VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

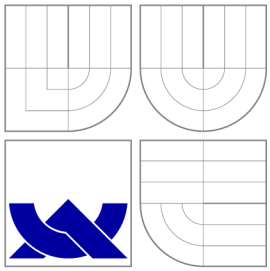
ROZŠÍŘENÍ PLÁNOVAČE TESTŮ PRO DISTRIBUOVANÉ SYSTÉMY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

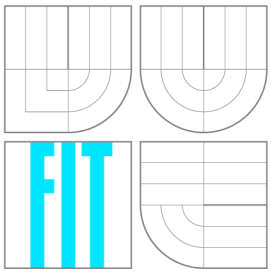
AUTOR PRÁCE
AUTHOR

FILIP MÉSZÁROS

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ROZŠÍŘENÍ PLÁNOVAČE TESTŮ PRO DISTRIBUOVANÉ SYSTÉMY

TEST PLANNING TOOL EXTENSION FOR DISTRIBUTED SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

FILIP MÉSZÁROS

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. FRANTIŠEK ŠČUGLÍK, Ph.D.

BRNO 2014

Abstrakt

Tato bakalářská práce se zabývá automatickým testováním softwaru s použitím plánovače testů. Popisuje tvorbu rozšíření existujícího plánovače testů tak, aby bylo možné efektivně rozdělit skupinu testů na části, které se budou vykonávat nezávisle na sobě. Jednotlivé testy jsou rozdělovány na základě společných vlastností prostředí, které je pro tyto testy nutno připravit, a na základě závislostí mezi testy. Práce dále popisuje, jaké optimalizace jsou použity pro rozdělování testů do podmnožin. Každá podmnožina testů je spouštěná na samostatném testovacím systému a tím je snížen čas potřebný pro otestování testovaného softvéru danou sadou testů. Vytvořený nástroj se úspěšně používá při každodenním testování několika produktů ve firmě Acision, pro kterou byl tento nástroj vytvářen.

Abstract

This bachelor thesis is about automatical software testing using the testing scheduler. It describes creation of the extension for the existing testing scheduler, so it will be possible to split effectively a group of tests to segments, that will be executed independently on each other. Tests are splitted according to the common characteristics of the enviroment, that need to be prepared for each test, and according to the dependencies between the tests. Furthermore, it describes what optimizations are used for splitting of the tests to subsets. Each subset of the tests runs on a standalone testing system, so the time needed for succesful completion of testing with the given set of tests is reduced. Created tool is succesfully used during everyday testing of the several products in the Acision company, to which was this tool made.

Klíčová slova

automatické testování softvéru, distribuce testů, paralelizace, plánovač, vyhodnocování výsledků, distribuce zátěže, verifikace, validace, regresní testování

Keywords

automatical software testing, test distribution, parallelization, scheduler, evaluation of results, load distribution, verification, validation, regression testing

Citace

Filip Mészáros: Rozšíření plánovače testů pro
distribuované systémy, bakalářská práce, Brno, FIT VUT v Brně, 2014

Rozšíření plánovače testů pro distribuované systémy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Františka Ščuglíka, Ph.D. a pod technickým vedením pana Miroslava Bureša.

.....
Filip Mészáros
19. května 2014

Poděkování

V první řadě bych chtěl poděkovat své rodině a přátelům, za podporu při studiu. Dále bych chtěl poděkovat svému odbornému konzultantovi Miroslavu Burešovi, za jeho rady, které mi pomohly k úspěšnému vytvoření této práce. Velké poděkování patří také mému vedoucímu práce, Ing. Františkovi Ščuglíkovi, Ph.D., a mým kolegům z práce, za jejich cenné rady a připomínky. Velice rád bych také poděkoval firmě Acision, která mi umožnila tuto práci vypracovávat.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Kapitola 1

Úvod

Testovanie softvéru je jednou z dôležitých etáp pri jeho vyvíjaní. Pred vydávaním softvéru a jeho predávaním zákazníkovi sa na výsledky testovania berie obrovský ohľad. Preto sa softvér často testuje veľkou sadou testov, ktorá má za úlohu odhaliť jeho možné chyby.

Vo firme Acision sa testujú jednotlivé produkty pomocou regresného testovania, ktoré často pozostáva z tisícok testov a trvá až niekoľko hodín. Toto testovanie je riadené nástrojom, ktorý je schopný vybrať z testovacej sady testy na základe niekoľkých parametrov, tieto testy naplánovať a spustiť ich. Naplánovanie testov je nutné z dôvodu, že jednotlivé testy totiž nie je možné spúšťať v náhodnom poradí. Je to okrem iného aj z dôvodu, že testy majú medzi sebou závislosti, ktoré je nutné dodržiavať.

Problémom tohto nástroja je to, že je schopný spúšťať testy len na jednom testovacom systéme. Postupom času sa však toto regresné testovanie stalo časovo náročným a bolo potrebné nájsť riešenie pre zníženie časovej náročnosti tohto testovania. Riešením tohto problému sa venuje táto diplomová práca.

