tyto testy nabízí.

Společnost Ranorex byla založena roku 2007 v Rakousku, hlavní sídlo měla ve Štýrském Hradci, které bylo roku 2020 uzavřeno. Dnes má společnost hlavní sídlo v Houstonu v Texase.

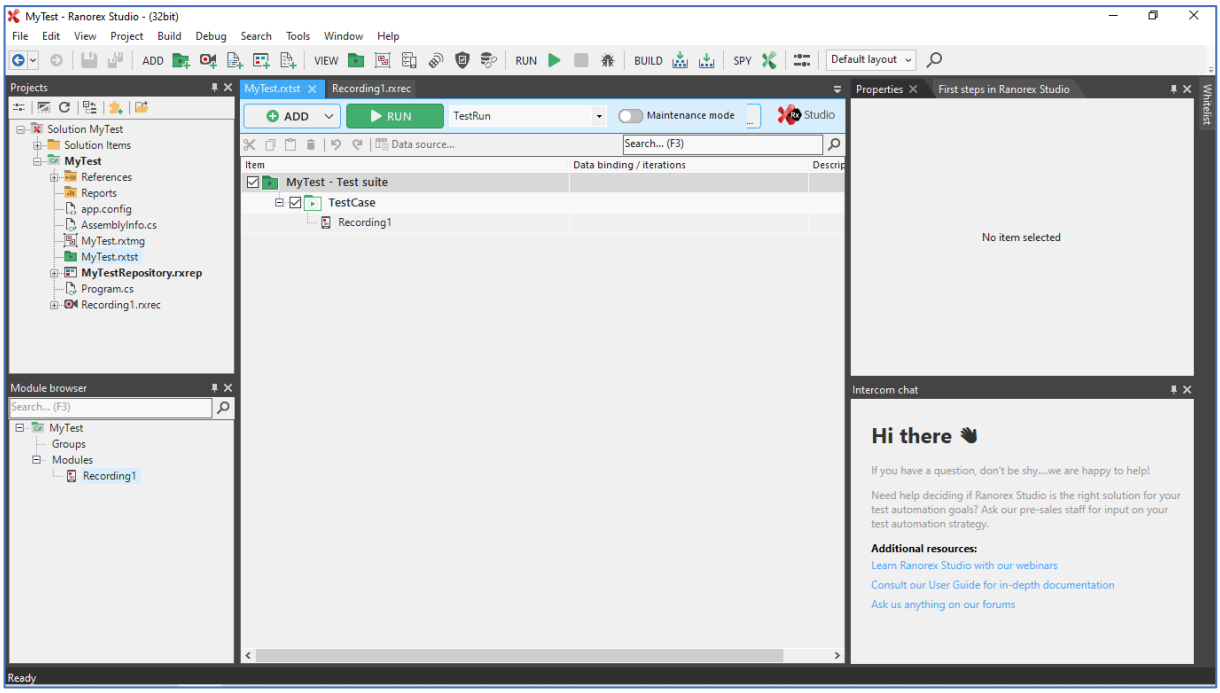
8.3.1 Ranorex studio

Bylo založeno roku 2007 společností Ranorex a poskytuje nástroje pro automatizaci testů uživatelského rozhraní pro desktopové, webové a mobilní zařízení. Ranorex studio nabízí rozpoznávání objektů GUI, filtrování prvků GUI pomocí vlastní technologie RanoreXPath a Ranorex Recorder. Ranorex Recorder zaznamenává interakce uživatele s aplikací a následně vytváří testovací skript. Ranorex podporuje jazyk C# stejně jako Visual Studio. RanoreXPath[58] rozeznávání objektů se opírá o chytrou technologii, která snižuje nároky na údržbu testů a zjednodušuje testování dynamicky vytvářených uživatelských rozhraní.

8.3.2 Prostředí Ranorex studia

Prostředí Ranorex Studio je vzhledově velmi podobné Visual S tudiu. Složení projektu můžeme vidět v levém panelu, podobá se Solution Exploreru. V podokně níže vidíme Module Browser, který zobrazuje vše, co modul obsahuje. V prostředním panelu vidíme správu testu nebo testovací třídu samotnou.

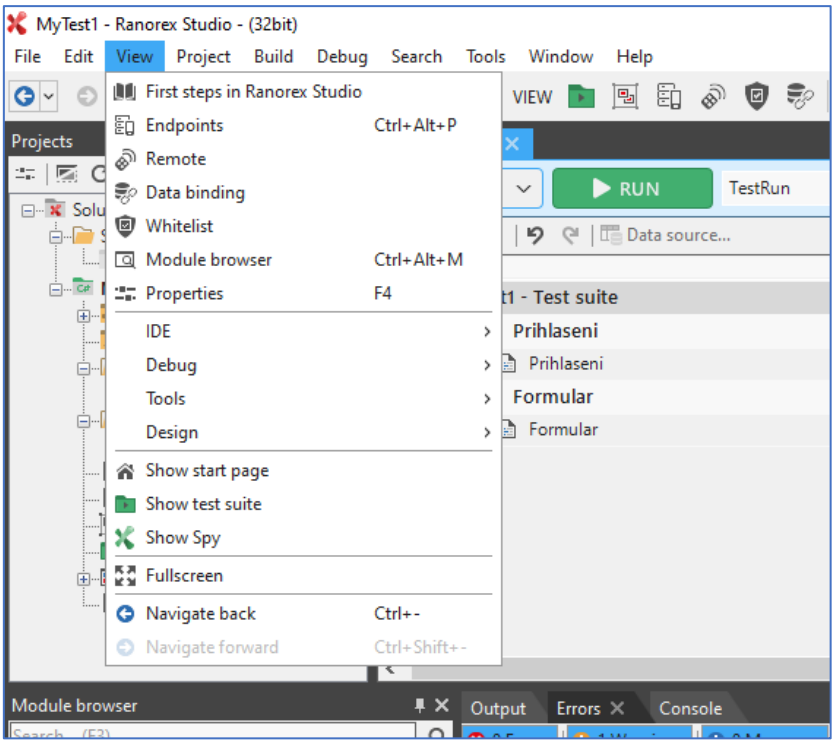
[57] RANOREX GMBH. Easy-to-use tools for beginners and experts. [online]. 2021. [cit. 30.3.2021].
[58] RANOREX GMBH. RanoreXPath [online]. 2021. [cit. 30.3.2021].



