Mendelova univerzita v Brně
Provozně ekonomická fakulta

Simulátor splátkového kalendáře s funkcí pro automatizované testování vstupních parametrů

Diplomová práce

Vedoucí práce:
Ing. Petr Jedlička, Ph.D.

Bc. Filip Walder

Brno 2015

Touto cestou bych rád poděkoval zejména vedoucímu své diplomové práce Ing. Petru Jedličkovi, Ph.D., za odborné vedení celé práce a věcné připomínky. Dále bych chtěl poděkovat Tomáši Holíkovi, Ljudmile Škyl a Janu Samkovi za poskytnuté rady ohledně anuitních typů úvěrů a analýze uplatnění automatizovaného testování.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Simulátor splátkového kalendáře s funkcí pro automatizované testování vstupních parametrů**
vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 21. května 2015 _____

Abstract

Walder, F. The installment calendar simulator with features for automated testing of input parameters. Diploma thesis. Brno: Mendel University in Brno, 2015.
The work deals with the design and implementation of the application that simulates an installment calendar for annuity loan payments redeeming automated testing of its input parameters.

Keywords

Automated testing, Java , payment schedule, annuity.

Abstrakt

Walder, F. Simulátor splátkového kalendáře s funkcí pro automatizované testování vstupních parametrů. Diplomová práce. Brno: Mendelova univerzita v Brně, 2015.
Práce se zabývá návrhem a implementací aplikace pro simulování splátkového kalendáře anuitních splátek úvěru s uplatněním automatizovaného testování jeho vstupních parametrů.

Klíčová slova

Automatizované testování, Java, splátkový kalendář, anuita.

Obsah

1 Úvod a cíl práce

1.1 Úvod

V dnešní době se již valná většina lidí setkala s možností poskytnutí půjčky. Mezi nejběžnější bankovní půjčky patří především revolvingové, hypoteční, hotovostní nebo spotřebitelské úvěry. Důvod čerpání je prostý. Lidé začínají podnikat, zakládat rodiny, potřebují revitalizovat své bydlení či nakoupit konkrétní spotřebič. Vzniká tak vztah mezi věřitelem a dlužníkem. Dlužník před výběrem konkrétní půjčky potřebuje znát výhodnost daného úvěru, popřípadě sumu měsíčních splátek včetně celkové doby splácení. Tyto informace může získat prostřednictvím splátkového kalendáře. Ten zobrazuje měsíční splátky, které dlužník musí splácet po celé úvěrové období. Potenciální zákazník tak zjistí konečnou sumu, kterou zaplatí, ještě předtím, než se rozhodne o danou půjčku zažádat.

O to důležitější je pro banku mít správně zobrazen a plně funkční splátkový kalendář pro různé typy úvěrů. V těchto splátkových kalendářích se nesmí vyskytovat ani sebemenší výpočetní chyba. Musí tak být provedena důkladná analýza všech možností, které mohou nastat. Banka má vlastní testery, jejichž úkolem je zhodnocení konkrétního stavu aplikace, kterou vede k jejímu konečnému schválení a nasazení. Firmy poskytující tyto komplexní bankovní aplikace bankám mají vlastní profesionálně vyškolené testery, jejichž úkolem je právě kontrola zmíněných aplikací.

Testeři pro své regresní a integrační testování vytvářejí nepřeberné množství smluv, které jsou vázané na fiktivní klienty. Tyto smlouvy obsahují mimo jiné parametry, na jejichž základě dochází k vytvoření splátkového kalendáře, který je nedílnou součástí smluv. Právě při špatném nastavení těchto parametrů dochází k vytvoření neadekvátního splátkového kalendáře. Při práci s danými smlouvami v různých modulech může tester reportovat chyby, které v koncovém důsledku nesouvisí s aktuálním procesem, ale právě se špatně nastavenými parametry ve smlouvě. Těmito chybami se musí zabývat vývojář, který hledá příčiny, a firmě se tak zvyšují celkové náklady na vývoj a testování.

Jelikož se systém vlivem nových požadavků může neustále měnit a narůstat, firmy tak nakládají větší finanční náklady na jejich otestování. Standardně je zapotřebí otestovat stálou (neměnnou) a nově přibývající funkčnost, aby zákazník, v tomto případě banka, byl spokojen a nemusel provádět reklamace. Firmy se snaží zavádět a využívat různé automatizované nástroje, díky kterým nedojde k navýšení finančních rezerv při testování. Je-li automatizace špatně zavedena, firmy mohou vynaložit větší náklady na její neustálou kontrolu a údržbu. Proto je důležité analyzovat, kde je výhodné tuto automatizaci zavádět a kde naopak tuto automatizaci neprovádět vůbec, nebo jen částečně.

1.2 Cíl

Cílem práce je navrhnout a implementovat simulátor splátkového kalendáře, včetně uplatnění automatizovaného testování jeho vstupních parametrů. Aplikace bude sloužit týmu testerů a analytiků firmy EmbedIT pro běloruský bankovní sektor. Prostřednictvím tohoto programu si budou moci testeři ověřovat správné nastavování vstupních parametrů pro anuitní splátky úvěrů. Aplikace bude obsahovat možnost uplatnění automatizované kontroly, která bude implementována na základě důkladné analýzy a návrhu. Tato automatizovaná kontrola bude navržena tak, aby firma nevynakládala svoje finanční rezervy na její neustálou kontrolu a údržbu. Součástí softwaru bude také programová dokumentace zajišťující korektní pochopení programu. Systém bude navržen tak, aby bylo možné ho upravit pro další týmy testerů z jiných destinací a byl co nejvíce uživatelsky přívětivý.

2 Úvěr

Úvěry jsou běžnými produkty pro domácnost i firmy, které nabízejí bankovní i nebankovní instituce. Firmy a domácnosti využívají těchto úvěrů nejen tehdy, když mají nedostatek vlastních finančních zdrojů, ale v některých případech i v situacích, pokud pro tyto zdroje mají vyšší zhodnocení, než jsou náklady spojené s úvěrem. Mezi základními typy úvěrů patří kontokorentní, spotřebitelský, provozní, investiční nebo dlouhodobé hypoteční úvěry (Šoba, Širůček, Ptáček, 2013, s. 121).

2.1 Anuita

Anuitou rozumíme platbu, která se pravidelně opakuje v určité frekvenci (ačkoliv původní význam znamenal platbu v určité výši placenou pouze jednou za rok). Anuitou tak může být pravidelná investice v konkrétní výši, například 10 000 Kč měsíčně do podílových fondů, pravidelná úložka na stavební spoření nebo na spořící účet, pravidelná splátka hypotečního úvěru, penzijní spoření atd. (Šoba, Širůček, Ptáček, 2013, s. 63).

Jelikož se jedná o pravidelně se opakující částku, jsou výpočty jednodušší, než kdyby se počítalo s každou dílčí částkou zvlášť. Ve vztazích se totiž využívá efektu pravidelného opakování anuit (Šoba, Širůček, Ptáček, 2013, s. 63).

Pro každý anuitní typ úvěru platí, že se anuita rozkládá na dvě složky, a to úmorovou část (úmor) a úrokovou část (úrok) (Šoba, Širůček, Ptáček, 2013, s. 121–123). Můžeme tedy použít následující vztah:

$$anuita = \text{úrok} + \text{úmor} \tag{1}$$

2.2 Úrok

Jedná se o veličinu, která hraje významnou roli při uzavírání obchodů bank. Tento důležitý faktor ovlivňuje výhodnost jak z pohledu věřitele, tak i dlužníka (Radová, Dvořák, Málek, 2013, s. 24).

Dojde-li k zapůjčení finančních prostředků mezi dvěma subjekty, bude subjekt, který tyto finanční prostředky zapůjčil, požadovat odměnu jako náhradu za dočasnou ztrátu kapitálu, za riziko spojené se změnami tohoto kapitálu a také za nejistotu, že kapitál nebude splacen v dané lhůtě a výši. Tuto odměnu považujeme za úrok. Věřitel tedy získává úrok za to, že poskytl své peníze dočasně někomu jinému. Naopak z pohledu dlužníka je úrok cena, kterou platí za získání úvěru (Radová, Dvořák, Málek, 2013, s. 28).

Častou chybou je zaměňování pojmů úrok a úroková sazba. Hlavním rozdílem je, že úrok se udává v peněžních hodnotách. Jedná se o cenu za zapůjčení peněz z pohledu dlužníka a odměna za zapůjčení peněz z pohledu věřitele. Úroková sazba je úrok vyjádřený v procentech z hodnoty kapitálu (Šoba, Širůček, Ptáček, 2013, s. 13–14).

Úrok lze kvantifikovat dle následujícího vztahu:

$$U = K \times r \times t \tag{2}$$

Kde U je výše úroku v peněžních hodnotách, K je výše kapitálu, r je úroková sazba za úrokové období a t je počet úrokových období úročení kapitálu K nebo také doba splatnosti (Šoba, Širůček, Ptáček, 2013, s. 20).

Úroky se hradí vždy pouze z nesplacené části úvěru (Šoba, Širůček, Ptáček, 2013, s. 121).

2.3 Úrokové standardy

Z definice (2) vyplývá, že je výpočet úroku závislý na době, kdy je peněžní částka (kapitál) zapůjčena. Dobu, za kterou počítáme úrok, nazýváme dobou splatnosti (Radová, Dvořák, Málek, 2013, s. 24).

Pro vyjádření doby splatnosti dělíme dobu splatnosti ve dnech délkou roku ve dnech a lze využít následující kódy:

1. Pro čitatel
   - ACT – započítává se skutečný počet dní smluvního vztahu a obvykle se neuvažuje o prvním dnu.
   - 30E – celé měsíce se započítávají bez ohledu na skutečný počet dní jako 30 dní.
   - 30A – liší se od 30E maximálně o jeden den, který je započten pouze v případě, že konec smluvního vztahu připadne na 31. den v měsíci.
2. Pro jmenovatel
   - rok jako 365 dní (resp. 366 pro přestupný rok).
   - rok jako 360 dní.

Kombinací uvedených možností dostáváme různé standardy pro stanovení doby splatnosti (Radová, Dvořák, Málek, 2013, s. 28).

2.3.1 Německý standard

Jedná se o jednoduchý a hojně používaný standard, který se označuje jako 30E/360. Tento standard říká, že každý měsíc má 30 dní, a to bez ohledu na jeho skutečný počet dní, a každý rok má 360 dní. Nemusí se tedy rozlišovat, zdali má měsíc 28, 30 nebo 31 dní (Šoba, Širůček, Ptáček, 2013, s. 14).

2.3.2 Francouzský standard

Tento standard se označuje jako ACT/360 a oproti německému standardu počítá se skutečným počtem dní v měsíci a s 360 dny v roce. Zatímco při německém standardu má každý měsíc 30 dní, zde již musíme rozlišovat měsíc únor s 28 dny nebo březen s 31 dny (Šoba, Širůček, Ptáček, 2013, s. 15).

2.3.3 Anglický standard

Poslední standard označujeme jako ACT/365 a počítá se skutečným počtem dní jak v měsíci, tak v roce. V případě přestupného roku počítá s 366 dny, v opačném případě s 365 dny v roce (Šoba, Širůček, Ptáček, 2013, s. 15).

2.4 Typy úročení

Úročení představuje způsob, jakým se započítávají úroky. Obecně existují dva základní typy úročení (Šoba, Širůček, Ptáček, 2013, s. 21). Každý typ se liší způsobem, jakým se daný úrok napočítává, proto je důležité znát, jaký typ úročení je zvolen.

2.4.1 Jednoduché úročení

U jednoduchého úročení dochází stále pouze k úročení původního kapitálu, to znamená, že se neúročí připsané úroky z předchozích úrokovacích období. Je zde využito aritmetické posloupnosti (Šoba, Širůček, Ptáček, 2013, s. 21).

Budoucí hodnotu kapitálu vypočítáme z následujícího vzorce:

$$K_n = K_0 \times (1 + r \times n) \tag{3}$$

Kde K_n je budoucí hodnota kapitálu, K_0 je současná hodnota kapitálu, r je úroková sazba vyjádřená jako desetinné číslo a n je počet úrokových období nejčastěji v letech (Radová, Dvořák, Málek, 2013, s. 37).

2.4.2 Složené úročení

Kromě původního kapitálu se zde úročí i úroky připsané za předchozí úroková období a dochází tak k připsání úroků z úroků. Zatímco jednoduché úročení využívá aritmetické posloupnosti, zde je využita geometrická posloupnost (Šoba, Širůček, Ptáček, 2013, s. 21).

Budoucí hodnota kapitálu bude vyjádřena následujícím způsobem:

$$K_n = K_0 \times (1 + r)^n \tag{4}$$

Kde K_n je budoucí hodnota kapitálu, K_0 současná hodnota kapitálu, r je úroková sazba vyjádřená jako desetinné číslo a n je počet úrokových období nejčastěji v letech (Radová, Dvořák, Málek, 2013, s. 49).

Rozdíl mezi jednoduchým a složeným úročením zobrazuje následující tabulka, ze které je patrné, že z pohledu dlužníka je složený úrok méně výhodný, stejně tak jako jednoduché úročení z pohledu věřitele.

Tab. 1 Jednoduché a složené úročení – úroková sazba 10 %

Typ úročení	Počet úrokovacích období		
	0	**1**	**2**
Jednoduché	100 Kč	110 (100 + 10) Kč	120 (100 + 10 + 10) Kč
Složené	100 Kč	110 (100 + 10) Kč	121 (100 + 10 + **11**) Kč

Zdroj: (Šoba, Širůček, Ptáček, 2013, s. 21)

2.5 Úmor

Za úmor považujeme rozdíl mezi anuitou a úrokem. Stav úvěru v běžném období je stav úvěru v předchozím období snížený o úmor v běžném období (Radová, Dvořák, Málek, 2013, s. 162). Úmor současné splátky zjistíme tedy jako rozdíl mezi aktuální splátkou a úrokem (Radová, Dvořák, Málek, 2013, s. 156).

Dle Šoba, Širůčka a Ptáčka (Šoba, Širůček, Ptáček, 2013, s. 121) představuje úmor postupné splacení zapůjčené částky. Jedná se tedy o umořování jistiny, pro které platí, že součet úmorové části ve všech splátkách úvěrů je roven zapůjčené částce, resp. poskytnutému úvěru.

2.6 Způsoby splácení úvěru

Splacení úvěrů se obecně označuje jako umořování a probíhá dle umořovacího plánu úvěru. Tento umořovací plán se také označuje jako splátkový kalendář. Jedná se o harmonogram, který je schválen jak dlužníkem, tak i věřitelem, obsahující termíny plateb jednotlivých splátek (Šoba, Širůček, Ptáček, 2013, s. 121). Obecně rozlišujeme dva způsoby splácení úvěru, které jsou popsány v následujících podkapitolách.

2.6.1 Splácení stejnými splátkami

Splácení stejnými splátkami neboli anuitní typ splácení se využívá především u hypotečních a spotřebitelských úvěrů. Principem je, že dlužník platí stále stejnou výši splátky (Šoba, Širůček, Ptáček, 2013, s. 124).

Při stanovení anuity se vychází z úloh o důchodu. Na stejnou problematiku se však díváme obráceným způsobem. Počáteční hodnotu úvěru lze pokládat za počáteční hodnotu důchodu a jednotlivé anuity pokládáme za výplaty důchodu (Šoba, Širůček, Ptáček, 2013, s. 124).

Při výpočtu se využívá tzv. diskontování (odúročení). Jedná se o výpočet současné hodnoty z hodnoty budoucí.

$$v = \frac{1}{1 + \mathrm{r}} \tag{5}$$

Kde v je diskontní faktor a r je úroková sazba za úrokové období (Šoba, Širůček, Ptáček, 2013, s. 125).

Anuitu tak vyjádříme následovně:

$$a = \mathrm{D} \times \frac{\mathrm{r}}{1 - v^n} \tag{6}$$

Kde *a* je výše splátky (anuita), *D* představuje výši úvěru, *r* je roční úroková sazba, *v* je diskontní faktor definovaný dle vzorce (5) a *n* je doba splatnosti úvěru v letech (Šoba, Širůček, Ptáček, 2013, s. 125).

Po výpočtu anuity zbývá zjistit prvotní úrok:

$$U_1 = D_0 \times \mathrm{i} = a \times (1 - v^n) \tag{7}$$

Kde U_1 je úrok za první období, D_0 je počáteční výše úvěru (Radová, Dvořák, Málek, 2013, s. 141). Ostatní proměnné jsou definovány vztahem (6).

Po zaplacení r splátek je zbytek dluhu $\mathrm{D_r}$ a platí:

$$U_{r+1} = D_r \times \mathrm{i} = a \times (1 - v^{n-r}) \tag{8}$$

Kde U_{r+1} je úrok v období *r+1*, D_r je zůstatek úvěru *r*-tém období (Radová, Dvořák, Málek, 2013, s. 142). Ostatní proměnné jsou definovány vztahem (6).

Z definice (1) můžeme jednoduchou úpravou vypočítat úmor jako rozdíl anuity a úroku pro příslušné období. Pro jednotlivé úmory platí, že tvoří geometrickou posloupnost (Radová, Dvořák, Málek, 2013, s. 142–143).