Táto diplomová práca sa zaoberá vytvorením rozšírenia pre existujúci nástroj na plánovanie testov, aby bolo možné testy distribuovať na niekoľko testovacích systémov. Takouto formou sa zabezpečí to, že sa časová náročnosť regresného testovania zníži. Vďaka tejto diplomovej práci je teraz možné testy rozdistribuovať na teoreticky ľubovoľný počet testovacích systémov. V prípade kritickej časovej tiesne je teda možné pozbierať všetky voľné testovacie systémy, a skrátiť tak čas potrebný na otestovanie systému na minimum.

Diplomová práca je rozdelená na 7 kapitol. Kapitola 2 obsahuje popis základných princípov testovania softvéru a možnosti znižovania cien regresného testovania. Ďalej kapitola popisuje, ako sa softvér testuje v praxi vo firme Acision a popisuje princíp využívaného nástroja pre plánovanie a spúšťanie testov. V kapitole 3 si rozoberieme konceptuálny návrh vytváraného rozšírenia pre plánovač testov. Uvedieme si návrh riešenia nových problémov, ktoré vznikli pri riešení distribúcie testov na viaceré testovacie systémy. Kapitola 4 popisuje implementáciu jednotlivých častí vytváraného rozšírenia nástroja pre plánovanie a spúšťanie testov. V kapitole 5 si popíšeme naimplementované optimalizácie algoritmu na distribúciu testov. V predposlednej kapitole 6 si zhodnotíme výsledky, ktoré sa vytvorením tejto práce podarilo dosiahnuť. Nakoniec je v kapitole 7 uvedené celkové zhrnutie obsahu a takisto aj námety na ďalší vývoj tejto práce.

Kapitola 2

Testovanie softvéru

V nasledujúcej kapitole sú stručne zhrnuté teoretické vedomosti o testovaní softvéru. Testovanie softvéru je v dnešnej dobe jedným z kľúčových faktorov v IT sfére a patrí medzi základnú etapu modelu životného cyklu softvéru. Podľa štandardu IEEE 1059 [3] je testovanie proces analýzy softvéru a detekcie rozdielností medzi existujúcimi a požadovanými podmienkami a slúži na vyhodnocovanie vlastností testovaného softvéru.

Kapitola popisuje, prečo je testovanie softvéru dôležité a aké rôzne typy testovania softvéru sa dnes používajú. Vysvetlíme si základné pojmy súvisiace s testovaním softvéru a pozrieme sa na to, ako sa testuje softvér v praxi. Popíšeme si, ako sa automaticky testuje softvér vo firme Acision a aké nástroje táto firma využíva. V kapitole sa budeme venovať regresnému testovaniu a spôsobmi znižovania cien tohoto typu testovania.

Na záver kapitoly si popíšeme plánovač testov, ktorý firma Acision každodenne používa pri testovaní rôznych typov softvéru, ktorý firma vyvíja. Cieľom tejto bakalárskej práce je úprava tohto plánovača tak, aby podporoval distribúciu testov na viaceré testovacie systémy, a tým skrátil čas potrebný pre otestovanie softvéru danou sadou testov.

2.1 Typy testovania softvéru

Dokument [5] definuje pojem testovania softvéru ako *dynamickú* verifikáciu správania programu voči *očakávanému* správaniu programu na *konečnej* vzorke testov, vhodne *zvolenej* zo zvyčajne nekonečného množstva možných prípadov použitia. Roger S. Pressman, jeden z medzinárodne uznávaných odborníkov na zlepšovanie procesov softvérového inžinierstva v dokumente [9] prehlásil, že testovanie softvéru je kritickým prvkom zabezpečenia kvality softvéru, ktorý má jedinú primárnu úlohu, a to nájsť v ňom chyby.

Pred samotným preberaním rôznych typov testovania si najprv vysvetlíme dva základné pojmy, *verifikáciu* a *validáciu*. V softvérovom inžinierstve je verifikácia a validácia proces kontroly, či softvér spĺňa špecifikácie a či napĺňa požadovaný účel.

- **verifikácia** – je overovanie správnosti produktu vzhľadom k formulovaným požiadavkám. Verifikácia overuje, že softvér vyhovuje špecifikácii.
- **validácia** – je overenie správnosti produktu vzhľadom k reálnym požiadavkám. Validácia overuje, že softvér splňuje potreby zákazníka.

Pre verifikovanie, že daný softvér neobsahuje chybu, ho musíme otestovať pre každú možnú kombináciu vstupných hodnôt. Tento prístup je však vďaka komplexite softvéru často nemožný, pretože je množina všetkých vstupných hodnôt typicky nekonečne veľká.

Toto však znamená, že testovanie softvéru nám s takýmto prístupom môže poukázať na prítomnosť chýb, no nemôže nám dokázať, že v softvéri chyby nie sú.

Existujú rôzne spôsoby delenia testovania softvéru. Jedno z hlavných delení je delenie podľa spôsobu testovania. Toto delenie vychádza z toho, či je potrebné k prevedeniu testu daný softvér spustiť, alebo nie.