Obrázek 5: Prostředí Ranorex[59]

Panely se dají samozřejmě stejně jako ve Visual studiu uživatelsky přizpůsobit, a to pomocí záložky View, kde si určíme, co chceme na pracovní ploše zobrazit. Zobrazená okna poté můžeme přemisťovat podle naší potřeby.

[59] Zdroj vlastní



Obrázek 6: Možnosti nastavení pracovní plochy[60]

8.3.3 Vytváření testu a jejich spuštění

Testy se dají ve Ranorexu tvořit i pomocí záznamu obrazovky, kdy se spustí aplikace a Ranorex Recorder a manuálně se „naklikají“ testy. Z těchto záznamů se vytvoří skripty, které se dají následně spustit. Já jsem si ovšem vybrala tvořit test pomocí kódu, podobně jako ve Visual studiu. Postup je také podobný, do projektu lze založit nové složky a třídy.

Testy lze spouštět pomocí akce Run Code na konkrétní testovací třídě nebo na přehledu všech testů akcí Run. Testy jdou postupně za sebou, podle pořadí, jaké jim bylo přiděleno. Po proběhnutí testů se otevře nové okno s výsledky testů.

8.3.4 Možnosti testování a pořizovací cena nástroje

Ranorex nabízí podobně jako Visual Studio širokou nabídku testů přes grafické UI testy po výkonnostní, mimo mobilní aplikace podporuje i testování webových a desktopových aplikací.

Pro vyzkoušení je trial verze na 30 dní je zcela zdarma. Ceny se dále pohybuje od 690 $ po 4,790 $. Ceny se odvíjejí od nabízených možností a podpory.

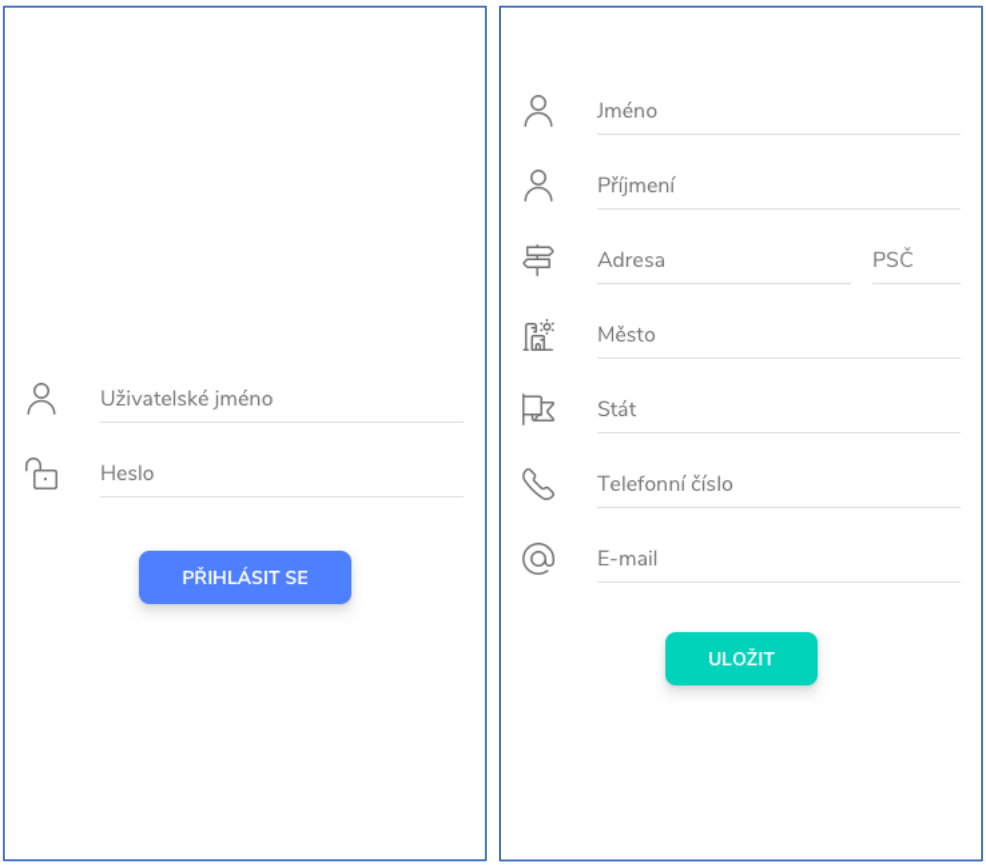
[60] Zdroj vlastní

9 Využití automatizovaných testů v praxi

Pro podrobnější prozkoumání práce s nástroji použiji vytvořenou jednoduchou aplikaci, která obsahuje základní validace pole a tlačítka. K základním GUI testů vytvořím use casy, kde ověřím vstupní data pomocí pozitivní i negativních testů.

9.1 Konkrétní Use case

Pro práci jsem si připravila případ, kdy se uživatel přihlásí do aplikace pomocí svého zaregistrovaného účtu. Po ověření a přihlášení systém zobrazí formulář s osobními údaji, který uživatel může uložit až po jeho vyplnění.