Může nastat situace, kdy je potřeba zjistit počet splátek na základě stanovené konstantní anuity. Pro vyčíslení počtu splátek lze využít vztahu, který získáme vyjádřením proměnné *n* ze vztahu (6) pro výpočet splátky:

$$n = \frac{ln(1 - \frac{D \times r}{a})}{\ln(v)} \tag{9}$$

Kde *n* je počet splátek, *D* je výše úvěru, r je úroková sazba za úrokové období, *a* je anuita (výše splátky) a *v* je diskontní faktor definován vztahem (5) (Šoba, Širůček, Ptáček, 2013, s. 133).

Pokud výsledná hodnota není celé číslo, musí se vždy zaokrouhlit nahoru na nejbližší přirozené celé číslo. Jestliže nastane tato situace, znamená to, že výše poslední splátky bude rozdílná (Šoba, Širůček, Ptáček, 2013, s. 133). Poté je třeba určit výši poslední anuity:

$$a_n = (\mathrm{D} - \mathrm{a} \times \left(\frac{1 - v^{n-1}}{r}\right) \times (1 + r)^n \tag{10}$$

Kde *D* je počáteční výše poskytnutého úvěru. Člen $\mathrm{a} \times \left(\frac{1-v^{n-1}}{r}\right)$ pak představuje, kolik jsme z původního dluhu (*D*) již splatili (*n-1*) anuitami ve výši (*a*). Rozdíl

$D - a \times \left(\frac{1-v^{n-1}}{r}\right)$ pak logicky znamená, kolik jsme ještě z původního dluhu (*D*) předchozími anuitami (*a*) nesplatili a kolik tak musíme zaplatit poslední *n*-tou splátku. Jelikož nesplacená část úvěru bude zaplacena až za *n* období (*n*-tou splátkou), musíme ji do tohoto období zúročit, a to právě prostřednictvím členu $(1+r)^n$ (Šoba, Širůček, Ptáček, 2013, s. 133).

2.6.2 Splácení nestejnými splátkami

V praxi se můžeme setkat i se situací, kdy úvěr nesplácíme stejnými splátkami, ale výše splátky je vždy jiná. Pro tento typ splácení je charakteristické, že ačkoliv jsou jednotlivé výše splátky různé, úmorová část je zpravidla stejná, nebo se mění dopředu známým způsobem (Šoba, Širůček, Ptáček, 2013, s. 125).

Každá splátka se tedy skládá z konstantní části úvěru – úmoru a proměnlivého úroku, který závisí na aktuálním zůstatku úvěru. Tento typ splácení se také nazývá splácení konstantním úmorem. Při něm se výše jednotlivých splátek spolu s výší úroků postupně snižuje (Radová, Dvořák, Málek, 2013, s. 151–152).

Při daném typu splácení lze úmory vyjádřit následovně:

$$\text{Úmory } M_1 = M_2 = \ldots = M_n = \frac{D_0}{n} \tag{11}$$

Kde D_0 je počáteční výše úvěru a *n* počet splátek, zpravidla v letech (Radová, Dvořák, Málek, 2013, s. 151–152).

Úrok v prvním období je dán vztahem:

$$U_1 = D_0 \times i \tag{12}$$

Kde *i* představuje roční úrokovou sazbu vyjádřenou jako desetinné místo (Radová, Dvořák, Málek, 2013, s. 151–152).

Úrok pro ostatní období můžeme dle definice (10) vyjádřit následovně:

$$U_r = D_{r-1} \times i \tag{13}$$

Kde U_r je úrok za *r*-té období a D_{r-1} představuje výši dluhu za předešlé období, *i* je roční úroková sazba vyjádřena jako desetinné místo (Radová, Dvořák, Málek, 2013, s. 151–152).

Na základě výše uvedených vztahů jsme schopni stanovit umořovací plán, kde výše splátky je dána součtem úroku a úmoru v daném období. Při tomto způsobu umořování tvoří jednotlivé úrokové platby aritmetickou posloupnost, která má diferenci rovnou rozdílu kterýchkoliv dvou po sobě jdoucích plateb (Radová, Dvořák, Málek, 2013, s. 152).

Protože jednotlivé úrokové platby tvoří aritmetickou posloupnost a úmory jsou konstantní, tvoří též celkové splátky v jednotlivých obdobích aritmetickou posloupnost se stejnou diferencí (Radová, Dvořák, Málek, 2013, s. 152).

Dle zmíněných způsobů plateb je zřejmé, že při splácení stejnými splátkami (konstantní anuitou) je výše zaplacených úroků vyšší než při splácení nestejnými splátkami (konstantním úmorem). Důvodem je, že základ pro výpočet úroků (zůstatek) se snižuje pomaleji než u splácení konstantním úmorem (Radová, Dvořák, Málek, 2013, s. 153).

3 Automatizované testování

Jednou z charakteristik dnešní doby je rychlý vývoj nových technologií, a to především v oblasti ICT. Vytvářejí se stále složitější a komplexnější systémy za účelem vyhovění zákazníkových požadavků.

Firmy kladou velký důraz na kvalitní otestování jejich produktů a vlivem implementace nových požadavků dochází k prodlužování časových a finančních prostředků pro testování. Vývojáři se také potřebují ujistit, že kód, který vytvořili, funguje tak, jak by to očekávali. Proto testování softwaru jiným způsobem než manuálním „klikáním“ se stává v poslední době velmi populárním (Burns, 2012, s. 7).

3.1 Výhody

Mezi největší výhodu zavedení automatizovaného testování je především časová úspora. Dalšími bezespornými výhodami jsou pak přesnost, efektivita a rozvoj dovedností oproti manuálnímu testování. Automatizaci jako takovou lze libovolně modifikovat a přizpůsobovat (McCafrey, 2012).

3.2 Nevýhody

Automatizované testování nepřináší pouze samé výhody, jak se na první pohled může zdát. Mezi největší nevýhodu patří neustálá údržba kódu při sebemenší změně v aplikaci. To má za následek vyhraňování kapacit určených pouze pro tvorbu a údržbu automatizovaných testů. Dále je potřeba vyhradit samostatný server, kde by se tyto skripty, respektive kódy, daly samostatně testovat na bezporuchovost a bezpečnost (McCafrey, 2012).

3.3 Kategorie testů

Automatizaci zavádíme pro konkrétní typy testů, a ne pro všechny může být výhodná. Proto je třeba znát, jaké typy testů existují, abychom si lépe uvědomili, kde je nejvýhodnější tuto automatizaci zavést a kde naopak toto zavedení neprovádět. Následující podkapitoly budou zobrazovat jednotlivé navazující úrovně testů z funkčního hlediska.

3.3.1 Unit testy

Jedná se o základní typ procesu testování, které provádí přímo programátor. Tento typ testu se zabývá testováním tzv. jednotek nebo také stavebních jednotek aplikace. V teorii, pakliže budeme uvažovat o masivní procesní aplikaci, můžeme za jednotku považovat jeden modul. V praxi však za jednotku považujeme třídu nebo metodu. Jedná se o testování jednotlivých tříd a jeho specifických metod. Unit testy jsou zdobené anotacemi (Java), nebo atributy (C#) a každá metoda je test. Účelem tohoto testování je zajištění kvalitní implementace kódu. Dané testování se vyzna-

čuje rychlostí a pro jednotlivé testovací balíčky platí, že by jejich testování nemělo zabrat více než vteřinu času (Dietrich, 2014). Jedná se tedy o prvotní testování aplikace, za které zodpovídá programátor.

3.3.2 Assembly testy

Tento typ testu zpravidla navazuje po unit testu a provádí ho rovněž programátor. Test bývá sestavován manuálně na základě zkušeností vývojáře. Principem je odhalit chyby a závažné nedostatky v procesu vývoje (Downing, 2011, s. 12–14). Je tedy zjevné, že tento typ testování má za úkol otestovat komunikaci mezi jednotlivými částmi kódu z unit testu. Lze jednoznačně vydedukovat, že uplatnění automatizace pro tyto typy testů není vhodné.

3.3.3 Integrační testy

Jedná se o testování nekompatibility včetně rozhraní mezi spolupracujícími komponentami. To znamená, že integrujeme jednotlivé subkomponenty, které společně tvoří větší celek. Jako komponentu si můžeme představit kus programu, který lze testovat nezávisle na ostatních částech programu tvořící úplný systém (Ammann, 2008, s. 222–223). Integrační testování začíná na úrovni jednotlivých modulů, kde se různé jednotky a komponenty sdružují. Integrační testování začíná, jakmile programátor skončí s vývojem a provedl unit popř. assembly testy (Limaye, 2009, s. 228–231). Automatizovat tyto testy je nejméně výhodné, jelikož se neustále mění vlivem nových požadavků a implementovaná automatizace by se musela neustále upravovat.

3.3.4 Smoke testy

Jedná se o základní testování funkčnosti aplikace. Cílem je zjistit, zda je možné s aplikací pracovat a při procházení jednotlivých modulů nedojde k pádu programu. Nejedná se tedy o konkrétní testování funkčnosti, ale o zajištění, že s aplikací může normální uživatel bez problému pracovat. Smoke test je jedním ze základních kritérií pro možnost přechodu na systémové testování. Pokud se daný test nevydaří a dojde k nečekaným pádům, musí být zahájeny okamžité nápravy. Aplikace se do dalších částí testování nedostane, dokud neprojde daným testem (Limaye, 2009, s. 279–280).

3.3.5 Systémové testy

Jedná se o tzv. end-to-end testování, jehož cílem je dohlédnout, že systém funguje dle očekávání. Systémové testování provádějí testeři, kteří zde zaujímají roli uživatele. Jedná se o procesní testování, které zahrnuje testovací scénáře, případy a data. Testovací případy jsou definovány odkazem na požadavky a návrhy. Spadá sem testování funkčnosti, uživatelského rozhraní, ale také testování výkonu a zabezpečení (Limaye, 2009, s. 182). Stejně tak jako u smoke testů i zde se dá do určité míry

uplatnit automatizace, a to v případě přípravě testovacích dat (Limaye, 2009, s. 226).

3.3.6 Regresní testy

Jedná se o proces opětovného testování softwaru, který byl modifikován. Regresní testování je ze všech typů časově nejrozsáhlejší a z procesního pohledu vývoje softwaru je jeho nedílnou součástí. Jedná se o testování stálé/neměnné funkčnosti částí aplikace. Ačkoliv se zprvu může zdát neopodstatněné testovat stále stejnou funkčnost pořád dokola, má toto testování své důvody. Malé změny v jedné části systému často způsobují problémy ve vzdálených částech systému, které by se měnit neměly. Regresní testování využíváme pro vyhledání právě těchto typů problémů (Ammann, 2008, s. 215–222). Z důvodu, že se dané testy téměř nemění a zabírají nejvíce času, je třeba využívat a zavádět do těchto testů automatizaci (Ammann, 2008, s. 320–337).

3.3.7 Akceptační testování

Tato fáze testování se provádí těsně před ostrým provozem aplikace a provádí si ji zákazník sám, popřípadě může k tomu využít třetí stranu. Součástí může být regresní a integrační testování. Výsledkem je poté souhlas k rutinnímu provozu. Pokud v průběhu akceptace jednotlivých částí dojde k nahlášení chyby nebo změny, je daný modul pozastaven a provedena náprava (Vymětal, 2009, s. 95).

3.4 Nástroje

Pro automatizované testování existuje nepřeberná spousta nástrojů. Budou zde popsány pouze nástroje, které mohou být v dané práci uplatněny, to znamená typu open-source. Tyto nástroje slouží pro usnadnění práce při zavádění automatizace do konkrétních testů. Každý nástroj je specifický svým použitím a může být vhodný pro jiné typy testů. Daná podkapitola poukáže na nejznámější a nejpoužívanější nástroje a jejich uplatnění v současné době.

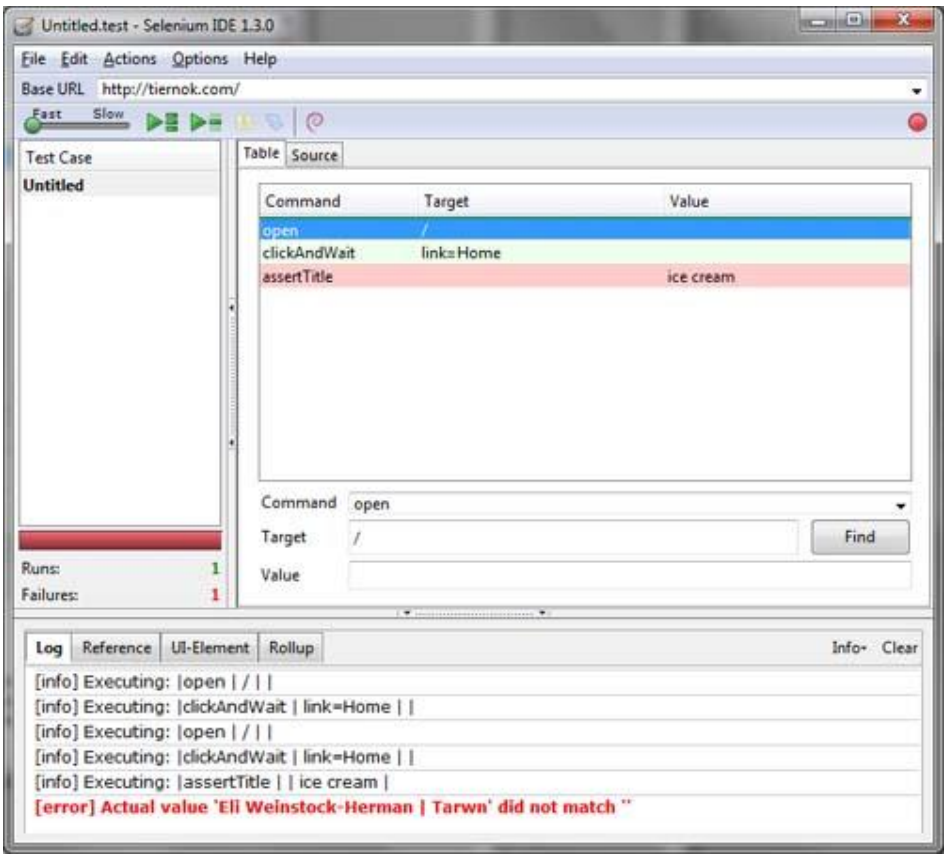
3.4.1 Selenium

Selenium je jedno z nejrozšířenějších testovacích prostředí na světě, které se v současné době používá. Jedná se o open-source projekt, který umožňuje jak testerům, tak developerům vyvíjet funkční testy spouštějící se v prohlížečích. Selenium pracuje na jakémkoliv prohlížeči, který podporuje JavaScript, jelikož bylo postaveno pomocí JavaScriptové knihovny a ta je jeho nedílnou součástí. Selenium umožňuje znázorňovat pracovní postupy testerů, které vykonávají v prohlížečích (Burns, 2012).

1. Selenium IDE

   Jedná se o add-on doplněk pro prohlížeč Firefox, puvodně vyvinutý Shinyem Kasatanim, jako způsob použití původního Selenium Core na serveru. Seleni-

um Core je klíčový JavaScriptový modul, který umožňuje Seleniu řídit prohlížeč. Využívá JavaScriptových volání tak, aby mohl komunikovat s DOM (Domain Object Model). Selenium IDE bylo vyvinuto tak, aby testeři a vývojáři měli možnost zaznamenávání svých akcí prováděných přímo v prohlížeči. Tyto akce potom za využití JavaScriptu mohli modifikovat (Burns, 2012).



Obr. 1 Ukázka Selenium IDE
Zdroj: (Weinstock-Herman, 2011)

2. Selenium WebDriver

Oproti jednoduchému doplňku pro Firefox, jako tomu bylo u Selenia IDE, lze WebDriver využít pro veškeré typy prohlížečů, které podporuje (Burns, 2012). Jedná se o sbírku jazykových specifických vazeb umožňujících řízení prohlížeče. Má podporu především pro programovací jazyk Java. Nutno podotknout, že Selenium WebDriver je nástupcem Selenium RC, který se již nepoužívá. Nativní ovládání prohlížeče z pohledu lokálního uživatele nebo přes vzdálený počítač je zde řešeno za pomocí Selenium serveru (Selenium, 2014).