- **Statické testovanie** – je testovanie, ktoré nevyžaduje beh softvéru. Statická analýza sa snaží odhaliť niektoré programátorské chyby, ako sú napríklad syntaktické chyby, neicializované premenné, nesprávna práca s pamäťou, delenie nulou, opakované uzatváranie súboru a ďalšie. Medzi statické testovanie patrí napríklad revízia kódu alebo použitie niektorého nástroja pre statické testovanie, napríklad syntaktický analyzátor, sémantický analyzátor, analyzátor závislostí, atď. Statické testovanie môže byť manuálne alebo automatické. Tento typ testovania je možný v ľubovoľnej fáze vývoja softvéru.

- **Dynamické testovanie** – je testovanie, ktoré vyžaduje beh testovaného softvéru. Dynamické testovanie môže produkovať výsledky, ktoré nie sú so statickou analýzou možné, alebo by boli použitím statického testovania časovo náročné. Tento typ testovania vyžaduje spustiteľnú verziu vyvíjaného softvéru.

Medzi ďalšie delenie patrí delenie testovania podľa spôsobu vykonávania testov.

- **Manuálne testovanie** – pri manuálnom testovaní vykonáva test používateľ priamou interakciou s testovaným softvérom. Tento typ testovania sa používa, pokiaľ test potrebuje ľudské ohodnotenie alebo úsudok.

- **Automatické testovanie** – automatické testovanie je prevádzané strojom. Tento typ testovania sa zavádza väčšinou do rozsiahlych projektov. Využíva sa pri opakovanom spúšťaní veľkého množstva testov, alebo testov s veľkými množstvami dát. Pri automatickom testovaní sa využíva nejaký automatizovaný nástroj, pričom môže ísť o nástroje pre vykonávanie testov, alebo o nástroje pre správu testov.

Trocha odlišne sa pristupuje k deleniu testov na základe toho, aké znalosti máme o testovanom produkte. Môže ísť o:

- **Testovanie pomocou bielej skrinky** – tento typ testovania vyžaduje prístup k zdrojovému kódu softvéru. Testy sa potom vytvárajú na základe znalosti tohto zdrojového kódu. Testovanie pomocou bielej skrinky však nemusí odhaliť neimplementované časti systému, alebo chýbajúce požiadavky.

- **Testovanie pomocou čiernej skrinky** – tento typ testovania nevyžaduje znalosť zdrojového kódu testovaného softvéru počas vytvárania testov. Pri návrhu testov sa používa externý pohľad na testovaný softvér. Produkt berieme ako čiernu skrinku, do ktorej sa nevieme pozrieť. O tejto čiernej skrinke vieme len to, ako sa chová navonok a ako vyzerá. Pri tomto type testovania sa zameriavame na vstupy a výstupy programu bez znalosti toho, ako je naimplementovaný.

- **Testovanie pomocou sivej skrinky** – testovanie pomocou sivej skrinky je forma testovania niekde medzi bielou a čiernou skrinkou. Využívajú sa v nej limitované vedomosti o implementácii testovaného softvéru. Nemáme napríklad k dispozícii celý zdrojový kód, ale iba dizajn softvéru, alebo databázu.

Testovanie softvéru môže byť zvyčajne vykonávané na rôznych úrovniach procesu vývoja alebo údržby softvéru. Cieľ testu môže byť rôzny, od jedného modulu, cez skupinu modulov, až po celý systém. Toto delenie je na základe úrovní testovania softvéru a vyzerá nasledovne:

- **Unit testy** – verifikujú funkcionalitu softvéru v častiach, ktoré sú testovateľné oddelene. Unit testy sú definované presnejšie v štandarde IEEE1008-87 [1].
- **Integračné testovanie** – je proces, v ktorom sa verifikuje interakcia medzi viacerými softvérovými komponentami.
- **Systémové testovanie** – zaoberá sa správaním celého systému. Počas systémového testovania sa aplikácia testuje ako celok, a preto je toto testovanie vhodné pre neskoršie fázy vývoja.
- **Akceptačné testovanie** – testuje správanie systému voči požiadavkám zákazníka. Akceptačné testy overujú to, ako je daný softvér schopný byť nasadený do ostrej prevádzky, a typicky sú súčasťou prevzatia softvéru zákazníkom.

Testovanie môže byť cielené na verifikovanie rôznych vlastností softvéru. Na základe toho, na akú časť systému je testovanie zamerané, môžeme testovanie rozdeliť na:

- **Inštalačné testovanie** – toto testovanie overuje, či je vytvorený softvér možné nainštalovať na cieľové prostredie. Na tento typ testovania sa môžeme pozrieť ako na systémové testovanie ovplyvnené viacerými faktormi, ako sú napríklad hardvérové požiadavky, alebo požiadavky na operačný systém.
- **Testovanie výkonu** – tento typ testovania je špeciálne zameraný na to, či softvér spĺňa stanovené požiadavky na výkon, ako napríklad doba odozvy, alebo počet vykonaných operácii za čas.
- **Regresné testovanie** – podľa štandardu IEEE610.12 [2] je regresné testovanie „selektívne pretestovávanie systému alebo komponenty na verifikáciu, že zmeny nespôsobili nechcené efekty a že systém alebo komponenta stále spĺňa špecifikované požiadavky.“ Myšlienkou tohto testovania je overiť to, že nové zmeny do funkcionality systému nezaniesli do systému nové typy chýb. Bežným spôsobom tohto testovania je periodické spúšťanie testov vytvorených v minulosti a kontrolovanie, či sa zmenilo správanie systému, a či sa chyby opravené v minulosti znovu neprejavili. Po tom, čo je softvér zmenený vo forme opravy chýb alebo pridania novej funkcionality, je regresné testovanie odporúčané pre overenie, že funkcionalita, ktorá predtým fungovala, sa stále správa tak, ako je od nej očakávané.
- **Testovanie použiteľnosti** – toto testovanie overuje, aké zložité je pre koncového používateľa pracovať s daným softvérom. Toto testovanie overuje taktiež prácu s dokumentáciou, alebo napríklad schopnosť zotavenia sa z chyby.

Uvedené delenia testovania a ich typy patria medzi najzákladnejšie. Existuje ešte niekoľko spôsobov delenia testovania softvéru, ktoré však pre rozsah nie sú v tejto kapitole popísané.

typ testu	počet testov v produkte MCO	počet testov v produkte SMSCv5
cluster	1846	2534
prerekvizita	355	390
odrekvizita	346	389

Tabuľka 2.1: Prehľad celkového počtu testov v dvoch najväčších produktoch firmy Acision

2.2 Testovanie softvéru v praxi

Firma Acision[1], pre ktorú je táto bakalárska práca vytváraná, vyvíja niekoľko komerčných produktov v sfére messagingu. Pri produktoch je kladený veľký dôraz práve na testovanie. Jedným z najväčších produktov, ktorý firma vyvíja je produkt *Message Controller*[2] (ďalej len MCO). Táto bakalárska práca je zameraná pre potreby tohto produktu a jeho derivátov. MCO je komerčný systém, ktorý je na trhu dostupný už niekoľko rokov, a preto sú hodnoty uvedené v tejto bakalárskej práci z oddelenia údržby[3], ktoré je zodpovedné za opravovanie chýb a celkovú údržbu systému.

Jednotlivé produkty sú každodenne testované regresnou sadou testov, ktorá v prípade MCO pozostáva z viac ako 1700 testov. Väčšina týchto testov pristupuje k testovanému softvéru ako k sivej skrinke, popísanej v predchádzajúcej kapitole. Každý test sa zameriava na nejakú vlastnosť testovaného systému pre verifikáciu, že sa táto vlastnosť správa podľa špecifikácie. Tabuľka 2.1 zobrazuje prehľad celkového počtu testov dvoch najväčších produktoch, ktoré sa vo firme Acision testujú. Jednotlivé testy je možné obmedziť tak, aby sa spúšťali iba pri testovaní špecifickej verzie produktu. V regresnom testovaní sa z tohto dôvodu nespúšťajú všetky testy, ktoré pre daný produkt existujú, ale spúšťa sa len ich podmnožina.

Údržba produktu MCO funguje formou kontinuálnej integrácie[4]. Kontinuálna integrácia je vývoj softvéru, pri ktorom členovia tímu integrujú ich prácu často. Obvykle každý člen tímu integruje svoju prácu aspoň raz denne, čo vedie k niekoľkým integráciám za deň [7]. Každá integrácia je verifikovaná automatickým prekladom[5], aby sa detekovali chyby čo najskôr. Raz denne sa potom pomocou regresných testov zisťuje, či integrácia novej časti softvéru prebehla úspešne.

V praxi to zjednodušene prebieha tak, že je vývojárovi[6] pridelený tzv. ticket, ktorý slúži na popis nájdenej chyby v systéme. Pomocou verzovacieho systému si vezme aktuálnu verziu zdrojových súborov a začne nájdenú chybu opravovať. Po tom, čo vývojár chybu opraví, zmeny do zdrojového kódu mu skontroluje iný vývojár. Ak so zmenami do zdrojového kódu obaja súhlasia, uložia tieto zmeny do verzovacieho systému. Týmto spôsobom sa zaisťuje kontinuálna integrácia na strane vývojárov.

Paralelne s týmto vývojom beží na jednom serveri program, ktorý monitoruje zmeny vo verzovacom systéme. Ak sa nájde nejaká zmena v zdrojových kódoch, spustí sa automatický preklad a zdrojové kódy sa spolu s dokumentáciou nanovo skompilujú. Vytvorí sa spustiteľná verzia nového softvéru. Ak sa preklad zdrojových kódov nepodarí, vývojár,

[1] http://www.acision.com/
[2] http://www.acision.com/services/messaging-infrastructure/message-controller
[3] angl. maintenance
[4] angl. continuous integration
[5] angl. build
[6] angl. developer

ktorý danú chybu spôsobil, je na stav upozornený, a môže sa čo najrýchlejšie pustiť do opravy spôsobenej chyby kompilácie.