Obrázek 7: Přihlašovací a úvodní obrazovka[61]

Podrobnější práce se systémem je taková, že uživatel vyplní svůj email a heslo na úvodní obrazovce po spuštění aplikace, po kliknutí na tlačítko přihlásit, systém ověří zda byl email zaregistrován. Pokud zaregistrován nebyl, zobrazí se dialogové okno s oznámením o neplatném emailu. Pokud není správně zadané heslo, zobrazí se dialog

[61] Zdroj vlastní.

o špatně zadaném hesle. Po úspěšném přihlášení se zobrazí základní formulář s osobními údaji, které uživatel vyplní a uloží, i zde jsou vložené validace na jednotlivá pole a pokud nejsou pole vyplněná, nelze formulář uložit. Po uložení se uživatel opět ocitá na úvodní obrazovce.

9.2 Test case

V této kapitole si podrobně popíšeme jednotlivé postupy uživatele v aplikaci, které budou simulovat reálné situace. Struktura jednotlivých test casů bude název, typ a cíl testu, vstupní data, postup, očekávaný výstup. První sada test casů simuluje situaci uživatele na přihlašovací obrazovce a jeho možnosti na obrazovce.

Název testu: Test1 – přihlášení

Typ testu: Pozitivní test

Cíl testu: Ověření přihlášení

Vstupní data: test@gmail.com, test

Postup:

- uživatel vyplní uživatelské jméno a heslo podle zadaných vstupů,
- uživatel klikne na tlačítko „Přihlásit se“,
- systém ověří zadané vstupy.

Očekávaný výsledek: Přihlášení do aplikace.

Název testu: Test2 – přihlášení

Typ testu: Negativní test

Cíl testu: Ověření přihlášení

Vstupní data: testtest@gmail.com, test

Postup:

- uživatel vyplní uživatelské jméno a heslo podle zadaných vstupů,
- uživatel klikne na tlačítko „Přihlásit se“,
- systém ověří zadané vstupy.

Očekávaný výsledek: Nelze přihlásit uživatele.

Název testu: Test3 – přihlášení

Typ testu: Negativní test

Cíl testu: Ověření přihlášení

Vstupní data: test@gmail.com, testy

Postup:

- uživatel vyplní uživatelské jméno a heslo podle zadaných vstupů,
- uživatel klikne na tlačítko „Přihlásit se“,
- systém ověří zadané vstupy.

Očekávaný výsledek: Nelze přihlásit uživatele.

Název testu: Test4 – přihlášení

Typ testu: Negativní test

Cíl testu: Ověření přihlášení

Vstupní data: @gmail.com, test

Postup:

- uživatel vyplní uživatelské jméno a heslo podle zadaných vstupů,
- uživatel klikne na tlačítko „Přihlásit se“,
- systém ověří zadané vstupy.

Očekávaný výsledek: Nelze přihlásit uživatele.

Název testu: Test5 – přihlášení

Typ testu: Negativní test

Cíl testu: Ověření přihlášení

Vstupní data: -

Postup:

- uživatel nevyplní uživatelské jméno a heslo podle zadaných vstupů,
- uživatel klikne na tlačítko „Přihlásit se“,
- systém ověří zadané vstupy.

Očekávaný výsledek: Nelze přihlásit uživatele.

Další sada testů ověřuje správné vyplnění formuláře, kdy textová pole očekávají text, číselná pole číslo a pole s emailem správný formát emailu.

Název testu: Test6 – Formulář

Typ testu: Pozitivní test

Cíl testu: Ověřit uložení formuláře

Vstupní data: test@gmail.com, test, Marta, Kummerová, Družební 6, 77900, Olomouc, Česko, 602363318, test@gmail.com

Postup:

- uživatel vyplní přihlašovací údaje a přihlásí se pomocí tlačítka,

- uživatel vyplní formulář podle zadaných vstupů,
- uživatel formulář uloží.

Očekávaný výsledek: Uložený formulář a návrat na přihlašovací obrazovku.

Název testu: Test7 – formulář

Typ testu: Negativní test

Cíl testu: Ověřit povinnost polí

Vstupní data: test@gmail.com, test, Družební 6, 77900, Olomouc, Česko, 602363318, test@gmail.com

Postup:

- uživatel vyplní přihlašovací údaje a přihlásí se pomocí tlačítka
- uživatel vyplní formulář podle zadaných vstupů,
- uživatel klikne na tlačítko uložit.

Očekávaný výsledek: Formulář nelze uložit.

Název testu: Test8 – formulář

Typ testu: Negativní test

Cíl testu: Ověřit povinnost polí

Vstupní data: test@gmail.com, test, Marta, Kummerová, Olomouc, Česko, 602363318, test@gmail.com

Postup:

- uživatel vyplní přihlašovací údaje a přihlásí se pomocí tlačítka
- uživatel vyplní formulář podle zadaných vstupů,
- uživatel klikne na tlačítko uložit.

Očekávaný výsledek: Formulář nelze uložit.

Název testu: Test9 – formulář

Typ testu: Negativní test

Cíl testu: Ověřit povinnost polí

Vstupní data: test@gmail.com, test, Marta, Kummerová, Družební 6, 77900, 602363318, test@gmail.com

Postup:

- uživatel vyplní přihlašovací údaje a přihlásí se pomocí tlačítka
- uživatel vyplní formulář podle zadaných vstupů,
- uživatel klikne na tlačítko uložit.

Očekávaný výsledek: Formulář nelze uložit

Název testu: Test10 – formulář

Typ testu: Negativní test

Cíl testu: Ověřit povinnost polí

Vstupní data: test@gmail.com, test, Marta, Kummerová, Družební 6, 77900, Olomouc, Česko

Postup:

- uživatel vyplní přihlašovací údaje a přihlásí se pomocí tlačítka
- uživatel vyplní formulář podle zadaných vstupů,
- uživatel klikne na tlačítko uložit.

Očekávaný výsledek: Formulář nelze uložit.