3.4.2 JUnit

Jedná se o open-source framework, který se široce využívá v komerční i nekomerční sféře pro testování dílčích částí zdrojového kódu v programovacím jazyce

Java. Tento framework je rozšířen do několika programovacích jazyků, jako jsou: Ada (AUnit), C# (NUnit), Python (PyUnit), Delphi (DUnit). Všechny tyto abstrakce jsou známé pod souhrnným označením jako xUnit. JUnit roste na popularitě, jelikož většina IDE (Integrated Development Environment) poskytuje pluginy na podporu spouštění testovacích jednotek Junit (Borba, 2010, s. 26). Obrázek níže slouží jako ukázka.

```
package identifier;

import org.junit.Test;
import org.junit.Assert;

public class IdentifierTestCase {

    protected Identifier id;

    @Test
    public void validate1() {
        id = new Identifier();
        boolean result = id.validateIdentifier("a1");
        Assert.assertEquals(true, result);
    }
}
```

Obr. 2 Ukázka použití JUnit
Zdroj: (Borba, 2010, s. 27)

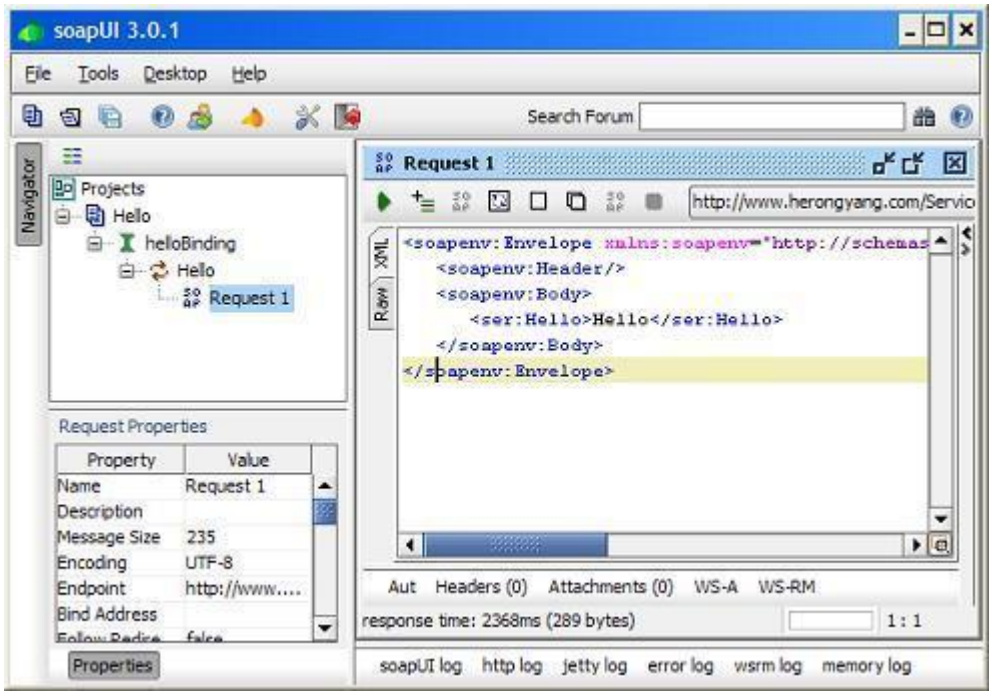
JUnit je anotačně založený flexibilní framework. Využívá statickou třídu `org.junit.Assert` se statickými přetíženými metodami. Tyto metody se využívají pro kontrolu mezi očekávanými a skutečnými výstupy testovací metody. Třída `Assert` poskytuje metody pro porovnání libovolných datových typů. Dále platí, že anotací `@Test` reprezentujeme testovací metodu a jakákoliv veřejně přístupná metoda může obsahovat danou anotaci (Acharya, 2014).

Nejnovější verze JUnit 4 má nové anotace `@Before` a `@After`, které nahrazují předešlé anotace `@setUp` a `@tearDown` používané v JUnit 3. Veřejné metody označené anotací `@Before` budou spuštěny před metodami označenými `@Test`. Naopak je tomu u anotace `@After`. Zcela novými anotacemi jsou `@BeforeClass` a `@AfterClass`, které jsou obdobné jako předešlé, ale využívají se pro veřejné statické metody (Acharya, 2014).

3.4.3 SoapUI

Jedná se o multiplatformní open-source řešení funkčního testování. Umožňuje snadno a rychle vytvářet a realizovat funkční, regresní, kompatibilní a zátěžové automatizované testy. Prostřednictvím jednoho testovacího prostředí zajistí kompletní pokrytí testu a podporuje všechny standardní protokoly a technologie. Snadno použitelné grafické rozhraní usnadňuje práci s webovými službami založené na architektuře SOAP a REST (SmartBear Software, 2015).

Tyto webové služby umožňují volání API takovým způsobem, který je nezávislý na jazykové platformě. Zpráva z jednoho konce na druhý je předána formou XML. Protokol request/response mezi klientem a serverem je definován ve specifikaci SOAP. Způsob, jakým se popisuje služba SOAP, je definován ve Web Services Description Language (WSDL) (Siriwardena, 2014 s. 1–11).



Obr. 3 Ukázka SoapUI
Zdroj: (Yang, 2009)

4 Analýza současného stavu

Tato kapitola bude popisovat aktuální problematiku testování splátkových kalendářů týmem testerů firmy EmbedIT pro běloruský bankovní sektor.

V současné době pro adekvátní nastavování parametrů slouží speciální excelovský soubor s makry, kde lze přibližně nasimulovat splátkový kalendář. Tento soubor je již zastaralý a jeho výsledky se značně liší s výsledky, které poskytuje firemní systém. Zmíněný soubor s makry dále postrádá programovou dokumentaci. Chybí tu také validace a uživatelská přívětivost. Kritickým faktorem je nastavování vlastní výše měsíční splátky (anuity) v procentech, kterou musí tester empiricky zadávat a upravovat, dokud není výsledný kalendář správně nastaven. Posledním hlavním nedostatkem je, že neobsahuje nové specifické požadavky, které jsou kladeny ze strany banky.

Jelikož se firemní systém vlivem nových požadavků neustále mění, dochází k postrádání určitého automatizovaného nástroje, který by dokázal upozorňovat na jakoukoliv změnu při generování splátkového kalendáře v rámci testování. K těmto nesrovnalostem často dochází vlivem špatného nebo neúplného nasazování databází, na kterých se má testovat. Další příčinou může být vývoj nových specifických požadavků, které nedopatřením ovlivní výpočet a zobrazení těchto kalendářů.

Pro zjednodušení uvažujeme o následujících anuitních typech úvěrů, viz podkapitola 2.6.1 *Splácení stejnými splátkami*, pro které se generují splátkové kalendáře. Společnost Home Credit, a. s. nabízí pro běloruské pobočky spotřebitelské (dále jen SS) a hotovostní (dále jen SC) anuitní typy úvěrů. Důvodem je, že se tyto dva základní typy úvěrů mírně liší ve svých výpočtech a tato kategorizace v následující analýze lépe zhodnotí jejich rozdíly.

4.1 Vstupní parametry

Před samotnou analýzou výpočtů SS a SC smluv je důležité znát základní vstupní parametry, které jsou nezbytné pro výpočet:

- **Cosua:** Fixní poplatek za čerpání úvěru.
- **Vysua:** Fixní poplatek za vedení účtu.
- **Mfsua:** Fixní poplatek za nadstandardní služby.
- **Procentuální poplatek**: Je vyjádřen procentuálně od základní výše dluhu. Reprezentuje různé formy připojištění, popř. se může jednat o poplatek za schválení hypotéky.
- **Anuita:** Jedná se o pravidelnou platbu vyjádřenou procentuálně ze základní výše úvěru s přesností na tři desetinná místa.
- **Počet splátek:** Definování celkového počtu splátek v měsících.
- **Výše dluhu:** Představuje požadovanou výši úvěru, o kterou zákazník žádá.
- **Typ smlouvy:** Pouze SS nebo SC určující rozdíly ve výpočtech.

- **Úroková míra [p.a.]**: Jedná se o navýšení zapůjčené částky za stanovené období v procentech.
- **Zaokrouhlení:** Vyjadřuje jednotkové zaokrouhlení všech splátek ve splátkovém kalendáři, viz podkapitola níže.
- **Odložení úroku (DIR):** Vyjadřuje počet měsíců s odloženým úrokem.
- **Odložení splátek (DP):** Vyjadřuje počet měsíců s odloženými splátkami.
- **Datum podpisu:** Datum, kdy byla smlouva podepsána a od kterého je smlouva o úvěru platná a účinná (Ipser, 2012).

4.2 Zaokrouhlování

Při analýze současného stavu přibyl nový požadavek na zaokrouhlování. Na základě volby zaokrouhlení se zaokrouhluje zadaná anuita, fixní poplatek, procentuální poplatek a vypočítaný úrok přímo. Úmor a zůstatek je zaokrouhlen nepřímo vlivem zaokrouhlení zmíněných parametrů. Zaokrouhlování probíhá dle zvolené hodnoty tzv. *R_N*, která může mít rozmezí 0,001–10 000 000 (Dirga, 2014).

Hodnota, která má být zaokrouhlena, se řídí následující definicí:

$$Z = VALUE / R\_N \tag{14}$$

Kde *Z* je tzv. rozhodující člen, *VALUE* představuje konkrétní hodnotu, která má být zaokrouhlena a *R_N* je zvolená jednotka zaokrouhlení (Dirga, 2014).

Z rozhodujícího členu vezmeme jeho frakci za desetinnou čárkou, na jehož základě probíhá zaokrouhlení:

1. **V případě, že Z frakce ≥ 0,5:**

$$VALUE = (Z - Z\ frakce) \times R\_N + R\_N \tag{15}$$

2. **V případě, že Z frakce < 0,5:**

$$VALUE = (Z - Z\ frakce) \times R\_N \tag{16}$$

Je patrné, že hodnota je zaokrouhlena dle *R_N* směrem nahoru, nebo dolů (Dirga, 2014).

Příklad A – zaokrouhlení směrem nahoru:

1. Zvolíme *R_N* = 20 a *VALUE* = 153 256.
2. Z definice (14) vypočítáme *Z* = 153 256/20 = 7662,8.
3. Určíme *Z* frakci = 0,8.
4. *Z* frakce ≥ 0,5 použijeme vzorec dle definice (15).
5. $(7662{,}8 - 0{,}8) \times 20 + 20 =$ **153 260.**

Příklad B – zaokrouhlení směrem dolů:

1. Zvolíme *R_N* = 20 a *VALUE* = 185 645.
2. Z definice (14) vypočítáme *Z* = 185 645/20 = 9282,25.
3. Určíme *Z* frakci = 0,25.
4. *Z* frakce < 0,5 použijeme vzorec dle definice (16).
5. $(9282{,}25 - 0{,}25) \times 20 =$ **185 640.**

4.3 Data měsíčních splátek

Data měsíčních splátek nebo také data splatnosti měsíčních splátek představují data, kdy mají být peněžní prostředky odepsány z klientova účtu, aby splnil své peněžité úvazky (Ipser, 2012).

Dlužník prostřednictvím splátkového kalendáře vidí všechna data měsíčních splátek. Mezi každými měsíčními splátkami je právě třicetidenní rozmezí shodné pro oba typy smluv. Datum prvotní splátky se odvíjí od data podpisu smlouvy o úvěru. Počet dní mezi datem první splátky a datem podpisu je také třicetidenní. Tato data měsíčních splátek jsou totožná pro SS i SC smlouvy (Ipser, 2012).

4.4 Jednoduchý splátkový kalendář

Za jednoduchý splátkový kalendář považujeme takový kalendář, který nezapočítává žádné odklady úroků nebo splátek. Může být tvořen kombinacemi zmíněných fixních nebo procentuálních poplatků.

4.4.1 Výpočet úroků

Úroky měsíčních splátek jsou vypočítávány na základě vzorce:

$$P = \mathrm{S} \times \left(\left(\frac{1 + X}{D} \right)^{30} - 1 \right) \tag{17}$$

Kde *P* je úrok měsíční splátky, *S* je výše aktuálního dluhu, *D* je počet dní v kalendářním roce. Pro nepřestupný rok se jedná o *365* dní, pro přestupný *366* dní. *X* je úroková míra vyjádřena jako desetinné číslo. Konstanta *30* vyjadřuje počet dní dané splátky (Ipser, 2012).

Na základě uvedeného vzorce lze snadno odvodit, že se jedná o anglický standard. Protože se jednotlivá data měsíčních splátek generují v třicetidenním rozmezí, lze tento standard na základě podkapitoly 2.3 *Úrokové standardy* vyjádřit jako anglický standard ACT 30/365 (366).

SS smlouva do úroku prvotní splátky nezapočítává den data podpisu. Proto pro SS smlouvu má vzorec (17) za konstantu číslo *29*, nikoliv *30*. Je tedy patrné, že

spotřebitelské úvěry SS mají úrok prvotní splátky nižší právě o jeden den (Ipser, 2012).

Přechod mezi nepřestupným a přestupným rokem

V případě přechodu mezi nepřestupným a přestupným rokem je zapotřebí při výpočtu úroku zohlednit, jaká část měsíční splátky spadá do předchozího nepřestupného a následujícího přestupného roku. Vzorec (17) pro výpočet úroku se tak musí upravit, viz následující příklad:

Datum prosincové splátky je 21. 12. 2015 (nepřestupný rok).
Datum lednové splátky je 20. 1. 2016 (přestupný rok).

Úrok pro období spadajícího do přestupného roku vypočítáme následovně:

$$P = \mathrm{S} \times \left(\left(\frac{1 + X}{366} \right)^{20} - 1 \right)$$

Úrok pro období spadajícího do nepřestupného roku vypočítáme tímto způsobem:

$$P = \mathrm{S} \times \left(\left(\frac{1 + X}{365} \right)^{10} - 1 \right)$$

Výsledný úrok pro lednovou splátku je roven součtu těchto dvou vzorců.

Přechod mezi přestupným a nepřestupným rokem

V případě přechodu mezi přestupným a nepřestupným rokem musíme při výpočtu úroku opět zohlednit část měsíční splátky spadající do přestupného a následujícího nepřestupného roku. Vzorec (17) tak upravíme, viz následující příklad:

Datum prosincové splátky je 21. 12. 2016 (přestupný rok).
Datum lednové splátky je 20. 1. 2017 (nepřestupný rok).

Úrok pro období spadajícího do nepřestupného roku vypočítáme následovně:

$$\mathrm{P} = \mathrm{S} \times \left(\left(\frac{1 + \mathrm{X}}{365} \right)^{20} - 1 \right)$$

Úrok pro období spadajícího do přestupného roku vypočítáme tímto způsobem:

$$\mathrm{P} = \mathrm{S} \times \left(\left(\frac{1 + \mathrm{X}}{366} \right)^{10} - 1 \right)$$

Výsledný úrok pro lednovou splátku je roven součtu těchto dvou vzorců.

4.4.2 Výpočet úmoru

Z definice (1) a kapitoly 5 *Úmor* lze jednoduchou úpravou na základě nově získaných poznatků definovat výpočet úmoru následovně:

$$\text{úmor} = anuita - (\text{úrok} + fp + pp) \tag{18}$$

Kde *anuita* je výše měsíční splátky, *úrok* je vypočítaný úrok za dané období, *fp* je fixní poplatek a *pp* procentuální poplatek.

4.4.3 Výpočet poslední výše anuity

Stanovení poslední výše anuity lze jednoduše odvodit následujícím vzorcem:

$$anuita = \text{úmor} + \text{úrok} + fp + pp \tag{19}$$

Kde *anuita* představuje poslední výši splátky, *úmor* je zbývající nesplacená dlužná část, *úrok* je úrokovou částí za poslední splátku, *fp* je fixní poplatek a *pp* procentuální poplatek. Dle podkapitoly 2.6.1 *Splácení stejnými splátkami* je zřejmé, že může dojít k situaci, kdy je poslední výše anuity jiná než ostatní konstantní anuity.

4.5 Splátkový kalendář s odkladem splátek (DP)

Pokud z nějakého důvodu nemůže dlužník začít splácet dluh, může zažádat o odklad splátek tzv. DP (deffered payment). Odklad splátek je možné provézt pouze před začátkem prvotního splácení. Dlužníkovi se však započítává úrok za odložené měsíce. Po dobu odložených měsíců se nezapočítávají poplatky (pokud nějaké jsou). Prvotní splátka po odložených měsících je tak navýšena o sumu úroků z odložených měsíců. Počet odložených splátek navyšuje celkový počet splátek ve splátkovém kalendáři (Ipser, 2012).

Jestliže splátce zažádá o DP, je zapotřebí vypočítat úrok prvotní splátky dle vzorce (17), který se následovně upraví:

$$P = \mathrm{S} \times \left(\left(\frac{1 + X}{365} \right)^{30+(30 \times N)} - 1 \right) \tag{20}$$

Kde nová hodnota *N* symbolizuje počet měsíců s odloženou splátkou. Jelikož vycházíme ze znalosti, že počet dnů mezi jednotlivými splátkami je přesně třicetidenní, vynásobíme počet odložených měsíců konstantou *30*. Nesmíme také zapomenout zahrnout i úrok za prvotní splátku, proto přičteme konstantu *30* (Ipser, 2012).

Z podkapitoly 4.4.1 *Výpočet úroků* plyne, že SS smlouvy mají úrok prvotní splátky o jeden den nižší, protože se zde úrok započítává od následujícího dne, kdy byla smlouva podepsána. Pro odložené splátky SS smluv by se vzorec (17) upravil následovně:

$$P = \mathrm{S} \times \left(\left(\frac{1 + X}{365} \right)^{29+(30 \times N)} - 1 \right) \tag{21}$$

4.5.1 Přetečení úroků

Pokud dlužník zažádal o odklad splátek po delší dobu nebo je měsíční splátka nízká a úroková míra vysoká, může nastat situace, kdy je suma úroků vyšší než anuita. Při prvotní řádné splátce tak může dojít k přetečení úroků za odložené splátky. V takovém případě je úrok roven rozdílu anuity a sumy všech poplatků (pokud nějaké jsou) a zbylé úroky přetékají do další splátky. Přetečené úroky jsou pak navýšeny o úrok za aktuální měsíc. Přetečené úroky však nenavyšují aktuální zůstatek dluhu, od kterého se odvíjí výpočet úroku měsíční splátky. Ve výpočtech musí být opět zohledněn pohyb z přestupného roku na nepřestupný a naopak (Ipser, 2012).