Z pohľadu testera je situácia podobná. Testerovi je taktiež pridelený ticket. Pomocou verzovacieho systému si vezme aktuálnu verziu všetkých dostupných testov a začne pracovať na novom teste, ktorého úlohou bude overiť, že vývojár opravil chybu popísanú v tickete tak, ako je uvedené v špecifikácii. Tester po dokončení testu zaradí tento test pomocou verzovacieho systému do aktuálneho repozitára a vyžiada si kontrolu napísaného testu.

Raz denne (typicky v noci) sa potom vezme najnovšia spustiteľná verzia softvéru a na tomto softvéri sa pustí aktuálna verzia všetkých dostupných testov. Pomocou regresného testovania (viď kapitola 2.1) sa potom overí, či sa zmenou zdrojového kódu nezaniesli do systému nové chyby. Všetky testy sú plne automatizované a dynamicky overujú, že testovaný systém splňuje všetky požiadavky.

Týmito postupmi je zaistená kontinuálna integrácia. Implementácii kontinuálnej integrácie sa venuje napríklad dokument [4]. Postupnou iteráciou týchto procesov sa dosahuje výsledná kvalita systému. Väčšina testov pristupuje k testovanému systému metódou sivej skrinky, takže tester nepotrebuje znalosť zdrojového kódu a môže tak vytvárať testy ešte pred dokončením samotnej implementácie systému.

Pre potreby tohto testovania vznikol nástroj, ktorý je schopný zostaviť plán všetkých testov a tieto testy postupne vykonať. Tento nástroj je nutný hlavne preto, lebo testy majú oddelenú časť pre test funkcionality a konfiguračné zmeny. Každý test patrí do nejakej skupiny, ktorá má rovnakú konfiguráciu systému. Tieto jednotlivé testy navyše majú medzi sebou závislosti, ktoré je nutné dodržiavať. Plánovač testov je schopný tieto závislosti riešiť a efektívne naplánovať poradie všetkých testov tak, aby sa čo najviac skrátil čas potrebný pre otestovanie softvéru. Nie je preto možné spúšťať testy v ľubovoľnom poradí s tým, že by sme nové testy pridávali nakoniec. Viac sa o tomto nástroji dozvieme v kapitole 2.4. Výhodou tohto nástroja je hlavne to, že je schopný jednoducho zaradiť novovytvorené testy do regresnej sady, ktorá testuje systém množinou všetkých testov. Postupným pridávaním testov do tejto regresnej sady sa však zvýšil čas potrebný pre otestovanie celého systému.

Regresné testovanie sa však po dobe stalo veľmi časovo náročné, nakoľko čas potrebný pre otestovanie celého systému sa zvýšil až na 15 hodín, a stále narastal. Vznikol problém, že sa počas noci nestihol otestovať celý systém, a na začiatku pracovnej doby sa ešte muselo čakať na výsledky regresného testovania.

Pre riešenie tohto problému vznikol nápad vytvorenia rozšíreného plánovača testov, ktorý by bol schopný spúšťať testy na viacerých testovacích systémoch. V praxi to znamená, že množina všetkých regresných testov sa rozdelí na niekoľko podmnožín, kde každá podmnožina testov sa spustí na samostatnom testovacom stroji. Týmto spôsobom sme schopní skrátiť čas potrebný pre otestovanie celého systému bez toho, aby sme museli znížiť počet testov v regresnej sade.

2.3 Princípy znižovania cien regresného testovania

Regresné testovanie je operácia náročná na čas, a preto sa typicky prevádza počas noci. Napriek tomu niekedy na otestovanie systému nestačí ani celá noc, nakoľko sa na testovaní podieľa iba jeden testovací stroj. Takáto situácia prináša mnohé riziká a môže negatívne ovplyvniť napríklad čas dodania softvéru, nakoľko je softvér nutné pred vydaním zákazníkovi otestovať práve regresným testovaním. Postupom času sa však čas potrebný pre dokončenie regresného testovania zvyšuje, nakoľko sa pridávajú stále nové a nové testy. Dokument

[6] popisuje niekoľko možností znižovania náročnosti regresného testovania. Medzi niektoré možnosti znižovania cien regresného testovania patria:

Voľba testov pre regresné testovanie

Medzi jedným zo základných spôsobov zníženia času potrebného pre regresné testovanie je zvolenie reprezentatívnej podmnožiny testov, z dostupnej testovacej sady. Tento prístup má však jednu obrovskú nevýhodu. Tou nevýhodou je, že môžeme prísť práve o testy, ktoré by odhalili v systéme chybu.

Prioritizácia testov

Prioritizácia testov znamená, že sú testy, ktoré odhaľujú dôležité chyby, vykonávané medzi prvými. Cieľom tohto prístupu je nájdenie kritických chýb v systéme čo najskôr. Ďalším prístupom môže byť prioritizácia testov na základe toho, že testy, ktoré pokrývajú najviac zdrojového kódu, sú uprednostňované.