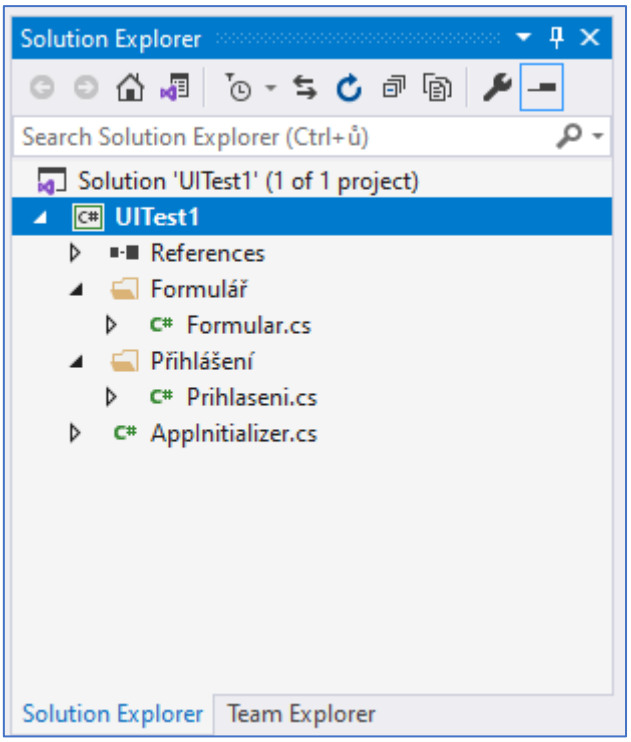
9.3 Nasazení testu ve Visual studiu 2019 s Xamarin.UITest

Stažení programu je možné na oficiálních stránkách Visual studia, ovšem samotná instalace zabere nějaký čas, před instalací je potřeba vybrat doplňky, které bude pro programování potřebovat, popřípadě na druhé záložce jiný jazyk nástroje.

Pro praktickou ukázku automatického testu jsem založila projekt Xamarin.UITest Cross-platform test ve Visual studiu 2019. V rámci tohoto projektu jsem vytvořila testy podle výše uvedených test casů.

9.3.1 Vytvoření projektu

Konkrétní projekt jsem vytvořila podle návodu z kapitoly 8.2. Název projektu jsem nechala defaultní a doinstalovala také doplňky NuGet, pro správný chod testů. V rámci projektu jsem založila složky, pro konkrétní testovací třídy.



Obrázek 8: Solution Explorer[62]

9.3.2 Vytvoření testů

Po otevření nové třídy Prihlaseni, kam budu psát testy, se mi zobrazí prázdná třída, kterou musím správně doplnit, aby šly testy spustit. Nejprve v namespacu třídy definuji, jaké metody budu používat, pomocí instance [TestFixture(Platform.Android)]. Ta určuje, zda budu automatizovat testy pro systém Android. Dále si třídu nastavím na veřejnou public třídu. Jak jsem dříve popsala, třída se skládá z části SetUp a Test, v SetUp části se aplikace spustí a mohou se zde určit akce, které proběhnout před každým jednotlivým testem. Ukázka test casu *Test1 – Přihlášení:*

```
[SetUp]
    public void BeforeEachTest()
    {
       app = AppInitializer.StartApp(platform);
    }
[Test]
    public void Test1_prihlaseni()
    {
```

[62] Zdroj vlastní

```
app.Repl();
    app.EnterText(x => x.Id("Username"), "test@gmail.com");
    app.DismissKeyboard();
    app.EnterText(x => x.Id("Password"), "test");
    app.DismissKeyboard();
    app.Tap(x => x.Marked("Login_Btn"));
    AppResult[] result = app.WaitForElement(x => x.Marked("Save_Btn"));
Assert.IsTrue(results.Any(), "Nelze uživatele přihlásit.");
 }
```

Na této ukázce kódu si můžeme všimnout základních příkazů, které Xamarn.UITest používá EnterText, Tap, Marked a WaitForElement. Konstrukce EnterText vloží na určené místo zadaný text. Tap znamená klikni na zadané místo. Marked znamená najdi element s názvem, který je zadaný. WaitForElement se používá, pokud od aplikace čekáme lehké zdržení, například načítání nové stránky. I v ostatních testech jsem použila stejnou konstrukci a příkazy. Jak vlastně zjistím název pole (elementu), který chci použít? Pomocí metody app.Repl(), která spustí terminálové okno, a v něm pomocí příkazu *tree* se hierarchicky zobrazí popsané prvky, které se vyskytují na obrazovce aplikace. Z tohoto příkazu tree tak mohu vyčíst, jak se prvky, které potřebuji použít nazývají.

Pro tvorbu testů je nutná alespoň základní znalost programovacího jazyka, při složitějších testech je možné uplatnit různé programovací cykly a podmínky. Například podmínka větvení If[63], která udává, že pokud něco platí nebo naopak neplatí, provedu danou akci nebo bloky Try/Catch, kdy se v průběhu testu snažím zachytit chybu[64]. V některých případech i cyklus While, kdy se blok příkazů opakuje, dokud není splněna podmínka[65].