4.6 Splátkový kalendář s odkladem úroků (DIR)

Dlužník může zažádat o odklad úroku, tzv. DIR (deferred interest rate), kdy po celou dobu jejich odložení splácí pouze úmor a poplatky. Je tedy patrné, že zatímco při odložení splátek se poplatky za odložené splátky nezapočítávají, při odložených úrocích ano. Odložení úroku nenavyšuje celkový počet měsíčních splátek, jako tomu bylo v případě odložení splátek DP. Úroková míra je pak přepočítána v závislosti na počtu měsíců s odloženými úroky následovně:

$$DIR = \left(\frac{T}{N}\right) \times Ir \tag{22}$$

Kde *DIR* je odložená úroková sazba, *T* představuje celkový počet splátek, *N* je zbývající počet splátek a *Ir* je původní úroková míra vyjádřena v procentech (Ipser, 2012).

DIR stejně jako DP lze uplatnit před prvotní splátkou, proto se spolu nedají kombinovat (Ipser, 2012).

4.7 Installment holidays (IH)

Klient má nově možnost v libovolném měsíci, vyjímaje první a poslední datum měsíční splátky, zažádat o tzv. installment holidays, dále jen IH. Jedná se o obdobný princip jako tomu je u DP, viz podkapitola 4.5. Maximální doba odložení splátek je zatím určena na délku dvou měsíců. Jedná se o novou funkcionalitu, kde lze očekávat změnu maximálního počtu měsíců s IH. Stejně jako u DP i zde dochází k napočítávání úroků za každý odložený měsíc. Pokud splátkový kalendář obsahuje fixní nebo procentuální měsíční poplatky, nejsou během IH napočítávány (Dirga, 2015).

IH lze kombinovat jak s jednoduchými, tak s buď DP, nebo DIR splátkovými kalendáři. Měsíční splátky s IH se však nesmí krýt s měsíčními splátkami DP, nebo DIR. Protože o IH lze zažádat až v průběhu splácení, nelze již výši měsíční splátky upravovat. To má za následek, že výše poslední anuity by mohla být vyšší než předešlé neměnné anuity. Proto pokud je využit IH, splátkový kalendář navyšuje počet

měsíčních splátek, dokud výše poslední anuity nebude stejná nebo nižší nežli předešlé, neměnné výše splátek (Dirga, 2015). Pro lepší přehlednost poslouží následující příklad:

Dlužná částka: 2 000 000. Úroková míra: 60 %. Počet splátek: 12. IH začátek: 4. měsíc. IH: 2 měsíce.

Tab. 2 Princip Installment holidays

Splátka	Anuita	Úmor	Úrok	Zůstatek
1.	22570	12570,00	10000,00	187430,00
2.	22570	13198,50	9371,50	174231,50
3.	22570	13858,43	8711,58	160373,08
IH				
4.	0,00	0,00	8018,65	160373,08
5.	0,00	0,00	8018,65	160373,08
6.	22570	0,00	22570,00	160373,08
7.	22570	14551,35	8018,65	145821,73
8.	22570	15278,91	7291,09	130542,82
9.	22570	16042,86	6527,14	114499,96
10.	22570	16845,00	5725,00	97654,95
11.	22570	17687,25	4882,75	79967,70
12.	22570	18571,61	3998,39	61396,09
13.	22570	19500,20	3069,80	41895,89
14.	22570	20475,21	2094,79	21420,69
15.	21420,69	21420,69	0,00	0,00

Veškerá data a jejich výpočet jsou fiktivní, zde je pouze poukázán princip výpočtu IH. Z tabulky je patrné, že IH stejně jako DP navyšuje celkový počet splátek a dochází k započítávání úroků za odložené měsíce. Pokud dlužník zažádá o DP, přenastaví se splátkový kalendář a předešlá výše měsíční splátky se navýší tak, aby poslední anuita byla stejná nebo nižší. V případě IH je však část splátek splacena a výše anuity se nesmí změnit. Proto dochází k navýšení počtu měsíčních splátek, dokud poslední anuita není stejná, nebo nižší než ostatní. Proto, jak je z tabulky patrné, ačkoliv je doba odložení splátek pro IH 2 měsíce a celkový počet měsíců je 12, je konečný počet měsíců 15, nikoliv 14 (Dirga, 2015).

4.8 Restrukturalizace

Poslední novou funkcionalitou je tzv. restrukturalizace. Klient se při splácení může dostat do tíživé finanční situace a potřebuje přehodnotit splátkový kalendář tak, aby mohl splácet menší měsíční splátky. Klient pouze zvolí měsíc splátky, kdy má proběhnout restrukturalizace. Od zvoleného měsíce dojde k navýšení počtů splátek o šest, nebo dvanáct měsíců a dojde k přepočítání výše anuitní splátky. Pokud byl

originální počet splátek menší nebo roven 12, bude počet splátek navýšen o 6. V opačném případě dojde k navýšení počtů splátek o 12, viz následující příklad:

Dlužná částka: 2 000 000. Úroková míra: 60 %. Počet splátek: 12 Rest.: 4. měsíc.

Tab. 2 Princip restrukturalizace

Splátka	Anuita	Úmor	Úrok	Zůstatek
1	22570	12570,00	10000,00	187430,00
2	22570	13198,50	9371,50	174231,50
3	22570	13858,43	8711,58	160373,08
Restrukturalizace				
4	15460	7441,35	8018,65	152931,73
5	15460	7813,41	7646,59	145118,32
6	15460	8204,08	7255,92	136914,23
7	15460	8614,29	6845,71	128299,94
8	15460	9045,00	6415,00	119254,94
9	15460	9497,25	5962,75	109757,69
10	15460	9972,12	5487,88	99785,57
11	15460	10470,72	4989,28	89314,85
12	15460	10994,26	4465,74	78320,59
13	15460	11543,97	3916,03	66776,62
14	15460	12121,17	3338,83	54655,45
15	15460	12727,23	2732,77	41928,23
16	15460	13363,59	2096,41	28564,64
17	15460	14031,77	1428,23	14532,87
18	14532,87	14532,87	0,00	0,00

Tabulka s fiktivními daty a výpočtem názorně ukazuje přepočet anuity a navýšení originálního počtu měsíců. Restrukturalizace může být kombinována s ostatními typy splátkových kalendářů popsaných v kapitole 4 *Analýza současného stavu*, vyjímajíc IH (Dirga, 2015).

4.9 Nová funkčnost

V průběhu sepisování diplomové práce vznikl nejnovější požadavek na výpočet splátkových kalendářů, který bude totožný pro oba typy smluv (SS i SC). Z tohoto důvodu lze pro zjednodušení využít označení S*.

Zákazník bude mít možnost zvolit datum splatnosti pro všechny měsíční splátky na určitý den v měsíci (například každý 20. den daného měsíce). Vybrány mohou být pouze kalendářní dny odpovídající 1. až 27. dni v měsíci (Samek, 2015).

Dále musí platit podmínka, že počet kalendářních dní mezi prvotní splátkou a datem podpisu může být v maximálním rozmezí 20 až 45 kalendářních dní, viz následující příklad:

Datum podpisu: 20. 04. 2014
Datum splatnosti: 03. 06. 2014

Datum splatnosti má vybraný 3. kalendářní den v měsíci, což odpovídá prvotní podmínce. Počet kalendářních dní mezi datem podpisu a prvotní splátkou činí přesně 44 dní. Druhá podmínka je tedy také splněna a může být generován splátkový kalendář:

1. splátka: 03. 06. 2014 …
2. splátka: 03. 07. 2014 …
3. splátka: 03. 08. 2014 …

Opět bude potřeba upravit vzorec (17) pro výpočet úroku měsíční splátky:

$$P = \mathrm{S} \times \left(\left(\frac{1 + X}{365} \right)^{DNY} - 1 \right) \tag{23}$$

Kde *DNY* představuje počet dní mezi aktuální a předešlou splátkou. Pokud se jedná o prvotní splátku *DNY*, představuje počet dní mezi první splátkou a datem podpisu. Pro lepší pochopení lze využít předchozí příklad:

Datum podpisu: 20. 04. 2014
Datum splatnosti: 03. 06. 2014

Pro výpočet úroku prvotní splátky je třeba stanovit počet kalendářních dní mezi datem podpisu a prvotní splátkou, která činí přesně 44 dní.

1. splátka: 03. 06. 2014. Zjištěné rozmezí dosadíme do vzorce (23):

$$P = \mathrm{S} \times \left(\left(\frac{1 + X}{365} \right)^{44} - 1 \right)$$

2. splátka: 03. 07. 2014. Počet dní mezi 1. a 2. splátkou dosadíme do vzorce (23):

$$P = \mathrm{S} \times \left(\left(\frac{1 + X}{365} \right)^{30} - 1 \right)$$

3. splátka 03. 08. 2014 Počet dní mezi 2. a 3. splátkou dosadíme do vzorce (23):

$$P = \mathrm{S} \times \left(\left(\frac{1 + X}{365} \right)^{31} - 1 \right)$$

Pro jednotlivé splátky je zapotřebí rozlišit, došlo-li k přestupu z nepřestupného na přestupný rok a naopak. Pro novou funkčnost metody IH, restrukturalizace, DP a DIR zůstávají stejné. Oproti IH a restrukturalizaci není tato funkčnost firmou prozatím naimplementována (Samek, 2015).

5 Návrh

Tato kapitola se zabývá řešením celkového návrhu simulátoru splátkového kalendáře včetně uplatnění automatizované funkce pro testování jeho vstupních parametrů. Základním stavebním kamenem jsou softwarové požadavky, které budou v dalším kroku blíže popsány. Díky těmto požadavkům bude v dalším postupu vytvořen návrh grafického uživatelského rozhraní. Konečným krokem je analýza metodik a jejich konkrétní výběr pro splnění vzniklých softwarových požadavků.

5.1 Softwarové požadavky

Pro zákazníka je často frustrující, pokud daný program neodpovídá jeho očekávání a nesplňuje podstatné úlohy, které by od něj očekával. Vývojář poté musí upravovat celý program, což stojí další čas a peníze. Všechny tyto neduhy vznikají špatným porozuměním a dá se jimi snadno předejít za pomoci definování softwarových požadavků (Wiegers, Beatty, 2013).

Často se setkáváme s pojmy, jako jsou obchodní nebo funkční specifikace. Za standardní označení se používá název softwarové požadavky, které popisují chování systému. Jedná se o požadavky toho, co by mělo být naimplementováno, nikoliv toho, jak by se daná implementace měla provést (Wiegers, Beatty, 2013).

Softwarové požadavky vznikly syntézou uživatelských požadavků. V tomto případě je za uživatele považován tým testerů firmy EmbedIT. Požadavky jsou popsány ve struktuře, ve které dochází k bližšímu popisu jednotlivých dílčích bodů.

1. Software umožní simulovat všechny typy splátek vycházejícího ze současného stavu.
    a. DP splátky.
    b. DIR splátky.
    c. Klasické splátky.
    d. Restrukturalizace.
    e. IH.
    f. Kombinace fixních poplatků a procentuálního poplatku.
2. Software umožní volbu typů smluv.
    a. SC smlouva.
    b. SS smlouva.
    c. Zohlednění nové funkcionality S*.
3. Software umožní volbu zaokrouhlovacích jednotek.
    a. Rozmezí musí být: 0,001 až 1 000 000.
4. Software umožní automatizovaně dopočítat adekvátní anuitu.
    a. Za adekvátní anuitu je považována taková hodnota, která povede k zobrazení nezáporných splátkových kalendářů.
    b. Za adekvátní anuitu je dále považována taková hodnota, kde poslední výše splátky bude stejná nebo nižší než ostatní neměnné výše splátek.

5. Software musí být uživatelsky přívětivý.
    a. Software musí vhodně formátovat všechny číselné vstupy a výstupy.
    b. Software musí sofistikovaně upozornit uživatele při chybějících nezbytných parametrech nebo při jejich neadekvátních hodnotách a kombinacích.
    c. Software musí zobrazit jednotlivé sumy výsledných položek.
6. Software musí umět provádět automatizovanou kontrolu.
    a. Software poskytne vhodný automatizovaný výstup.
    b. Software provede automatizovanou kontrolu na základě vloženého XML vstupního souboru.
    c. Automatizace musí byt proveditelná na všech testovacích databázích.
    d. Automatizovaná kontrola musí porovnávat výstupy ze simulátoru vůči výstupům z firemního systému.

5.2 Návrhové vzory

Návrhové vzory představují ověřené dlouhodobé postupy a poznatky vědců a výzkumníků vzniklé na základě pozorování uživatelů při práci se softwarem. Rozhraní, které podporují tyto vzory, pomáhají uživatelům rychleji dosáhnout jejich cílů než je tomu u rozhraní, které tyto vzory nepodporují. Díky těmto vzorům se uživatel v daném rozhraní lépe a rychleji zorientuje a také dochází ke snižování jeho kognitivní zátěže (Tidwell, 2011).

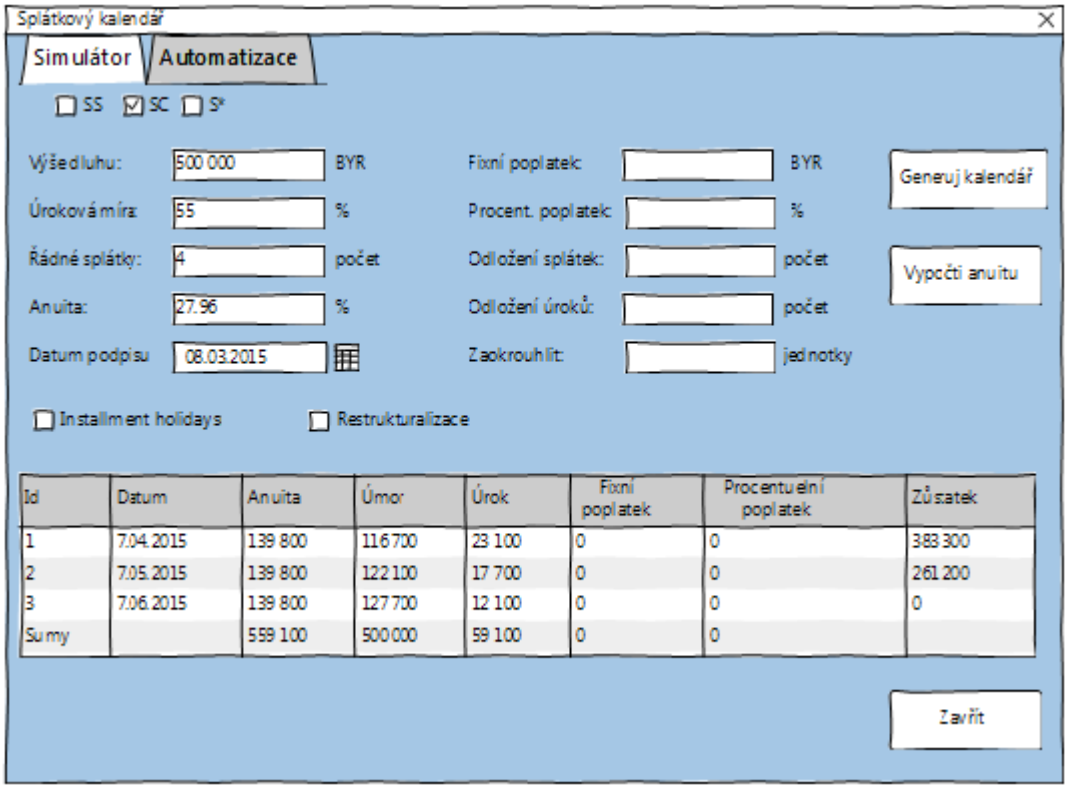
Jenifer Tidwell ve své knize popisuje mnoho návrhových vzorů, zde budou popsány pouze ty, které budou zavedeny v návrhu grafického uživatelského rozhraní.

- Good Defaults: V místech, kde je to možné, nechávat vhodné výchozí hodnoty. Je třeba si uvědomit, že pokud budeme tento vzor uplatňovat bez rozmyšlení, může dojít k opačnému efektu.
- Input Hints: V blízkosti textových polí (zpravidla vedle nich) stručně vysvětlovat obsah.
- Input Promt: Obdoba Input Hints jenom je zobrazování prováděno formou nápovědy (tooltip).
- Escape Hatch: Pokud se uživatel v aplikaci příliš zanoří nebo se ztratí, měl by mít vždy možnost vrátit se zpět, nebo ukončit aplikaci.
- Clear Entry Points: Vstupní body by měly být v aplikaci jasně viditelné.
- Smart Menu Items: Přizpůsobovat vhodné popisky (mohou být i obrázky) do menu tak, aby bylo patrné, co jeho akce provede (platí i pro linky a tlačítka).
- Bookmarks: Využití záložek pro přecházení z jednoho stavu do druhého. Výhodou je, že se uživatel dostane z každého stavu do každého.
- Button Groups: Společné funkce seskupovat do oddílů (Tidwell, 2011).