Distribúcia testov

Distribúcia testov znamená, že testy sú rozdelené do niekoľkých častí, pričom platí, že každá z týchto častí je spúšťaná na samostatnom stroji. Až donedávna bolo však vyčlenenie nového stroja na testovanie náročné na zdroje. V dnešnej sfére virtualizácie a cloudu sa však tento prístup považuje za jeden z najlepších. Metódam distribúcie testov sa venoval napríklad Gregory M. Kapfhammer, ktorý v dokumente [8] definoval niekoľko vlastností nástroja pre distribúciu testov, ktoré by mali byť splnené pre efektívnu distribúciu regresných testov. Medzi tieto vlastnosti patrí:

- **Transparentná a automatická distribúcia** – distribúcia n testov na m strojov by mala byť tak transparentná a automatická, ako je to len možné.
- **Nezávislosť medzi testami** – test je považovaný za nezávislý, ak jeho úspech či neúspech nezávisí na testoch, ktoré sú aktuálne spúšťané na ostatných strojoch.
- **Distribúcia záťaže** – záťaž pre jednotlivé testovacie stroje by mala byť rovnomerná.
- **Integrita testovacej sady** – distribúcia neovplyvní správnosť výsledkov a nezabráni vykonávaniu testov.
- **Centralizovaná správa testov** – vykonávanie testov a zobrazovanie výsledkov musí byť kontrolovateľné z jedného centralizovaného miesta.

2.4 Princíp použitého plánovača testov

Pre popis rozšírenia plánovača testov si najprv musíme vysvetliť princíp, akým daný plánovač testov funguje. Pre vysvetlenie tohto princípu si však najprv zavedieme niektoré kľúčové pojmy.

- **cluster** – je test, ktorý verifikuje funkčnosť nejakej časti systému. Každý cluster je pomenovaný pomocou prefixu `cl_`. Clustre sú určené na jednoznačné overenie, že daná funkcionalita softvéru funguje presne podľa špecifikácie. Príkladom clusteru môže byť test na overenie, že po prekročení maximálnej veľkosti sms správy sa táto

správa zahodí. Kód clusteru by potom zahŕňal napríklad poslanie dvoch sms správ. Prvá správa by bola rovnako veľká, ako nastavený limit pre maximálnu veľkosť sms správy. Odoslanie takejto sms správy by skončilo úspešným doručením tejto správy príjemcovi. Druhá sms správa by maximálny limit veľkosti prekračovala o jeden bajt. Cluster by sa takúto správu pokúsil odoslať, pričom by sa kontrolovalo, že správa by príjemcovi nebola doručená.

- **prerekvizita** – je test, ktorý nastavuje nejakú vlastnosť systému na požadovanú hodnotu, prípadne zapína časť systému. Každá prerekvizita má svoju vlastnú odrekvizitu a je pomenovaná pomocou prefixu `p_`. Ako príklad si môžeme rozviesť situáciu s posielaním sms správy prekračujúcej maximálny limit veľkosti správ. Kód v prerekvizite by napríklad obsahoval zapnutie komponenty, ktorá je zodpovedná na doručovanie sms správ. Ďalej by prerekvizita obsahovala kód pre zapnutie funkcionality, ktorá je zodpovedná za zahadzovanie správ prekračujúcich maximálnu veľkosť. Poslednou časťou kódu prerekvizity by mohlo byť zmenenie defaultnej hodnoty maximálnej veľkosti odosielaných sms správ zo 160 bajtov na 140 bajtov.

- **odrekvizita** – je test, ktorý vracia zmeny systému spôsobené jej vlastnou prerekvizitou. Každá odrekvizita je pomenovaná pomocou prefixu `u_`. Odrekvizita by v prípade vyššie spomínaného príkladu obsahovala obnovu zmeny maximálnej veľkosti odosielaných sms správ naspať na defaultnú hodnotu. Ďalej by odrekvizita obsahovala kód pre vypnutie funkcionality zahadzovania sms správ prekračujúcich maximálnu nastavenú veľkosť, a nakoniec vypnutie komponenty zodpovednej pre doručovanie sms správ.

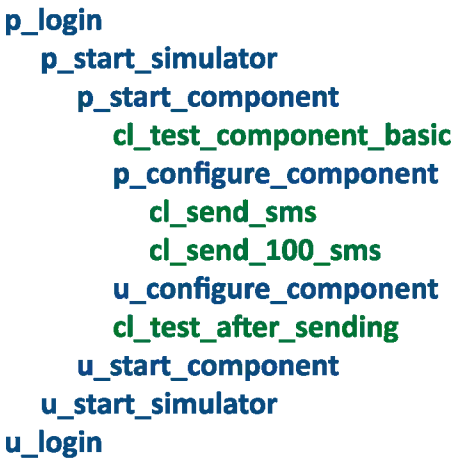
- **test** – je pomenovanie pre cluster, prerekvizitu alebo odrekvizitu. Každý test je ohraničený jednou funkciou v jazyku Tcl/Tk.

Plánovač testov je napísaný v skriptovacom jazyku Tck/Tk[7]. Je špeciálne navrhnutý pre potreby testovania softvéru, ktorý je vysoko konfigurovateľný. Využíva oddelenie časti kódu pre samotný test a pre konfiguračné zmeny. Idea plánovača je taká, že každý test sa rozdelí na prerekvizitu (prípadne viac prerekvizít) a cluster. Cluster obsahuje zdrojový kód, ktorý verifikuje, že daná časť systému funguje podľa špecifikácie. Prerekvizita obsahuje zdrojový kód, ktorý mení chovanie systému pre potreby verifikovania funkcionality systému. Zmena chovania systému môže byť prevádzaná napríklad príkazovou riadkou, zmenou konfiguračného súboru a podobne. Tento princíp usporiadava všetky testy do štruktúry, ktorá je znázornená na obrázku 2.1. Jednoduchý plán testov z produktu MCO je zobrazený v prílohe B.