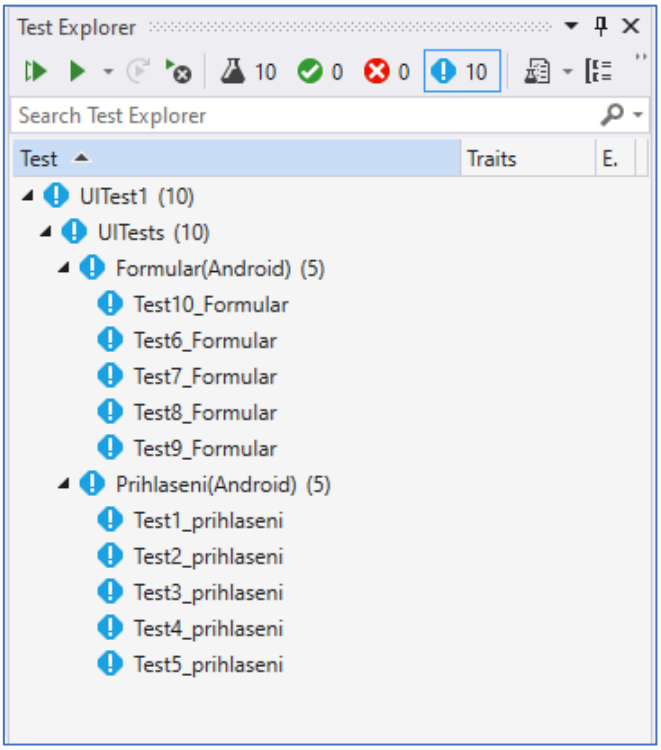
[63] HANÁK, Ján. C# 3.0: programování na platformě .NET 3.5. Brno: Zoner Press, 2009. s. 200.

[64]KOZLOVSKÝ, Pavel a NAGYOVÁ, Ingrid. Objektově orientované programování. Orlová: Obchodní akademie Orlová, 2007. s. 125.

[65] TROELSEN, Andrew W. C# a .NET 2.0 profesionálně. Brno: Zoner Press, 2006. s. 152.

9.3.3 Správa testů

Ve Visual studiu se o testy stará Test Exporer. Na obrázku níže vidíme testy, které jsem vytvořila, kategorizované podle tříd a seřazené abecedně. V panelu Test Exploreru také vidíme, že projekt obsahuje 10 testů, které zatím nebyly spuštěny.

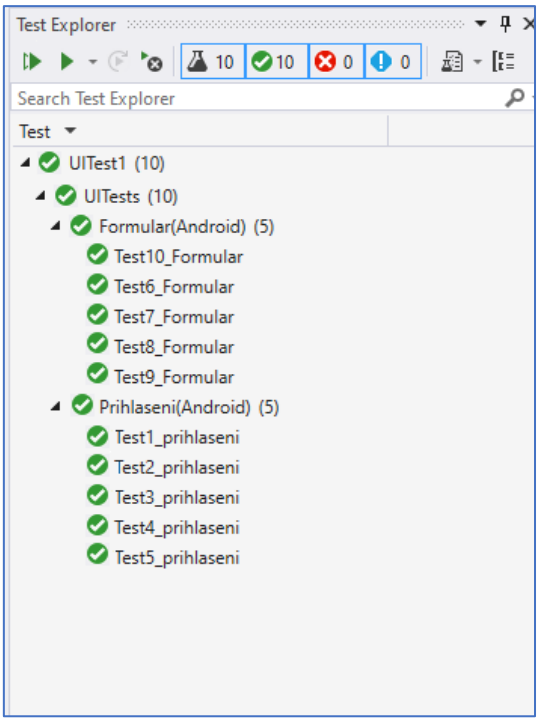


Obrázek 9: Test Explorer s ukázkovými testy[66]

9.3.4 Spuštění a výsledek testů

K počítači připojím telefon nebo jiné zařízení se systémem Android, na kterém mám nahranou odpovídající aplikaci a v Test Exploreru spustím akci RunAll. Po spuštění se mi na telefonu aplikace spustí a postupně se provedou všechny zadané testy. Na mobilním zařízení vidím, že se dané kroky kódu provádějí. Výsledek můžeme vidět na obrázku níže, testy proběhli v pořádku. Do třídy AppInitializer lze napsat kód, který výsledky testů uloží do souboru.

[66] Zdroj vlastní.



Obrázek 10: Test Explorer s testy[67]

9.3.5 Výhody a nevýhody nástroje

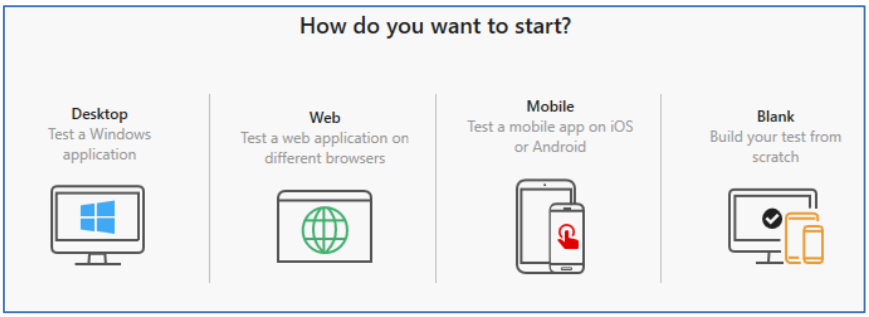
Visual Studio a Xamarin.UITest mají širokou škálu možností pro testování a jelikož je Visual Studio vývojové prostředí, přináší další možnosti ve formě doplňků a rozšíření. To je veliká výhoda oproti jiným pouze testovacím prostředím. Vývojový tým nemusí měnit program, na který jsou zvyklý, ve kterém vyvíjí a testují. Jako nevýhodu bych uvedla, že je potřebná základní znalost programovacího jazyka. V mém případně jazyka C#. Celkový dojem z nástroje je pozitivní, i přes složitost a robustnost nástroje, se v něm pracuje dobře a otázky mi pomohla velmi dobře vypracovaná dokumentace.

9.4 Nasazení testu ve Ranorex studiu

Samotné získání programu je dost složité, pokud nechcete za nástroj platit. Oficiální stránky nabízejí trial verzi zdarma, musela jsem však vyplnit podrobnější formulář, následně mi byl zaslán link se stažením souboru. Tria verze zdarma je platná pouze na 30 dní. Po stažení následovala instalace, které obsahovala přihlášení k registrovanému účtu. Poté se instalace úspěšně dokončila.