5.3 Návrh grafického uživatelského rozhraní

Na základě analýzy současného stavu bylo navrženo grafické uživatelské rozhraní. Návrh byl vytvořen v programu WireframeSketcher. Tento program umožňuje vývojářům a programátorům rychle a kvalitně vytvářet drátěné modely pro desktopové, mobilní a webové aplikace. Je dostupný také jako plug-in pro Eclipse IDE (Severin, 2008).

5.3.1 Návrh okna simulátoru splátkového kalendáře



Obr. 4 Okno simulátoru splátkového kalendáře

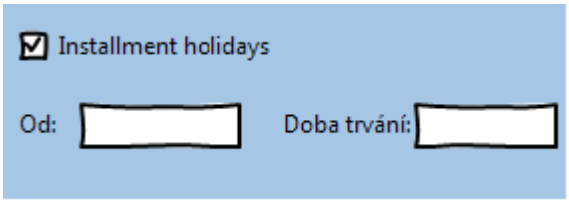
Dané okno obsahuje veškeré nezbytné parametry pro simulaci různých typů splátek. Na základě návrhového vzoru Bookmarks obsahuje vrchní část panelu záložku pro přepínání mezi dvěma okny. Pro výběr konkrétního typu smlouvy slouží vrchní checkboxy, ve kterých bude vždy jeden typ automaticky zaškrtnut při spuštění. Je zde dodržen návrhový vzor Clear Entry Points, který je pro velké množství vstupních parametrů nezbytný. Stisknutím tlačítka „Generuj kalendář“ dojde k zobrazení splátkového kalendáře do tabulky nacházející se v dolní části programu. Dalším návrhovým vzorem je Escape Hatch, díky kterému má uživatel možnost vždy odejít. Aby se uživatel lépe orientoval, nechybí také návrhový vzor Input Hints, který popisuje vstupní parametry. Tlačítka mimo jiné budou obsahovat obrázky odpovídající jejich funkčnostem – Smart Menu Items. Vstupní parametr da-

tum podpisu bude mít vždy předvyplněné aktuální datum – Good Defaults. Pro výběr data poslouží komponenta DatePicker.

Text box „fixní poplatek“ reprezentuje veškeré fixní poplatky zmíněné v podkapitole 4.1 *Vstupní parametry*. Pokud má být systém uživatelsky přívětivý, je důležité snažit se minimalizovat počet nezbytných vstupních parametrů.

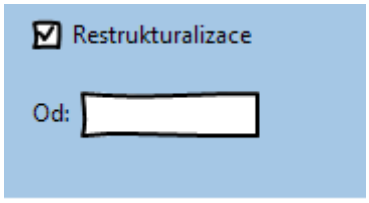
Tlačítko „Vypočti anuitu“ automaticky vypočítá anuitu, kterou zobrazí do vstupního text boxu, a vygeneruje splátkový kalendář. Jednotlivá tlačítka, vstupní textová pole i checkboxy jsou uspořádány do grup dle návrhového vzoru Button Groups. Tlačítka „Vypočti anuitu“ a „Generuj kalendář“ jsou neaktivní do doby, dokud nejsou správně vyplněny veškeré zadané hodnoty.

Při spuštění programu budou povinné hodnoty: výše dluhu, úrok, počet splátek a anuita barevně podbarveny, dokud nebudou zadány korektní hodnoty. Ostatní text boxy se podbarví v případě, že je zadaná nevalidní hodnota, popř. kombinace hodnot. V okamžiku, kdy jsou veškeré hodnoty korektní a žádný text box není podbarven, budou tlačítka „Vypočti anuitu“ a „Generuj kalendář“ aktivní. Aby uživatel nebyl zmaten mnoha vstupy, je zapotřebí některé nepovinné vstupní parametry schovat a nechat je zobrazit až v okamžiku potřeby, viz následující obrázek.



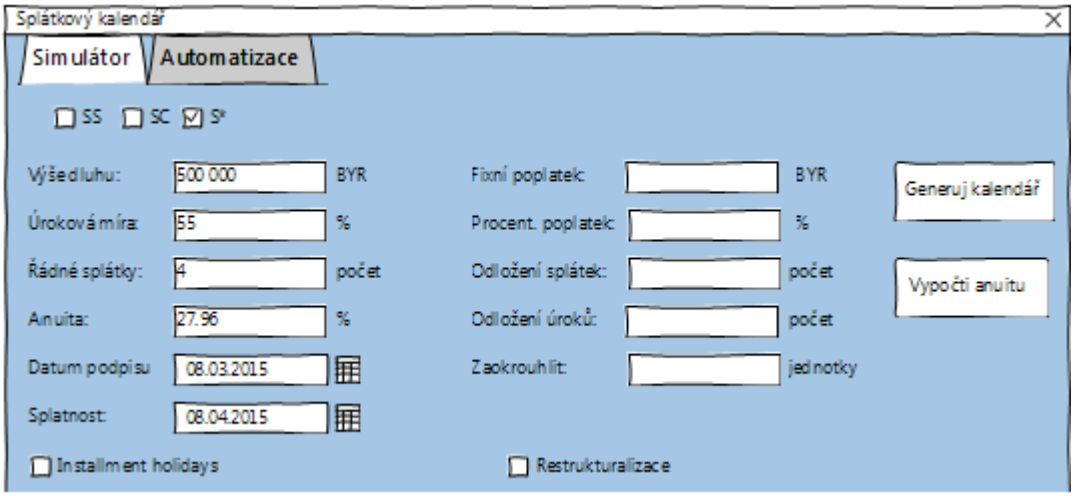
Obr. 5 Installment holidays – rozšířené GUI

Z obrázku je patrné, že se v případě potřeby, tj. je zaškrtnut checkbox „Installment holidays“, zobrazí uživateli nové vstupní parametry spadající pod požadovanou funkčnost.



Obr. 6 Restrukturalizace – rozšířené GUI

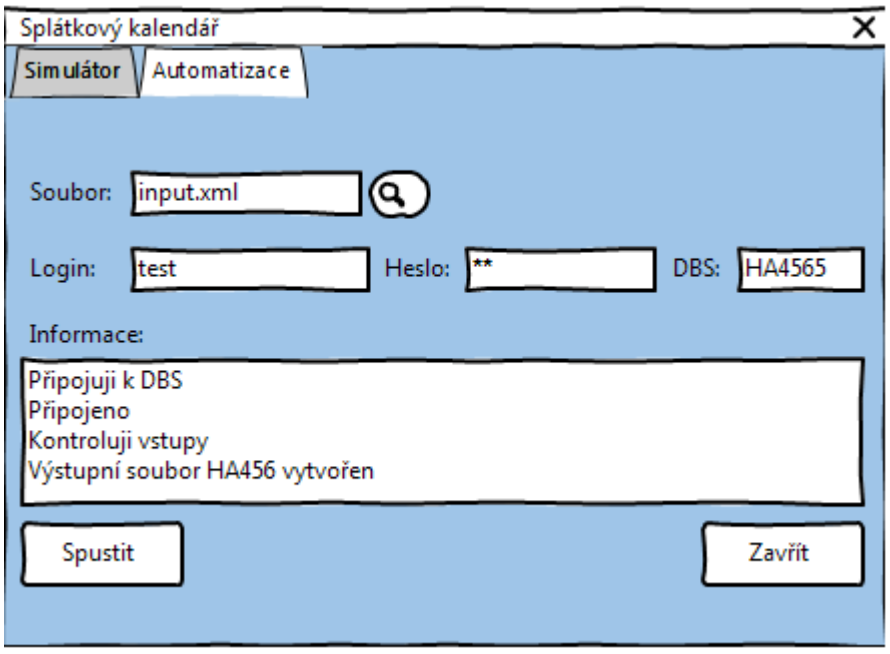
Pokud je označen checkbox „Restrukturalizace“, dojde k zobrazení text boxu žádajícího uživatele zvolit číslo splátky, od které bude restrukturalizace provedena. Toto informativní zobrazení bude využívat návrhového vzoru Input Prompt. Dle analýzy současného stavu nebudou moci být checkboxy „Installment holidays“ a „Restrukturalizace“ společně zaškrtnuty.



Obr. 7 Nová funkčnost – rozšířené GUI

Při označení checkboxu „S*“ dojde k zobrazení vstupního parametru pro zadání data splatnosti popisovaného v podkapitole 4.9 *Nová funkčnost*. Bude zde využit návrhový vzor Input Prompt, který uživatele informuje o formátu daného vstupu. Pro samotné vložení hodnoty zde bude použita komponenta DatePicker.

5.3.2 Návrh okna pro automatizovanou kontrolu



Obr. 8 Okno pro automatizované testování parametrů

Dané okno bude mít na starost provedení automatizované kontroly. Okno obsahuje textovou oblast pro zobrazování průběhu provádění automatizované kontroly. Opět se nesmí zapomenout na dodržování návrhového vzoru Escape Hatch, proto se i zde nachází tlačítko „Zavřít“. Každé tlačítko bude obsahovat kromě stručného

popisku i obrázky odpovídající jejich funkčnostem – Smart Menu Items. Pomocí návrhového vzoru Input Prompt bude textové pole pro soubor zobrazovat uživateli informaci, že je očekáván XML vstupní soubor. Pokud žádný soubor není vybrán, bude toto textové pole podbarveno a tlačítko „Spustit“ bude neaktivní. Při stisknutí tlačítka s lupou se uživateli zobrazí průzkumník, díky kterému může vybrat cestu k XML souboru. Jakmile je soubor nalezen a potvrzen výběr, zobrazí se jeho název v textovém poli pro soubor. Toto textové pole již nebude podbarveno a tlačítko „Spustit“ bude aktivní.

Pokud uživatel vyplní textová pole nutná pro přihlášení do konkrétní databáze (login, heslo, název databáze), vytvoří se HTML soubor, jehož název bude odpovídat názvu databáze, na kterém proběhla automatizovaná kontrola. Pokud připojení k dané databázi selže, nebo zde nebude nasazena procedura, uživateli se v textové oblasti pro informace zobrazí zdůvodnění.

5.4 Návrh výpočtu anuity

Ze softwarových požadavků vyplývá, že program musí umět automatizovaně dopočítat adekvátní výši anuity v závislosti na vložených vstupních hodnotách. Z poznatků plynoucích z kapitoly 2 *Úvěr* a kapitoly 4 *Analýza současného stavu* lze vzorec (6) pro výpočet anuity upravit následujícím způsobem:

$$A = (\mathrm{S} + \mathrm{fp} + \mathrm{pp}\ ) \times \frac{(X \times (X+1)^{Y})}{((X+1)^{Y} - 1)} \tag{24}$$

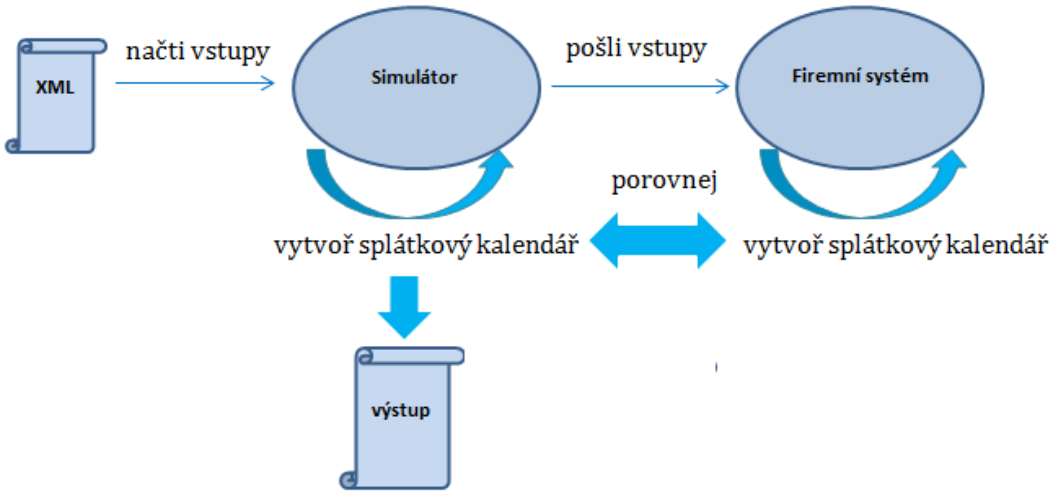
Kde *S* je výše aktuálního dluhu, *fp* je fixní měsíční poplatek, *pp* procentuální měsíční poplatek, *X* představuje měsíční úrokovou míru vyjádřenou jako desetinné číslo a *Y* představuje počet měsíčních splátek.

5.5 Návrh automatizované kontroly

Na základě předchozích analýz bylo zjištěno, že se automatizovaná kontrola nejvíce vyplatí pro regresní a smoke testy, viz podkapitola 3.3 *Kategorie testů*. Jedná se tedy o kontrolu již nasazených funkcí, kde nedochází k žádné změně.

Typy výpočtů pro IH a restrukturalizaci spadají pod novou funkčnost, která zatím není zahrnuta v regresních scénářích. Protože tyto dvě zmíněné funkčnosti mohou být stále upravovány, nebudou prozatím do automatizované kontroly zahrnuty. Nová funkčnost „S*“, popisovaná v podkapitole 4.9, není firmou ještě naimplementována, a z toho důvodu nebude do automatizované kontroly také zahrnuta.

Z analýzy rovněž vyplynulo, že při testování a vývoji na různých databázích, nemusí být vše řádně nasazeno. Tento problém potom způsobuje nesprávné generování dat. Při každém nasazování databáze je potřeba ověřovat, zda výpočetní modely pro jednotlivé splátkové kalendáře nebyly ovlivněny. Aby tato automatizovaná kontrola měla smysl, měla by automatizovaně tento problém řešit, viz následující obrázek:



Obr. 9 Návrh principu automatizované kontroly

Na základě vstupních dat definovaných v XML souboru zobrazí simulátor výsledky, které porovná s výsledky získanými z požadované firemní databáze, jež je součástí firemního systému. Výstup obou výsledků bude zanesen do patřičného výstupního souboru. V případě nerovnosti mezi jednotlivými výsledky simulátoru a firemní databáze dojde ke zvýraznění hodnoty obsažené ve výsledku simulátoru.

5.5.1 Získávání dat z firemního systému

Protože klíčovým prvkem automatizované kontroly jsou potřebná data z firemního systému, pro porovnání je třeba navrhnout, jakým způsobem bude možno tato data získat. Dle přání firmy EmbedIT zde bude využita black-box SQL funkce:

```
instalsched.gener_instalment_collection("
+ "p_credit_amount       => ?,"
+ "p_annuity             => ?,"
+ "p_payment_num         => ?,"
+ "p_annuity_date        => ?,"
+ "p_date_start_interest => ?,"
+ "p_cosua               => ?,"
+ "p_vysua               => ?,"
+ "p_mfsua               => ?,"
+ "p_interest_rate       => ?,"
+ "p_type_count          => ?,"
+ "p_def_int_period      => ?,"
+ "p_def_int_rate        => ?"+ ")
```

Ačkoliv se jedná o tzv. black-box SQL funkci, je zapotřebí definovat jednotlivé vstupní parametry:

1. `p_credit_amount` představuje výši požadovaného úvěru.
2. `p_annuity` je výše měsíční splátky.
3. `p_payment_num` je počet měsíčních splátek.
4. `p_annuity_date` je datum první splátky.
5. `p_date_start_interest` je datum, kdy se začne započítávat úrok.
6. `p_cosua` je fixní poplatek za čerpání úvěru.
7. `p_vysua` je fixní poplatek za vedení účtu.
8. `p_mfsua` je fixní poplatek za nadstandardní služby.
9. `p_interest_rate` je roční úroková míra s přesností na tři desetinná místa.
10. `p_type_count` představuje typ účtu. V případě, že se bude počítat s odloženými úroky DIR, musí být hodnota proměnné nastavena na 3, jinak na 2.
11. `p_def_int_period` představuje počet dní s odloženým úrokem.
12. `p_def_int_rate` je nová roční úroková míra, která je vypočítána dle vzorce (22) v případě, že se bude počítat splátka s odloženým úrokem.

Nevýhodou dané funkce je absence parametrů definujících zaokrouhlování a procentuální poplatek. Jelikož zaokrouhlovací jednotka může být na každé databázi nastavena manuálně, nebude v XML vstupním souboru zohledněn pouze procentuální poplatek.

Tato SQL funkce na základě definovaných vstupních parametrů provede požadované výpočty na konkrétní databázi, na které byla zavolána. Po provedení těchto výpočtů vrátí funkce požadovaná data (splátkový kalendář).

5.5.2 Struktury XML

XML formát definovalo konsorcium W3C jako formát pro přenos obecných dokumentů a dat. XML je zkratka pro eXtensible Markup Language, tj. rozšiřitelný značkovací jazyk. Návrh XML vychází ze staršího a obecnějšího standardu SGML. Sada značek ve formátu XML není pevná a může být definována pro různé sady dokumentů různě (Mlýnková, Pokorný, 2008, s. 15). Pro popis struktury XML dokumentů se používají nejznámější jazyky XML Schema a DTD. Pomocí těchto popisů se dá ověřit, zdali je XML dokument validní, tj. odpovídá jeho předepsané struktuře (Mlýnková, Pokorný, 2008, s. 21).