Jednotlivé testy sa potom vykonávajú sekvenčne. Výhodou tohto prístupu je, že v prípade, že sa zmena konfigurácie nepodarila, daný cluster môžeme preskočiť, a tým ušetriť čas potrebný na dokončenie testovania. Test je úspešný, pokiaľ skončí s nulovou návratovou hodnotou. V opačnom prípade je test neúspešný. Ďalšou výhodou je znovupoužiteľnosť prerekvizít. Jednotlivé prerekvizity totiž môžeme kombinovať a využiť ich tak pre potreby rôznych typov scenárov pre testy.

Na obrázku 2.1 je znázornený príklad štyroch clusterov, pričom platí, že tieto clustre vyžadujú dve rôzne konfigurácie systému. Na obrázku je znázornené, že pre potreby dvoch clusterov `cl_send_sms` a `cl_send_100_sms` je nutné vykonať štyri prerekvizity, a to prerekvizity `p_login`, `p_start_component`, `p_start_simulator` a `p_configure_component`.

[7] https://www.tcl.tk/



Obrázok 2.1: Ukážka plánu testov

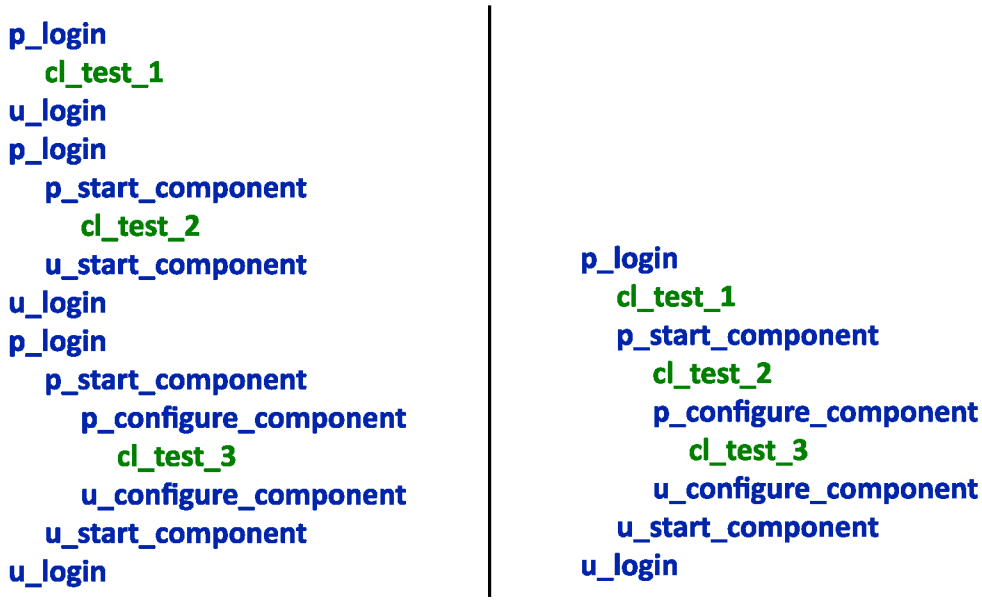
Zvyšným dvom clusterom stačia iba 3 prerekvizity, a to prerekvizity `p_start_simulator`, `p_login` a `p_start_component`. Cluster `cl_test_after_sending` má rovnakú množinu prerekvizít ako cluster `cl_test_component_basic`, avšak tento cluster sa spúšťa až za clusterom `cl_send_100_sms`. Je preto nutné vrátiť konfiguráciu systému, ktorú vykonala prerekvizita `p_configure_component` pomocou odrekvizity `u_configure_component`. Takýmto spôsobom sa dosiahne, aby mal tento cluster rovnakú konfiguráciu systému ako má cluster `cl_test_component_basic`. Po dokončení testovania všetkých clusterov ešte musíme spustiť zvyšné tri odrekvizity, aby sa systém zanechal s rovnakou konfiguráciou, s akou sa začínalo pri testovaní.

Pre zjednodušenie budú v tomto dokumente všetky clustre zobrazené zelenou farbou. Prerekvizity a odrekvizity budú zobrazené modrou farbou. Takýmto spôsobom si môžeme jednoducho znázorniť princíp plánovača testov, ktorý oddeľuje testy slúžiace na testovanie funkčnosti systému, a testy slúžiace na zmenu vlastností systému.