[67] Zdroj vlastní

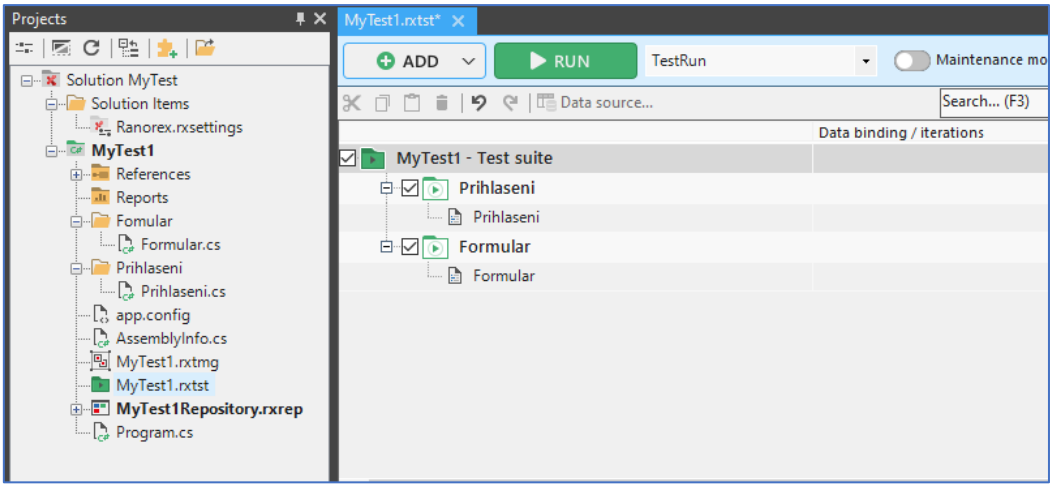
Po spuštění programu se zobrazí průvodce nastavení projektu, kde jsem v prvním kroku určila mobilní aplikace, další možnosti byli desktopová aplikace, webová aplikace a vytvoř si vlastní test. V druhém kroku jsem z možností platforem vybrala mobilní aplikace pro Android. Poslední krok obsahoval nezbytné vlastnosti, které projekt musí obsahovat, aby fungoval správně. Následně se projekt vytvořil a začala jsem ho plnit testy.



Obrázek 11: Výběr testování[68]

9.4.1 Vytvoření testů

I přes možnost vytvořit test pomocí záznamu, jsem zvolila cestu kódu. Zvolila jsem tak z toho důvodu, abych udržela podobnost s testy ve Visual studiu. Jelikož Ranorex Studio podporuje jazyk C#, mohla jsem se pustit do psaní testů. Podobně jsem si i vytvořila složky a třídy, do kterých testovací kód vložila.



Obrázek 12: Vytvoření tříd

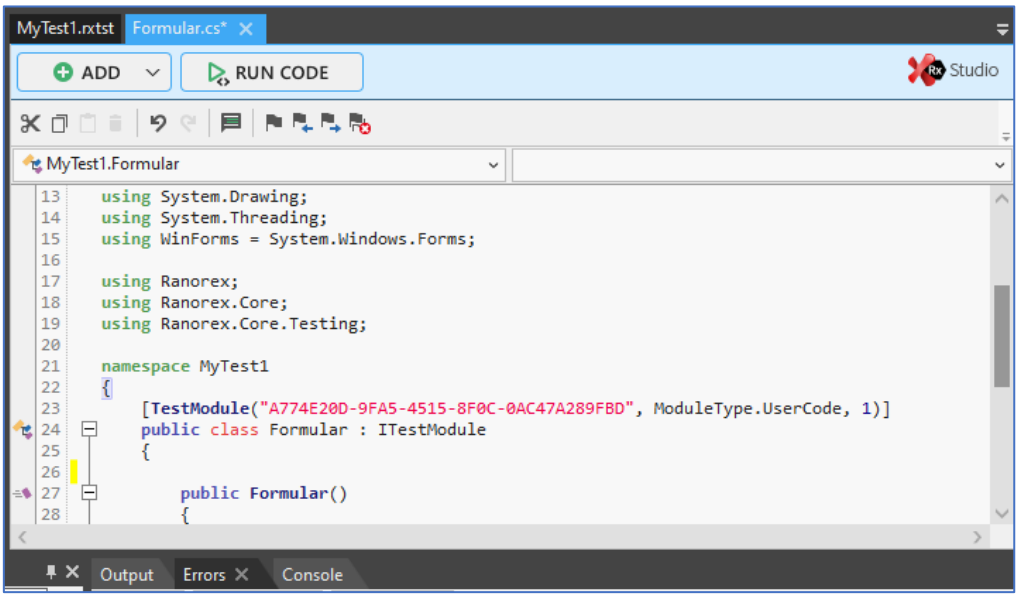
[68] Zdroj vlastní

9.4.2 Správa testů

Testy lze spravovat pomocí souboru typu Repository. Do toho souboru jsem vložila vytvořené složky a třídy. Soubor Repository má možnost funkce Run, která spustí příslušné testy.

9.4.3 Spuštění a výsledek testů

Jak už jsem zmínila v předchozí podkapitole, testy je možné hromadně spustit za souboru typu Repository. Při tvorbě konkrétního testu, si uživatel může spustit jen konkrétní třídu, a to funkcí Run Code, která se nachází v hlavičce otevřené třídy.



Obrázek 13: Samostatné spuštění třídy[69]

Po proběhnutí testů se otevře nové okno se shrnutím výsledků testů. Podobně jako ve Visual Studiu, i zde vidíme název projektu, jednotlivých testů a dobu trvání. Při vyskytnutí chybu informace, kde chyba nastala.