DTD

Jazyk DTD (Document Type Definition) umožňuje definovat elementy a atributy, které se smí v XML dokumentu používat. Jazyk DTD má postačující vyjadřovací sílu a je oblíben pro svoji jednoduchost.

- DTD může být zahrnut přímo v XML dokumentu, a to jeho přidáním do konstrukce DOCTYPE:

```
<?xml version=:"1.0" encoding "UTF-8"?>
    <!DOCTYPE osoba [
    <!ELEMENT jméno (#PCDATA)>
    <!ELEMENT příjmení (#PCDATA)> ]>
    <jméno>Filip</jméno>
    <příjmení>Walder</příjmení>
```

- Z důvodu přehlednosti je lepší, odkazovat se na něj externě:

```
<?xml version=:"1.0" encoding "UTF-8"?>
<!DOCTYPE SYSTEM "osoba.dtd">
<jméno>Filip</jméno>
<příjmení>Walder</příjmení>
```

V případě, že je potřeba vytvářet složitější struktury, popř. vyjadřovat přípustnou strukturu XML dokumentů přesněji, je tento jazyk nedostačující (Mlýnková, Pokorný, 2008, s. 47).

XML Schema

XML schéma popsané v jazyce XML Schema je XML dokumentem obsahujícím speciálně definované elementy. Tyto elementy definují strukturu XML dokumentu. Tento jazyk má řadu nesporných výhod:

- Na rozdíl od jazyka DTD nevyžaduje speciální syntax – XML schémata v jazyce XML Schema jsou opět XML dokumenty.
- Má silnou podporu datových typů (string, boolean, date) a umožňuje uživateli definovat vlastní datové typy.
- Zachovává většinu prvků jazyka DTD.
- Umožňuje přesně, tj. v rámci daného rozmezí, vyjádřit přípustné počty výskytu elementů v rámci rodičovského elementu.
- Umožňuje definovat elementy se stejným názvem, ale s různým obsahem.
- Umožňuje specifikovat unikátnost obsahu elementu, hodnoty atributu nebo jejich kombinace v rámci požadované části XML dokumentu (Mlýnková, Pokorný, 2008, s. 49).

Dle Mlýnkové a Pokorného (Mlýnková, Pokorný, 2008, s. 47) se vžilo označení XSD, tj. zkratka z anglického termínu „XML Schema Definition“, aby se nemuselo používat poměrně krkolomné označení „XML schéma popsané v jazyce XML Schema“.

5.5.3 Volba jazyka pro popis struktury XML

Pro vstupní XML soubor je zapotřebí stanovení a definování jeho přípustných struktur. Také bude důležité stanovit maximální možný počet vložených rodičovských elementů. Na základě zjištěných a popsaných výhod bude zvolen jazyk XML Schema, neboli XSD.

5.5.4 Návrh formátu výstupního souboru

Na základě softwarových požadavků je třeba zvolit vhodný výstupní formát automatizované kontroly. Toto kritérium je velice důležité, protože má zásadní vliv na uživatelovu kognitivní zátěž. Je třeba si uvědomit, že pro jedny vstupní parametry se vygenerují dvě tabulky (simulátor a databáze) splátkových kalendářů. Vhodný formát je potom zásadní pro uživatelovu rychlou orientaci a přehlednost.

Využití formátu XML

Nejjednodušším způsobem se může jevit možnost zvolit stejný formát výstupního souboru jako u vstupního. Při hlubším zamyšlení může být tento způsob naopak kontraproduktivní. Velké množství značek, které by výstupní XML soubor obsahoval, by mohlo uživatele velice mást a zpomalit tak jeho orientaci v daném souboru. Dále by uživatel viděl jednotlivé výsledky ze simulátoru a databáze pod sebou. Ukázka návrhu XML výstupu:

```
<document>
 <simulator id=1>
   <cisloSplatky>1</cisloSplatky>
   <datumSplatky>10.2.2015</datumSplátky>
   <anuita>10 000</anuita>
            …
   <cisloSplatky>2</cisloSplatky>
   <datumSplatky>10.3.2015<datumSplátky>
   <anuita>10 000<anuita>
            …
 </simulator>
 <databaze id=1>
   <cisloSplatky>1</cisloSplatky>
   <datumSplatky>10.2.2015</datumSplátky>
   <anuita>10 000</anuita>
            …
   <cisloSplatky>2</cisloSplatky>
   <datumSplatky>10.3.2015<datumSplátky>
   <anuita>10 000<anuita>
            …
 </databaze>
</document>
```

Je zjevné, že tento typ formátu pro výstup není vhodný. Pro lepší orientaci je třeba ukázat pouze konkrétní hodnoty bez značek a výsledky ze simulátoru i z databáze mít vedle sebe.

Využití formátu XLS, XLSX

Excel poskytuje prostředí, ve kterém jsou jednotlivé položky přehlednější, než je tomu v případě XML. Lze zde jednoduše přidat filtry, pozadí hlavičky tabulek

a mnohé další. Uživatelé (v tomto případě testeři firmy EmbedIT) dané formáty hojně využívají pro svoje testování a zápis výsledků. Lze tedy očekávat rychlou orientaci a vlastní designové úpravy bez sebemenších problémů. Nevýhodou těchto formátů je pouze závislost na specifických produktech pro jejich čtení a modifikaci.

Využití HTML formátu

HTML je zkratka pro Hypertext Markup Language, tj. hypertextový značkovací jazyk. HTML je určen k vytváření dokumentů obsahujících hypertextové odkazy a pokročilejší formátování (Písek, 2014, s. 15–18). Nespornou výhodou HTML formátů je jejich možnost čtení v libovolném internetovém prohlížeči (při dodržování W3C standardů). Dále se dá interně, popř. externě, využívat CSS definující vzhled stránek. Nespornou výhodou je možnost přidání JavaScriptu (skriptovací jazyk), díky kterému lze definovat různé funkcionality, které uživateli můžou zpříjemnit prohlížení dokumentu (Písek, 2014, s. 30–137).

Využití formátu PDF

PDF (Portable Document File) – jedná se o formát na ukládání a výměnu textových dokumentů, který je primárně určen pro čtení firmou Adobe Inc. Velkou předností daného formátu je, že se výsledný dokument zobrazuje na všech zařízeních stejně. Další výhodou je možnost zaheslování výsledného dokumentu, takže pouze autorizovaná osoba může dokument prohlížet. PDF využívá kompresních algoritmů zajišťujících malou velikost výsledného souboru. Do PDF je možné vkládat také poznámky a hodnoty, například využitím Adobe Acrobat softwaru (Salomon, 2007, s. 928–930).

5.5.5 Volba formátu výstupního souboru

Na základě předešlých návrhů bude zvolen HTML výstupním formátem. HTML pomocí CSS a JavaScriptu umožňuje přívětivě definovat výstup a je multiplatformní. V případě potřeby lze jednoduše převézt HTML do formátu PDF.

5.5.6 Návrh XML vstupního souboru

Daná kapitola se zabývá popisem jednotlivých XML elementů vstupního souboru pro automatizované testování. Prvotním krokem bylo definování struktury XML dokumentu pomocí zvoleného XSD souboru, který je zobrazen v příloze, Obr. 11. XML soubor bude vypadat následovně:

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes"?>
<document>
 <inputParams>
  <name>SC class</name>
  <loan>SC</loan>
  <creditAmount>566446</creditAmount>
  <annuity>29800</annuity>
```

```
  <paymentNum>45</paymentNum>
  <signDate>20.03.2015</signDate>
  <cosua>0</cosua>
  <vysua>0</vysua>
  <mfsua>0</mfsua>
  <interestRate>54</interestRate>
  <defIntRate>0</defIntRate>
  <defIntPay>0</defIntPay>
  <round>10</round>
 </inputParams>
</document>
```

Vstupní soubor je tvořen kořenovým elementem `<document>` určujícím rozsah celého XML dokumentu. Tento kořenový element obsahuje párový subelement s názvem `<inputParams>`, který obsahuje následující potomky:

1. Text mezi párovou značkou `<name>` představuje název.
2. Text mezi párovou značkou `<loan>` představuje typ smlouvy (SS/SC).
3. Číslo mezi párovou značkou `<creditAmount>` vyjadřuje výši požadovaného úvěru.
4. Číslo mezi párovou značkou `<annuity>` vyjadřuje výši měsíční splátky.
5. Text mezi párovou značkou `<signDate>` definuje počáteční datum, kdy byl úvěr podepsán a od kterého se začne napočítávat prvotní splátka.
6. Čísla mezi párovými značkami `<cosua>`, `<vysua>`, `<mfsua>` představují jednotlivé fixní poplatky.
7. Číslo mezi párovou značkou `<interestRate>` představuje úrokovou míru s maximální přesností na tři desetinná místa.
8. Číslo mezi párovou značkou `<defIntRate>` představuje počet měsíců s odloženými úroky (DIR).
9. Číslo mezi párovou značkou `<defIntPay>` představuje počet měsíců s odloženými splátkami.
10. Poslední párová značka `<round>`, která se nachází uvnitř subelementu `<inputParams>`, definuje jednotkové zokrouhlení jednotlivých položek splátkového kalendáře zmíněné v podkapitole 4.2.

Subelement `<inputParams>` může být dle definovaného XSD souboru použit maximálně desetkrát, aby se zamezilo velké časové náročnosti při jeho parsování.

5.5.7 Transformace vstupních parametrů

XML vstupní parametry se převedou do vstupních parametrů pro simulátor, na jehož základě dojde k vygenerování splátkového kalendáře. Simulátor pošle stejné vstupy do SQL funkce, která vygeneruje splátkový kalendář z konkrétní databáze. Protože SQL funkce obsahuje některé rozdílné vstupní parametry, je zapotřebí provést transformaci mezi vstupními parametry simulátoru a SQL funkce.

Proměnná SQL funkce `p_date_start_interest`

Tato proměnná představuje datum, kdy se začíná započítávat úrok. Je tedy totožná s datem podpisu, kterou obsahuje simulátor. V případě, že se bude jednat o SS typ smlouvy, je zapotřebí proměnnou `p_date_start_interest` navýšit o jeden kalendářní den, viz podkapitola 4.4.1 *Výpočet úroků*.

Proměnná SQL funkce `p_annuity_date`

Tato proměnná představuje datum první splátky. Z předešlých poznatků je patrné, že se jedná o datum podpisu, které je navýšeno o 30 kalendářních dní, viz podkapitola 4.3 *Data měsíčních splátek*. V případě, že simulátor bude obsahovat vstupní parametr definující odložené splátky, bude dané datum navýšeno o počet měsíců s odloženými splátkami dle podkapitoly 4.5.

Proměnná SQL funkce `p_type_count`

V případě, že se bude počítat s odloženými úroky DIR, musí být hodnota proměnné nastavena na 3, jinak na 2.

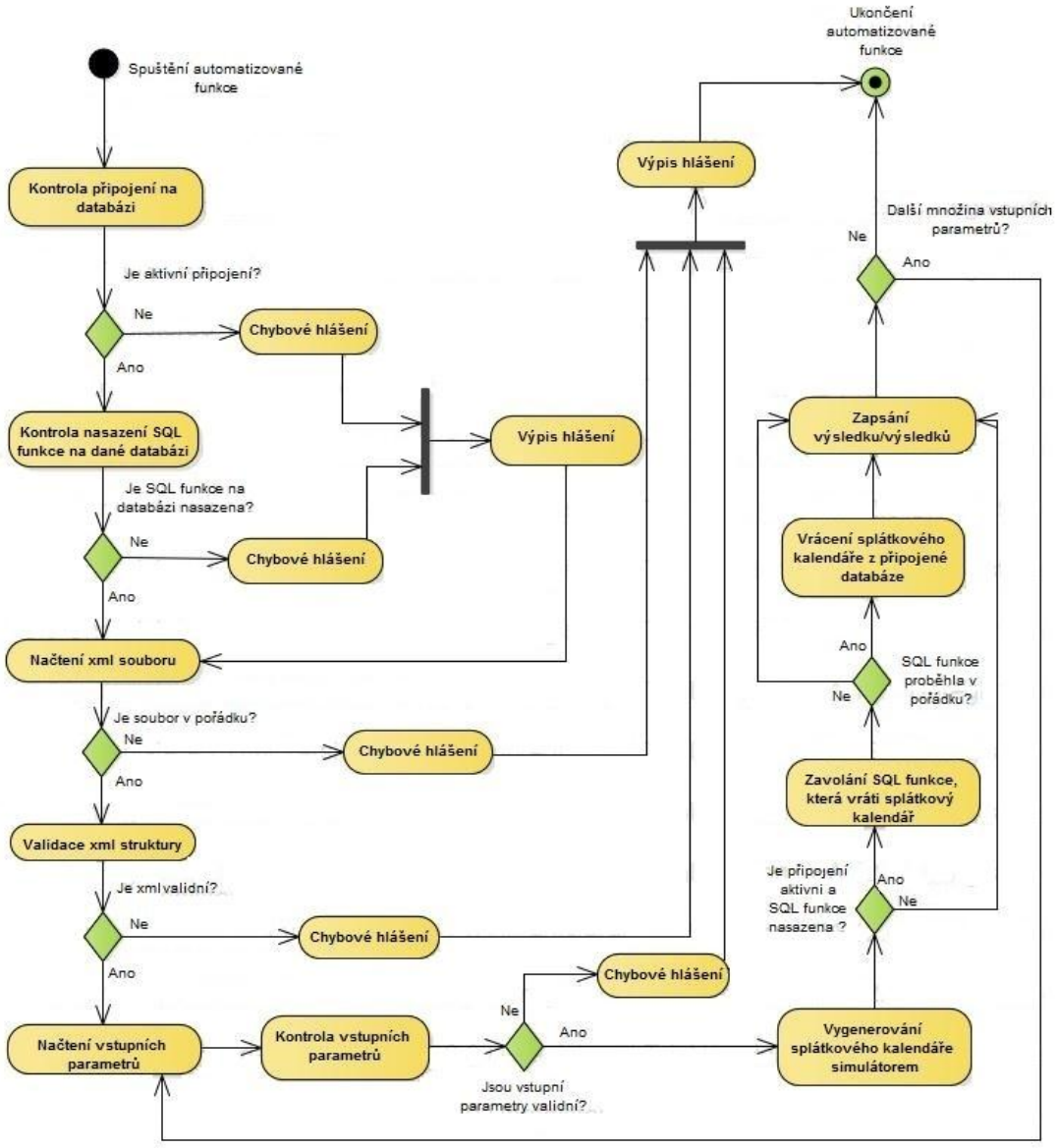
Proměnná SQL funkce `p_def_int_rate`

Představuje novou roční úrokovou míru. Tato proměnná je defaultně nastavena na hodnotu 0 a v případě, že se má vygenerovat splátkový kalendář s odloženými úroky (DIR), uloží simulátor do hodnoty proměnné novou vypočítanou úrokovou míru dle definice (22).

5.6 Diagram aktivit

Jedná se o vývojový diagram, jehož prostřednictvím lze zachytit aktivity, které mohou probíhat sekvenčně nebo paralelně. Diagram aktivit začíná černě označeným kruhem, který definuje počátek algoritmu a končí kruhem, v němž je menší soustředný vyplněný kruh. Mezi těmito symboly se nacházejí další ovály, které reprezentují jednotlivé kroky a do nichž se zapisuje konkrétní činnost (Virius, 2012, s. 15–16).

Diagram aktivit popisující proces simulace splátkového kalendáře je zobrazen v příloze. Na níže vloženém obrázku se nachází diagram aktivit popisující proces automatizované funkce navrženého simulátoru.



Obr. 10 Diagram aktivit – automatizovaná funkce

6 Implementace

Daná kapitola se bude zabývat implementací simulátoru, která proběhne v programovacím jazyce Java, přesněji na platformě Java SE 7. Pro implementaci je zvoleno vývojové prostředí Eclipse IDE. Pro práci s databází a komponentou DatePicker bude zapotřebí využít knihovny třetích stran, jelikož nejsou součástí Java Core API. Jedná se o následující knihovny:

JDBC

JDBC (Java DataBase Connectivity) představuje Java API, která definuje způsob, jakým klient přistupuje k databázi. Jedná se o množinu Java rozhraní, které umožňují programově komunikovat s různými typy databází. Tato komunikace je tvořena následujícími kroky:

1. Stanovení připojení k databázi.
2. Vykonávání SQL dotazů pro výběr, modifikaci nebo vytvoření databáze.
3. Ukončení připojení k databázi.

Pro úspěšné připojení je potřeba využít JDBC Driver Manager, který obsahuje veškeré typy JDBC ovladačů. JDBC ovladače jsou speciálně navrženy pro interakci s příslušnou DMBS (Database Management System), česky nazývaný systém řízení báze dat (Ganesh, Sharma, 2013, s. 281–282).

Připojení bude provedeno pro databázovou platformu Oracle s využitím externí knihovny `ojdbc14_g.jar`, která je volně dostupná internetové adrese: http://www.oracle.com/technetwork/apps-tech/jdbc-10201-088211.html (Oracle, 2009).

JCalendar

`JCalendar` je tzv. open-source komponenta umožňující vybírat datum z kalendáře. Obsahuje další subkomponenty, jako jsou: `JDateChooser`, `JDayChooser`, `JMonthChooser` a další. Výhodou těchto komponent je možnost přidání do Swing kontejneru, který po kliknutí zobrazí příslušný kalendář. Prvek vypadá jako klasické textové pole, které má vedle sebe ikonku kalendáře. Po kliknutí na danou ikonku dojde k vysunutí kalendáře, který se po výběru data zavře a výsledné datum se zobrazí do textového pole. Pro práci byla stažena externí knihovna `jcalendar-1.4.jar` dostupná na: http://toedter.com/jcalendar/ (Tödter, 2014).

6.1 Tvorba grafického uživatelského rozhraní

Při implementaci grafického uživatelského rozhraní došlo k převedení jeho návrhu, viz podkapitola 5.3. Aby bylo možné zobrazit jednotlivá GUI do záložek, bylo zapotřebí využít komponenty `JTabbedPane`:

```
tabbedPane=new JTabbedPane();
```

```
tabbedPane.addTab("Simulátor",contentPane);
tabbedPane.addTab("Automatizace",contentPane2);
```

Nejdříve byly vytvořeny dvě kontejnerové komponenty `contentPane` a `contentPane2` typu `JPanel`, které obsahují jednotlivé prvky popsané v návrhu. Poté byla vytvořena instance třídy `JTabbedPane`, do které jsou dané komponenty vloženy formou záložek.

6.2 Implementace simulátoru

Daná kapitola bude popisovat tvorbu simulátoru. Budou zde popsány nejdůležitější funkce, které tento simulátor obsahuje.

6.2.1 Výpočetní model

Při zobrazení splátkového kalendáře je třeba zohlednit, jaký výpočetní model bude pro jednotlivé splátky proveden.