9.4.4 Výhody a nevýhody nástroje

Velká výhoda tohoto nástroje spočívá v možnosti tvořit testy pomocí záznamu, i když jsem tvořila test kódem, tuto možnost jsem také otestovala. Není tudíž potřebná znalost žádného programovacího jazyka. Nevýhoda může být v tom, že vývojové prostředí je odlišné od testovacího, takže je práce týmu rozšířena o software navíc, který je také

[69] Zdroj vlastní

zpoplatněn. Po zamyšlením se nad výhodami a nevýhodami musím říct, že program je velmi přívětivý a přehledný. Dobře zpracovaná dokumentace mi pomohla se seznámením s nástrojem.

Závěr

Vypracování bakalářské práce mi pomohlo rozšířit a také ucelit pohled na testování softwaru. I přes mou praxi v testování mobilních aplikací jsem objevila několik způsobů, jak zefektivnit svoji budoucí práci. Při tvorbě této práce jsem měla možnost nahlédnout do způsobu testování aplikací se systémem iOS od společnost Apple. Testování systému iOS se zdálo být velmi zajímavé jako možnost pro rozšíření bakalářské práce o diplomovou práci.

Teoretickou část bakalářské práce jsem věnovala dílčím cílům. Popsání problematiky testování softwaru, testovacích prostředí, vývoje a historie mobilních aplikací Android a obecného popisu automatického testování.

První dílčí cíl spočíval v definování problematiky testování. Nastínila jsem základní pojmy týkající se testování. Rozdělila a popsala konkrétní druhy testů, jakými se dá software testovat, a co obnášejí. Shrnula jsem i metody, jak k testování přistupovat.

V druhém dílčím cíli jsem objasnila různé typy testovacích prostředí, jaké druhy testů obsahují a k čemu se používají.

Třetí dílčí cíl spočíval v seznámení s vývojem mobilních aplikací Android. Zde jsem popsala historii vývoje systému Android, jeho architekturu a vrstvy, ze kterých se skládá. Dále popisuji základy samotného vývoje aplikace a následnou distribuci aplikace.

Poslední dílčí cíl spočíval v popsání automatického testu, rozebírám, jak poznat, co stojí za to automatizovat a co se nevyplatí. Srovnávám výhody a nevýhody, které automatické testy obnášejí.

Hlavní cíl měl za úkol ověřit přístupy pro automatické testování GUI mobilních aplikací na platformě Android. Pro jeho splnění jsem prozkoumala trh nástrojů, které nabízejí testování mobilních aplikací. Vytvořila jsem srovnávací tabulku, kde jsem srovnávala nabízené technologie, dostupnou dokumentaci, licence a dostupnost a poslední vydanou stabilní verzi. Dále jsem z důvodu velké uživatelské základny a podobnosti funkcí vybrala nástroj Visual Studio 2019 s Xamarin.UITes a Ranorex studio. Popsala jsem, jak prostředí programu vypadá, jak se spravují testy, možnosti, které nabízejí, a pořizovací cenu nástroje. V neposlední řadě jsem vytvořila konkrétní testovací případy a práce s nimi, jako výstup je shrnutí výhod a nevýhod nástrojů.

Seznam zdrojů

AMMANN, Paul a Jeff OFFUTT. *Introduction to software testing*. New York: Cambridge University Press, 2008. ISBN 978-0-521-88038-1.

BUCHALCEVOVÁ, Alena. *Metodiky vývoje a údržby informačních systémů: kategorizace, agilní metodiky, vzory pro návrh metodiky*. Praha: Grada, 2005. ISBN 80-247-1075-7. Dostupné také z: https://ndk.cz/uuid/uuid:688d71f5-78ab-44c5-b9c9-ea997f1340c0

BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu.* Praha: Grada, 2016. ISBN 978-80-247-5594-6.

ECLIPSE FOUNDATION. Jubula. [online]. [cit. 30.3.2021]. https://www.eclipse.org/jubula/

EGGPLANT. *Mobile Gateways*. [online]. 2020 [cit. 30.3.2021]. https://www.eggplantsoftware.com/mobile-gateways

GOUSSET, Mickey a POKORNÝ, Jan. *Řízení životního cyklu aplikací ve Visual Studiu 2010*. Brno: Zoner Press, 2010. ISBN 978-80-7413-102-8. Dostupné také z: https://ndk.cz/uuid/uuid:841b7920-b2e7-11e4-9a04-5ef3fc9bb22f

HANÁK, Ján. *C# 3.0: programování na platformě* .NET 3.5. Brno: Zoner Press, 2009. s. 200. ISBN 978-80-7413-046-5. Dostupné také z: https://ndk.cz/uuid/uuid:f6700ed0-1929-11e4-a8ab-001018b5eb5c

HEROUT, Pavel. *Testování pro programátory.* České Budějovice: Kopp, 2016. ISBN 978-80-7232-481-1.

KADLEC, Václav. *Agilní programování: metodiky efektivního vývoje softwaru*. Brno: Computer Press, 2004. ISBN 80-251-0342-0. Dostupné také z: https://ndk.cz/uuid/uuid:441b1e60-6775-11e4-8214-005056827e51

KOZLOVSKÝ, Pavel a NAGYOVÁ, Ingrid. *Objektově orientované programování.* Orlová: Obchodní akademie Orlová, 2007. s. 125. ISBN 978-80-87113-22-6. Dostupné také z: https://ndk.cz/uuid/uuid:b1405e60-ca8d-4cab-96dc-ea8bc84689c2

LACKO, Ľuboslav. *Vývoj aplikací pro Android*. Brno: Computer Press, 2015. ISBN 978-80-251-4347-6.