```
public void pridejSplatku(int kolikata){
 if(…){
       pridejInstHoll(kolikata);
 }
  else if(…){
         pridejBezOdlSplatky(kolikata);
  }
   else if (…){
          pridejSOdlSplatky(kolikata);
          odlozenaSplatka=true;
   }
    else if(…){
           pridejSOdlozenimUroku(kolikata);
           odlozenyUrok=true;
    }
      else {
             pridejBezOdlSplatky(kolikata);
      }
}
```

Kód pro přehlednost neobsahuje vyplněné podmínky v jednotlivých `if` a `else` větvích. Pro každou splátku je dle konkrétní podmínky vybrán požadovaný výpočetní model.

6.2.2 Ukázka kódu pro volbu typu úročení

Každý výpočetní model pro jednotlivé splátky obsahuje metodu pro výpočet úroku. Jak bylo popsáno v analýze, je třeba stanovit, jaký typ úroku bude pro jednotlivé splátky počítán:

```
public double vypoctiUrok(String datum, double dluh){
 if (Vypocty.jePrestupny(datum)) Vypocty.dnyVRoce=366;
 else Vypocty.dnyVRoce=365;

 if(Vypocty.prestup==Vypocty.jePrestupny(datum)){
  if(Splatky.prvniSplatka)return prvotniUrok(dluh);
  else return standardniUrok(dluh);
 }
  else{
    Vypocty.prestup=!Vypocty.prestup;
    return slozenyUrok(datum,dluh)
 }
}
```

Nejdříve se v závislosti na datu splátky nastaví požadovaný počet dní v roce. Pokud nedošlo ke změně mezi přestupným a nepřestupným rokem a naopak, dojde k zavoláním privátních metod, které vracejí úroky v závislosti na aktuálním dluhu. V opačném případě se zavolá privátní metoda `slozenyUrok(datum,dluh)`, která vrací úrok v závislosti na aktuálním úroku a datu splátky.

6.2.3 Ukázka kódu pro automatizovaný výpočet anuity

Při automatizovaném dopočtu optimální anuity byl použit upravený vzorec dle definice (24). Při implementaci se však ukázalo, že i tento vzorec nevypočítá anuitu vždy adekvátně, viz podkapitola 5.1 *Softwarové požadavky*.

Jedná se o případy, kdy dochází k výpočtu úroků v závislosti na přestupném a nepřestupném roce dle podkapitoly 4.4.1 *Výpočet úroků*. Je potřeba zvolit způsob, jak efektivně dopočítat anuitu tak, aby poslední výše splátky byla stejná nebo co nejbližší ostatním splátkám:

```
public void mouseClicked(MouseEvent arg0) {
 nactiVstupy();
 if(Vypocty.lzeGenerovat()){
   Vypocty.nastavVlastniAnuitu();
   Vypocty.puleniIntervalu(Vypocty.datumPodpisu);
 }
 else {//chybové hlášení … }
 anuitaUrok_frmt.setValue(Vypocty.anuitaUrok);
 //vypis splatkoveho kalendare dle nastavene anuity …}
```

Při kliknutí na tlačítko „vypočtiAnuitu“ dojde k načtení vstupních parametrů z GUI. Poté dochází k ověření, jestli suma poplatků není příliš vysoká. Pokud tato kontrola proběhla v pořádku, dojde k nastavení předběžné anuity dle vzorce (24). Jelikož nastavená anuita nemusí být optimální, je zapotřebí zvolit iterativní způsob, díky kterému bude tato anuita dopočítána. Je zde zvolena metoda půlení intervalu, jejíž

prostřednictvím dojde k dopočítání optimální výše anuity. Jakmile je anuita vypočítána, je její hodnota zapsána do GUI, kde dojde k zobrazení splátkového kalendáře dle vypočítané anuity. Níže je popsán kód implementující metodu půlení intervalu:

```
public static Splatky puleniIntervalu(String datumPod){
 double val=Vypocty.anuitaUrok;
 Splatky kalendar=new Splatky();
 double hlAnuita=0;
 double poslAnuita=0;
 double rozdilAnuit=0;
 boolean contin=true;
 while(contin){
   val/=2;
   Vypocty.generujSplatkac(kalendar);
   hlAnuita=kalendar.splatky.get(a-2).getAnuita();
   poslAnuita=kalendar.splatky.getLast().getAnuita();
   rozdilAnuit1=hlAnuita-posledniAnuita;
   if(posledniAnuita<0)Vypocty.anuitaUrok-=val;
   if(posledniAnuita>hlAnuita)Vypocty.anuitaUrok+=val;
```

Nejdříve se inicializují základní proměnné a do hodnoty proměnné `val` se uloží nastavená výše anuity, která je vyjádřena procentuálně na tři desetinná místa. Tato proměnná se v každé iteraci snižuje o polovinu (`val/=2`). Poté se do instance třídy `Splatky` nahraje pole jednotlivých splátek v závislosti na nastavené výši anuity. V případě, že je poslední výše splátky záporná, to znamená, že výše jednotlivých splátek je příliš vysoká, proto musí dojít ke snížení (`anuitaUrok-=val`). V opačném případě dojde k navýšení (`anuitaUrok+=val`).

```
   if(poslAnuita<=hlAnuita && poslAnuita>0){
    val=2*val;
    double pomVal=val/10;
    Vypocty.anuitaUrok-=pomVal;
    double rozdilAnuit=pocitej(calendar,datumPod);
    if(!(rozdilAnuit>0 && rozdilAnuit<rozdilAnuit1)){
       Vypocty.anuitaUrok+=pomVal;
        Vypocty.anuitaUrok+=pomVal;
        rozdilAnuit= pocitej(calendar,datumPod);
        if(!(rozdilAnuit>0&& rozdilAnuit<rozdilAnuit1)){
             Vypocty.anuitaUrok-=pomVal;
             contin=false;
        }
    }
   }
  } return kalendar;}
```

Poslední část kódu je tvořena podmínkou, která ověřuje, jestli je poslední splátka nezáporná a současně nižší nebo stejná jako ostatní konstantní anuity, tedy je-li v přijatelném rozmezí.

Pro dosažení cíle, aby výše poslední splátky byla stejná nebo co možná nejbližší ostatním splátkám, je třeba ověřit, jestli se pomocí navýšení/snížení výše anuity, nedostane k lepšímu řešení. Pokud je algoritmus v dané větvi, je zapotřebí prozkoumat jeho nejbližší okolí oběma směry. Proto krok určující míru navýšení je ponížen o desetinu (`double pomVal=val/10`). Pro první směr je anuita poníženа o danou hodnotu (`Vypocty.anuitaUrok-=pomVal`) a pomocí metody `pocitej(kalendar,datumPod)` dojde k získání nového rozdílu mezi poslední a konstantní splátkou. V případě, že daný směr nenašel lepší řešení, algoritmus vybere opačný směr a provede navýšení `Vypocty.anuitaUrok+=pomVal`.

Pokud ani tento směr nenašel lepší řešení, znamená to, že původní výše anuity, se kterou se algoritmus dostal do dané větve, je nejvhodnější a algoritmus se ukončí. V opačném případě jeden ze zvolených směrů poskytuje lepší řešení a algoritmus se opakuje.

V ojedinělých případech, v závislosti na kombinaci vstupních parametrů, může nastat situace, kdy časová složitost algoritmu může být pro uživatele zdlouhavá. Z tohoto důvodu je algoritmus omezen maximálním počtem iterací. Zvolená hodnota maximálního počtu opakování je 100.

6.2.4 Validace vstupních parametrů

Validace jednotlivých vstupních parametrů probíhá dle regulárních výrazů. Níže jsou zobrazeny dva vybrané regulární výrazy pro procenta a datum.

```
public static String procenta=// 0,001-100||0.001-100
"^100$|^0$|^100\\.[0]{1,3}$|^100\\,[0]{1,3}$|
^[1-9]{1}[0-9]{1}$|^[1-9]{1}[0-9]{0,1}\\,
[0-9]{1,3}$|^[1-9]{1}[0-9]{0,1}\\.[0-9]{1,3}|^0\\,
[0-9]{1,3}$|^0[0-9]{0,1}\\.[0-9]{1,3}$|^[0-9]{1}$";

public static String datum= //dd.MM.yyyy||d.M.yyyy
"^(0?[1-9]|[12][0-9]|3[01]).(0?[1-9]|1[012]).
((19|20)\\d\\d)$";
```

Jakmile je dle regulárních výrazů vyjádřen vzor pro jednotlivé vstupní parametry, lze jednoduše vytvořit funkci, která porovná, jestli zadaný řetězec odpovídá vzoru:

```
public static boolean validniProcento(String s){
     return(s.matches(procenta));
  }
 public static boolean validniDatum(String s){
     return(s.matches(datum));
  }
```

6.3 Implementace automatizované funkce

Daná kapitola se bude zabývat tvorbou automatizované funkce a popisem nejdůležitějších dílčích částí. V příloze Obr. 13 je ukázka výstupního HTML souboru.

6.3.1 Ukázka validace XML struktury

Pro kompletní práci s XML soubory bylo vybrané aplikační rozhraní JAXP (Java API for XML Processing). Toto rozhraní je od verze Java 1.5 součástí Java Core API, není tedy zapotřebí stahovat knihovny třetích stran.

Pro validaci XML struktury byl využit balík `javax.xml.validation`. Jedná se o API umožňující provést ověření, zdali je daný XML dokument instancí daného XML schématu (Juneau, 2014, s. 546–547).

```
public static boolean validujXMLZnacky(){
 try{
   Source xmlSource = new StreamSource(xmlFile);
   SchemaFactory factory = SchemaFactory
       .newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
   Schema schema = factory.newSchema(new StreamSouce
   (Xml.class.getResourceAsStream("validator.xsd")));
   Validator validator = schema.newValidator();
    validator.validate(xmlSource);
   return true;
 }
 catch (SAXException e) {
      return false;
 }
}
```

Nejprve dojde k vytvoření proudového zdroje ze zadaného XML souboru. Pro zjednodušení uvažujeme, že zadaný soubor opravdu existuje a není prázdný. Dále je zapotřebí připravit instanci tovární třídy `SchemaFactory` pro vytváření `Schema` objektů. Jedná se o kompilátor schémat, kterému je zapotřebí definovat jazyk schématu. Instance třídy `Schema` reprezentuje konkrétní schéma vytvořené ze souboru s názvem validator.xsd, který je součástí programu. Předposledním krokem je vytvoření instance třídy `Validator`, které se předhodí vzorové schéma. Jedná se o procesor, který porovnává XML dokument se zadaným schématem. V případě, že při validaci dojde k vyhození výjimky, je XML soubor vzhledem ke schématu nevalidní.

6.3.2 Ukázka parsování XML dokumentu

Pro analýzu XML dokumentu byl využit balík `javax.xml.parsers`. Níže je zobrazena zjednodušená ukázka kódu, který parsuje XML dokument a načítá jeho vstupní parametry.

```
DocumentBuilder dBuilder=null;
DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
dBuilder = dbFactory.newDocumentBuilder();
Document doc = dBuilder.parse(xmlFile);
doc.getDocumentElement().normalize();
// nactu všechny znacky s nazvem Inputparams
NodeList nList =
doc.getElementsByTagName("inputParams");
// pro kazdy tag projizdim jeho sub elementy
for (int temp = 0; temp < nList.getLength(); temp++) {
    Node nNode = nList.item(temp);
    if (nNode.getNodeType() == Node.ELEMENT_NODE) {
    //nastavim pocatecni hodnoty
        Element eElement = (Element) nNode;
          //ulozeni do promene typu int
       int odlUroky=Integer.valueOf
           (eElement.getElementsByTagName("defIntRate")
           .item(0).getTextContent());
   …}
…}
```

Třída `DocumentBuilder` slouží pro převedení řetězcové reprezentace XML dokumentu na DOM reprezentaci. Tovární třída `DocumentBuilderFactory` umožňuje vytvářet parsery (instance třídy `DocumentBuilder`). Instance třídy `Document` představuje kompletní XML dokument poskytující primární přístup k daným datům.

Instance třídy `NodeList` reprezentuje uspořádanou kolekci uzlů (`Node`) získaných z požadovaného dokumentu. Pro každý uzel uložený v dané kolekci dochází k prohledávání jeho subelementů. Jakmile daný subelement odpovídá požadovanému názvu, je uložen do konkrétní proměnné. Celý kód by měl obsahovat bloky `try-catch-finally`, které jsou pro přehlednost v ukázce vynechány.

6.3.3 Ukázka kódu připojeni k DBS

Základním krokem pro provedení automatizované funkce je připojení se ke konkrétní databázi. Před samotným připojením je zapotřebí zaregistrovat konkrétní ovladač, díky kterému bude možné inicializovat připojení.

```
private void registerJDBC(){
 try {
       DriverManager.registerDriver
      (new oracle.jdbc.driver.OracleDriver());
 }
 catch (SQLException ex) {
```

```
        ex.printStackTrace();
        System.err.println("JDBC not registered!");
        return;
    }
    System.out.println("Oracle JDBC Driver Registered!");
  }
```

V případě, že došlo k úspěšnému zaregistrování ovladače, je možné zažádat o připojení ke konkrétní databázi:

```
public void conectDBS()  {
   registerJDBC();
   try {
      connection =
      DriverManager.getConnection(url,user,pswd);
       try{
        callProcedure();
       }
       catch(ParseException ex){
         nasazenaProcka=false;
       }
    }
     catch (SQLException e) {
       System.out.println("Unable to connect!");
       e.printStackTrace();
       return;
     }
      finally{
        if (connection!=null) {
         try {
              connection.close();
             }
          catch (SQLException e) {}
        }
      }
   }
```

Nejdříve se zavolá soukromá metoda `registerJDBC()`, která zaregistruje ovladač. Poté dojde k poslání požadavku o připojení k databázi. Pokud došlo k úspěšnému připojení, zavolá se metoda `callProcedure()`, která vykoná SQL funkci (viz podkapitola 5.5.1), jejíž výsledek se nahraje do pole.

V případě, že na dané databázi není nasazen požadovaný balík s funkcí, dojde k vyhození výjimky. Pokud bylo navázané připojení k databázi, dojde vždy k uzavření tohoto připojení, které je ošetřeno blokem `finally`.

7 Diskuse

Naimplementovaný simulátor splátkového kalendáře obsahuje zcela novou funkčnost popisovanou v podkapitole 4.9, která nebyla firmou ještě naimplementována. Z toho důvodu lze očekávat, že tento funkční model může být pozměněn, popř. se může ukázat v rámci testování, že simulátor tuto funkčnost počítá nesprávně.

Nedostatkem práce je, že výsledná automatizovaná funkce prozatím nepodporuje veškeré vstupní parametry. Aby tato automatizace byla firmou plně využitelná, je zapotřebí, aby se firemní SQL funkce rozšířila o nové parametry definující tyto typy výpočtů. Bez dané funkce nemůže dojít k porovnávání výsledků, a proto nebyly veškeré vstupní parametry do automatizované funkce zahrnuty.

Protože firemní SQL funkce prozatím neobsahuje parametr, který definuje jednotku zaokrouhlení, je potřeba před spuštěním automatizované funkce na konkrétní databázi ověřit, jakou jednotku zaokrouhlení má nastavenou. Tuto jednotku pak tester musí zadat do vstupního XML souboru.

Posledním nedostatkem automatizované funkce je postrádání parametru definující procentuální poplatek, protože v dané SQL funkci není prozatím zohledněn.

Díky uplatnění návrhových vzorů je výsledný program uživatelsky přívětivý. Podařilo se naimplementovat veškeré stávající i nové funkční požadavky. Výsledná automatizovaná funkce podporuje kontrolu stávající funkcionality na všech firemních databázích. Tato funkce byla navržena tak, aby firma nemusela vynakládat svoje finanční rezervy na její neustálou kontrolu a údržbu.

7.1 Možnosti dalšího pokračování

Výsledný program nabízí několik rozšíření. Jedná se především o přidání nových parametrů do vstupního XML souboru, díky kterým bude možno automatizovat veškeré funkce, které prozatím nebyly do automatizace zahrnuty. Jedná se o tyto funkce: zaokrouhlování, installment holidays, restrukturalizace a nová funkčnost výpočtu SS a SC smluv popisované v kapitole 4 *Analýza současného stavu*.

8 Závěr

Cílem diplomové práce bylo vytvoření simulátoru splátkového kalendáře s funkcí automatizovaného testování jeho vstupních parametrů. Aplikace slouží testerům a analytikům firmy EmbedIT pro běloruský bankovní sektor.

Při zpracování diplomové práce byl kladen důraz na využití co nejaktuálnější literatury. Aktuální zahraniční literatura byla použita především v kapitolách zabývajících se automatizací a testováním. Při vytvoření této aplikace bylo zapotřebí nastudovat interní firemní dokumentace popisující nové i stávající funkce, které výsledný simulátor poskytuje.

Praktickým přínosem bylo zaplnění dosavadních nedostatků popsaných v kapitole 4 *Analýza současného stavu*. Díky této aplikaci si testeři mohou ověřovat správné nastavení vstupních parametrů pro anuitní typy úvěrů a dochází tak k zamezení chybovosti při zakládání smluv vlivem nesprávně zadaných parametrů. Program poskytuje automatizovaný dopočet výše anuity v závislosti na vstupních parametrech, díky čemuž došlo ke zkrácení vedlejších časů testera.

Dalším přínosem práce bylo vytvoření automatizované funkce, jejímž prostřednictvím mohou testeři rychle kontrolovat správnost generování splátkových kalendářů na jednotlivých databázích při jejich nasazování prostřednictvím výsledného simulátoru.

Firmě byla předvedena ukázka této automatizované funkce, která byla shledána velice užitečnou, a bylo rozhodnuto o jejím rozšíření tak, aby zahrnovala veškeré vstupní parametry. Toto rozšíření však bude proveditelné nejdříve, jakmile bude nová funkčnost, kterou poskytuje vytvořený simulátor firmou, naimplementována.

Po prvotní ukázce výstupního HTML souboru automatizované funkce bylo dle přání firmy zapotřebí přidat JavaScriptové funkce, díky kterým je možné jednotlivé tabulky schovávat. Posledním požadavkem bylo, aby výsledný HTML soubor byl samostatně plně funkční, proto nevyužívá žádné externí knihovny a CSS je definováno interně.

Realizovaný systém je v současné době naimplementován jako tzv. beta verze a probíhá jeho testování firmou EmbedIT. Bylo provedeno školení testerů, na kterém bylo názorně ukázáno, jak danou automatizovanou funkci správně používat včetně tvorby XML vstupního souboru. Na základě připomínek testerů dojde k upravení programu a budou odstraněny vzniklé nedostatky z implementace. Jakmile budou tyto nedostatky odstraněny, bude daný simulátor používán jako kontrolní nástroj při provádění smoke a regresních testů.

9 Literatura

ACHARYA, Sujoy. *Mastering Unit Testing Using Mockito and JUnit*. Birmingham: Pack Publishing, 2014, 314 s. ISBN 978-1-78398-250-9.

AMMANN, Paul a Jeff OFFUTT. *Introduction to Software Testing*. New York: Cambridge University Press, 2008, 346 s. ISBN 978-0-521-88038-1.

BORBA, Paulo et al. *Testing Techniques in Software Engineering: Second Pernambuco Summer School on Software Engineering*. Germany: Springer Science & Business Media, 2010, 313 s. ISBN 978-3-642-14334-2.

BURNS, David. *Selenium 2 Testing Tools: Beginner's Guide*. Birmingham: Pack Publishing, 2012, 437 s. ISBN 978-1-84951-830-7.

DIETRICH, Erik. *Starting to Unit Test: Not as Hard as You Think*. United Kingdom: Price World Publishing, 2014, 80 s. ISBN 978-1-61984-995-2.

DIRGA, David. EMBEDIT. *IABY-283: Rounding the monthly payment*. version 0.1. Brno, 2014, 9 s.

DIRGA, David. EMBEDIT. *IABY-299: Financial protection.* version 0.15. Brno, 2015, 124 s.

DOWNING, Steven et al. *Handbook of Test Development*. New Jersey: Routledge, 2011, 792 s. ISBN 978-1-135-28338-4.

GANESH, Samarthyan a Tushar SHARMA. *Oracle certified professional Java SE 7 programmer exams 1Z0-804 and 1Z0-805 a comprehensive OCPJP 7 certification guide*. New York: Apress, 2013, 656 s. ISBN 978-1-4302-4765-4.

IPSER, Jaroslav. EMBEDIT. *IABY-286: 3rd product for 5th Element: version 0.9*. Brno, 2012, 44 s.

JUNEAU, Josh. *Java 8 Recipes: apply proven sollutions to speed up your java 8 development*. 2nd ed. New York: Apress, 2014. ISBN 978-1-4302-6826-0.

LIMAYE, Milind G. *Software Testing*. New Delhi: Tata McGraw-Hill Education, 2009, 523 s. ISBN 978-0-070-13990-9.

MCCAFFREY, James. *NET Test Automation Recipes: A Problem-Solution Approach*. New York: Apress, 2012, 404 s. ISBN 978-1-430-25077-7.

MLÝNKOVÁ, Irena a Jaroslav POKORNÝ. *XML technologie: principy a aplikace v praxi*. 1. vyd. Praha: Grada, 2008, 267 s. Průvodce (Grada). ISBN 9788024727257.

ORACLE. *Oracle JDBC Drivers release: Oracle Database 10g Release 2 JDBC Drivers* [online]. 2009 [cit. 2015-05-05]. Dostupné z: http://www.oracle.com/technetwork/apps-tech/jdbc-10201-088211.html

PÍSEK, Slavoj. *HTML: začínáme programovat*. 4., aktualiz. vyd. Praha: Grada, 2014, 181 s. Průvodce (Grada). ISBN 978-80-247-5059-0.

RADOVÁ, Jarmila, Petr DVOŘÁK a Jiří MÁLEK. *Finanční matematika pro každého*. Praha: Grada Publishing, 2013, 304 s. ISBN 978-8-024-74831-3.

SALOMON, David. *Data compression: The complete reference*. 4th ed. California: Springer, 2007, xxii, 1359 s. ISBN 978-1-84628-603-2.

SAMEK, Jan. EMBEDIT. *IABY-354: Payment date.* version 0.3. Brno, 2015, 46 s.

SELENIUM WebDriver. SeleniumHQ [online]. 2014 [cit. 2015-02-08]. Dostupné z: http://www.seleniumhq.org/projects/webdriver/

SEVERIN, Peter. *WireframeSketcher: Wireframing Tool for Professionals* [online]. 2008 [cit. 2015-03-13]. Dostupné z: http://wireframesketcher.com/

SIRIWARDENA, Prabath. *Advanced API Security: Securing APIs with OAuth 2.0, OpenID Connect, JWS, and JWE*. New York: Apress, 2014, 260 s. ISBN 978-1-430-26817-8.

SMARTBEAR SOFTWARE. SoapUI The Swiss-Army Knife of Testing. SoapUI [online]. 2014 [cit. 2015-02-08]. Dostupné z: http://www.soapui.org/about-soapui/what-is-soapui-.html

ŠOBA, Oldřich, Martin ŠIRŮČEK a Roman PTÁČEK. *Finanční matematika v praxi.* Praha: Grada Publishing, 2013, 304 s. ISBN 978-8-024-78424-3.

TIDWELL, Jenifer. *Designing interfaces*. 2nd ed. Sebastopol: O'Reilly, c2011, xxv, 547 s. ISBN 978-1-449-37970-4.

TÖDTER, Kai. JCalendar. Toedter.com [online]. 2014 [cit. 2015-04-09]. Dostupné z: http://toedter.com/jcalendar/

VIRIUS, Miroslav. *Pascal: programování pro začátečníky*. 1. vyd. Praha: Grada, 2012, 253 s. Průvodce (Grada). ISBN 978-80-247-4116-1.

VYMĚTAL, Dominik. *Informační systémy v podnicích: teorie a praxe projektování*. Praha: Grada Publishing, 2009, 144 s. ISBN 978-8-024-76280-7.

WEINSTOCK-HERMAN, Eli. 2011. *Less Than Dot: A Technical Community for IT Professionals* [online]. [cit. 2015-04-11]. Dostupné z: http://blogs.lessthandot.com/index.php/webdev/uidevelopment/automated-web-testing-with-selenium/

WIEGERS, Karl Eugene a Joy BEATTY *Software requirements*. 3rd ed. Redmond, WA: Microsoft press, c2013, xxxii, 637 s. Best practices. ISBN 978-0-7356-7966-5

YANG, Herong. 2009. *What Is soapUI 3.0.1?: WSDL Tutorials - Herong's Tutorial Examples* [online]. [cit. 2015-04-11]. Dostupné z: http://www.herongyang.com/WSDL/soapUI-301-What-Is-soapUI.html

Seznam obrázků

Seznam tabulek

Přílohy

A Přiložené CD

Přiložené CD obsahuje následující položky:

1. Diplomovou práci ve formátu PDF.
2. Zdrojové kódy v adresáři src v programovacím jazyce Java.
3. Externí knihovny v adresáři libs.
4. XML soubory produktyTest1.xml. a produktyTest2.xml.
5. Soubor readme.txt.
6. Soubor license.txt.
7. Programovou dokumentace v adresáři dokumentace.
8. Spustitelné soubory Splatkac.jar a Splatkac.exe v adresáři dist.

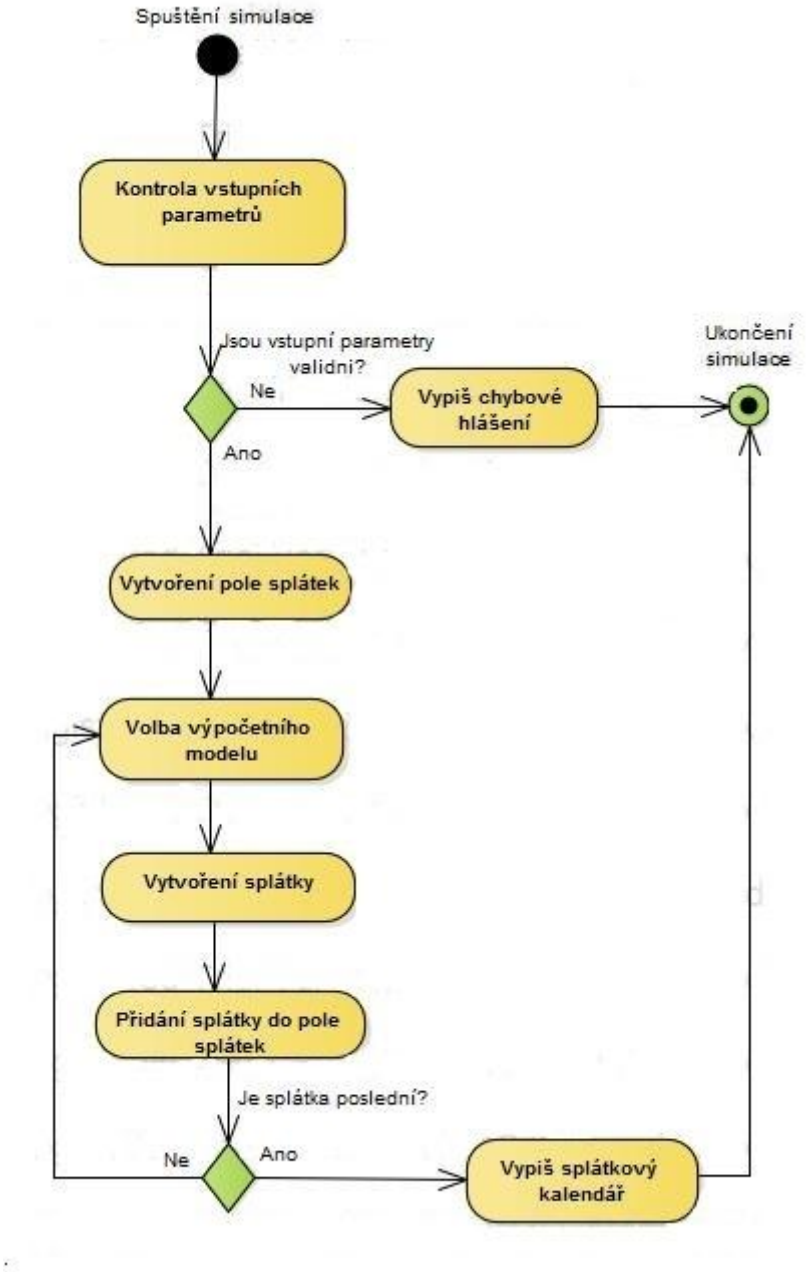
B Návrh XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:element name="document">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="inputParams" maxOccurs="10" >
     <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string" nillable="true"></xs:element>
        <xs:element name="loan" type="xs:string" nillable="true"></xs:element>
        <xs:element name="creditAmount" type="xs:int"></xs:element>
        <xs:element name="annuity" type="xs:int"></xs:element>
        <xs:element name="paymentNum" type="xs:int"></xs:element>
        <xs:element name="signDate" type="xs:string"></xs:element>
        <xs:element name="cosua" type="xs:int"></xs:element>
        <xs:element name="vysua" type="xs:int"></xs:element>
        <xs:element name="mfsua" type="xs:int"></xs:element>
        <xs:element name="interestRate" type="xs:decimal"></xs:element>
        <xs:element name="defIntRate" type="xs:int"></xs:element>
        <xs:element name="defIntPay" type="xs:int"></xs:element>
        <xs:element name="round" type="xs:decimal"></xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>

```

Obr. 11 Návrh XSD

C Diagram aktivit



Obr. 12 Diagram aktivit – simulace splátkového kalendáře

D Výstup automatizované funkce

SS DP

Input params

SC DP

Input params

SS classic short

Input params

Simulator output						
ID	**Date**	**Anuity**	**Interest**	**Amortization**	**Fixed fee**	**Balance**
1	02.06.2015	956 050	181 890	764 160	10 000	4 235 840
2	02.07.2015	956 050	159 500	786 550	10 000	3 449 290
3	01.08.2015	956 050	129 880	816 170	10 000	2 633 120
4	31.08.2015	956 050	99 150	846 900	10 000	1 786 220
5	30.09.2015	956 050	67 260	878 790	10 000	907 430
6	30.10.2015	951 600	34 170	907 430	10 000	0
	Sum	**5 731 850**	**671 850**	**5 000 000**	**60 000**	

Procedure output						
ID	**Date**	**Anuita**	**Interest**	**Amortization**	**Fixed fee**	**Balance**
1	02.06.2015	956 050	181 890	764 160	10 000	4 235 840
2	02.07.2015	956 050	159 500	786 550	10 000	3 449 290
3	01.08.2015	956 050	129 880	816 170	10 000	2 633 120
4	31.08.2015	956 050	99 150	846 900	10 000	1 786 220
5	30.09.2015	956 050	67 260	878 790	10 000	907 430
6	30.10.2015	951 600	34 170	907 430	10 000	0
	Sum	**5 731 850**	**671 850**	**5 000 000**	**60 000**	

SS DIR short

Input params

Simulator output						
ID	**Date**	**Anuity**	**Interest**	**Amortization**	**Fixed fee**	**Balance**
1	02.06.2015	1 394 350	0	1 369 350	25 000	3 630 650
2	02.07.2015	1 394 350	177 120	1 192 230	25 000	2 438 420
3	01.08.2015	1 394 350	123 160	1 246 190	25 000	1 192 230
4	31.08.2015	1 277 450	60 220	1 192 230	25 000	0
	Sum	**5 460 500**	**360 500**	**5 000 000**	**100 000**	

Procedure output						
ID	**Date**	**Anuita**	**Interest**	**Amortization**	**Fixed fee**	**Balance**
1	02.06.2015	1 394 350	0	1 369 350	25 000	3 630 650
2	02.07.2015	1 394 350	183 380	1 185 970	25 000	2 444 680
3	01.08.2015	1 394 350	123 480	1 245 870	25 000	1 198 810
4	31.08.2015	1 284 360	60 550	1 198 810	25 000	0
	Sum	**5 467 410**	**367 410**	**5 000 000**	**100 000**	

Obr. 13 Ukázka výstupního HTML souboru automatizované funkce