MICRO FOCUS. *Silk Test 21.0: Testing Mobile Applications*. [online]. 2020 [cit. 30.3.2021]. https://www.microfocus.com/documentation/silk-test/210/en/silktest-testingmobile-en.pdf

MICROSOFT. *Distribuční smlouva pro vývojáře Google Play.* [online]. 2021. [Cit. 22. 01. 2021]. https://play.google.com/intl/ALL_cz/about/developer-distribution-agreement.html

MICROSOFT. *IntelliSense v aplikaci Visual Studio.* [online]. 2021. [Cit. 22. 01. 2021]. https://docs.microsoft.com/cs-cz/visualstudio/ide/using-intellisense?view=vs-2019#see-also

MICROSOFT. *Xamarin.UITest.* [online]. 2021. [Cit. 02. 03. 2021]. https://docs.microsoft.com/en-us/appcenter/test-cloud/frameworks/uitest/

PAGE, Alan, JOHNSTON, Ken a ROLLISON, Bj. *Jak testuje software Microsoft*. Brno: Computer Press, 2009. ISBN 978-80-251-2869-5. Dostupné také z: https://ndk.cz/uuid/uuid:f0046150-e94d-11e4-a511-5ef3fc9ae867

PATTON, Ron. *Testování softwaru*. Praha: Computer Press, 2002. ISBN 80-7226-636-5. Dostupné také z: https://ndk.cz/uuid/uuid:a5b6bf60-0476-11e9-95ba-5ef3fc9bb22f

PROGRESS SOFTWARE CORPORATION. *GUI Testing in Visual Studio.* [online]. 2021. [cit. 30.3.2021]. https://www.telerik.com/teststudio/visual-studio-testing-plugin-benefits

PROGRESS SOFTWARE CORPORATION. [online]. 2021. [cit. 30.3.2021]. https://docs.telerik.com/teststudio/getting-started/first-project

HOCKE, Raimund. *What is SikuliX?* [online]. 2017. [cit. 30.3.2021] http://sikulix.com/

RANOREX GMBH. *Easy-to-use tools for beginners and expert*s. [online]. 2021. [cit. 30.3.2021]. https://www.ranorex.com/automated-gui-testing-tools

RANOREX GMBH. *RanoreXPath*. [online]. 2021. [cit. 30.3.2021]. https://www.ranorex.com/help/latest/ranorex-studio-advanced/ranorexpath/introduction/

SINGH, Yogesh. *Software testing.* New York: Cambridge University Press, 2012. ISBN 978-1-107-01296-7.

SMARTBEAR SOFTWARE. *Automated Testing.* [online]. 2020. [cit. 30.3.2021]. https://support.smartbear.com/testcomplete/docs/tutorials/getting-started/intro/automated-testing.html

ŠOCHOVÁ, Zuzana a KUNCE, Eduard. *Agilní metody řízení projektů*. Brno: Computer Press, 2014. ISBN 978-80-251-4194-6. Dostupné také z: https://ndk.cz/uuid/uuid:31f1bec9-aeda-4308-b5b6-9073ac1e7afa

T-PLAN LIMITED. *Overview*. [online]. 2021. [cit. 30.3.2021]. https://www.t-plan.com/

THE JS FOUNDATION. *Představujeme Appium*. [online]. [cit. 30.3.2021]. https://appium.io/

TROELSEN, Andrew W. *C# a .NET 2.0 profesionálně.* Brno: Zoner Press, 2006. s. 152. ISBN 80-86815-42-0. Dostupné také z: https://ndk.cz/uuid/uuid:dc914c20-c711-11e6-b22f-5ef3fc9ae867

UJBÁNYAI, Miroslav. *Programujeme pro Android.* Praha: Grada, 2012. ISBN 978-80-247-3995-3.

VASANTH K. *How to do mobile app automation testing using MonkeyTalk tool?* [online]. 2016. [cit. 30.3.2021]. https://www.perfomatix.com/mobile-app-automation-testing/

WATIR. Watir is…. [online]. 2021. [cit. 30.3.2021]. http://watir.com/

ZEENYX. AscentialTest® Features. [online]. [cit 30.3.2021]. https://zeenyx.com/features/

Seznam obrázků a tabulek

Anotace

Bibliografický údaj: Kummerová, Marta. Přístupy k automatizovanému testování GUI na platformě Android. 2021.

Bakalářská práce. Moravská vysoká škola Olomouc. Vedoucí práce: PhDr. Jan Lavrinčík, Ph.D.

Název práce: Přístupy k automatizovanému testování GUI na platformě Android

Autor: Marta Kummerová

Ústav: Ústav informatiky a aplikované matematiky

Vedoucí práce: PhDr. Jan Lavrinčík, Ph.D.

Počet stran: 59

Abstrakt: Bakalářská práce se zaměřuje na nástroje pro automatické testování grafického rozhraní pro mobilní aplikace na platformě Android. Práce popisuje testování softwaru v průběhu jeho vývoje, který je nedílnou součástí vývojového procesu. Teoretická část se pojednává o vývoji softwaru a mobilních aplikací, obecně o testování softwaru a její problematice a jsou zde popsány základy automatického testování. Praktická část je zaměřena detailněji na automatické testování, uvádí konkrétní příklady testů a jejich aplikaci a v neposlední řadě prozkoumává dva různé nástroje na grafické testování mobilních aplikací. V závěru je zhodnocení práce s vybranými nástroji.

Klíčová slova: Vývojové modely softwaru, mobilní aplikace, testování, testovací případ, automatické testy, testovací prostředí, Visual Studio.

Title: Approaches to automated GUI testing on the Android platform

Author: Marta Kummerová

Department: Department of Computer Science and Applied Mathematics

Supervisor: PhDr. Jan Lavrinčík, Ph.D.

Number of pages: 59

Abstract: Bachelor thesis focuses on tools for automatic testing of the graphical interface for mobile applications on the Android platform. The thesis describes software testing during its development, which is an integral part of the development process. The theoretical part deals with the development of software and mobile applications, software testing in general and its issues, and describes the basics of automatic testing. The practical part is focused in more detail on automatic testing, gives specific examples of tests and their application and, last but not least, explores two different tools for graphical testing of mobile applications. In the end there is an evaluation of work with selected tools.

Keywords: Software development models, mobile applications, testing, test case, automated tests, test environment, Visual Studio.