Jednotlivé prerekvizity na seba môžu, ale nemusia byť závislé, a preto sa poradie spúšťania prerekvizít môže pre clustre meniť. Príkladom závislosti môžu byť 2 prerekvizity, kde jedna prerekvizita zapína nejakú súčasť systému s jeho základnými nastaveniami a druhá prerekvizita tieto nastavenia mení. V druhej prerekvizite môže byť preto nastavené, aby sa spustila, iba ak sa predtým úspešne splnila prerekvizita na zapnutie spomínanej časti systému. Plánovač testov sa snaží plánovať jednotlivé clustre za seba tak, aby sa nemuseli nutne vykonať všetky prerekvizity a odrekvizity pre každý cluster zvlášť. Znamená to, že ak majú dva clustre niektoré prerekvizity rovnaké, tak sa tieto clustre naplánujú tak, že sa rovnaké prerekvizity združia a spúšťajú sa spravidla len raz. Táto vlastnosť je znázornená na obrázku 2.2. Ľavá časť obrázku ukazuje, ako by vyzeral plán, ak by plánovač testov nedisponoval vlastnosťou združovania testov.

V regresnej sade sa bežne nachádzajú clustre, ktoré majú rovnakú množinu prerekvizít. Takéto clustre sa radia za seba, aby sa minimalizovalo spúšťanie prerekvizít a odrekvizít, ktoré majú spoločné. Neplatí teda pravidlo, že každý cluster má svoju vlastnú unikátnu prerekvizitu.

Medzi ďalšie výhody tohto plánovača testov patrí možnosť špecifikovania pravidiel, s akými sa budú dané testy spúšťať. Je napríklad možné nastaviť clusteru zoznam cluste-



Obrázok 2.2: Ukážka vlastnosti združovania testov

rov, takzvaných *Preoptov*, ktoré musia byť úspešne vykonané ešte predtým, ako sa vykoná samotný cluster. V prípade, že sa aspoň jeden z týchto *Preoptov* nevykoná úspešne, cluster sa preskočí. Výhoda tohto prístupu je v tom, že ak zo sady testov, ktoré testujú podobnú funkcionalitu, neprejde nejaký základný test, môžeme predpokladať, že všetky pokročilé testy by taktiež skončili neúspechom, a preto ich môžeme preskočiť.

Jednotlivé testy môžeme označovať vhodne zvoleným názvom nejakej funkcionality, pričom môžeme využiť vlastnosť plánovača, ktorý nám naplánuje len testy zvolené týmto názvom. Týmto spôsobom môžeme v regresnej sade vytvárať isté logické celky, ktoré v prípade potreby môžeme otestovať samostatne. Pre regresný beh je použitá funkcionalita *all*, ktorá združuje všetky testy, ktoré nie sú explicitne označené funkcionalitou *notall*.

Dva najväčšie produkty vo firme Acision, sú produkty MCO a SMSCv5. Regresné testovanie týchto dvoch produktov trvá približne 15 hodín. Spúšťanie regresných testov pre tieto dva produkty je zabezpečované príkazmi uvedenými v príkladoch 2.1 a 2.2. Spúšťanie takýchto regresných testov využíva práve spomínanú funkcionalitu *all*.

```
meszarosf@mco-v156#./Run_test.tcl -i mco.ini -B TEST.REVIEW_23 -f all -tw
```

Príklad 2.1: Spúšťanie regresných testov v produkte MCO

```
meszarosf@smsc-v156#./Run_test.tcl -i smsc.ini -f all 2nd -tw
```

Príklad 2.2: Spúšťanie regresných testov v produkte SMSCv5

Za zmienku patrí aj možnosť označiť niektoré testy do kategórie známych chýb, označenej ako *known bugs*. Testy z tejto kategórie sú v prípade zlyhania vylúčené zo zoznamu testov, ktoré zlyhali. Typicky sa takto označujú testy, ktoré úspešne testujú nejakú funkcionalitu, ale vďaka nejakej nesúvisiacej chybe dopadajú neúspechom. Ďalej sa táto funkcionalita využíva napríklad pre testy, ktoré testujú nejakú časť systému, ktorá sa nestihla do

danej vydanej verzie produktu opraviť. V prípade, ak je test hotový, tak sa nechá v regresnej sade, ale pre danú verziu sa test pridá do kategórie známych chýb. Takýmto testom je možné explicitne uviesť rozsah verzie softvéru, v ktorom je tento test chybný. Tieto testy aj spolu s verziami softvéru, v ktorom sa chyby prejavujú, sú uložené v súbore *KNOWN.BUGS*. Označenie testu, ktorý končí neúspechom vďaka nejakej nesúvisiacej chybe je znázornené na príklade 2.3. Tento test testuje chybu, ktorá sa objavila vo verzii produktu 2.3-05.01, avšak opravená bola až vo verzii 2.3-05.02.

```
# MCRD0112570 (Problem with response to SMPP Bind operation)
cl_smpp_028                    >=2.3-05.01 ~ >=2.3-05.02
```

Príklad 2.3: Príklad testu ktorý končí neúspechom pre verziu softvéru 2.3-05.01

Medzi poslednú vlastnosť, ktorú si spomenieme, je možnosť interaktívneho módu. Interaktívny mód je vlastnosť, ktorá nám umožňuje pozastaviť beh testovania pred alebo za nejakým testom. Interaktívny mód taktiež umožňuje pozastaviť beh testovania, ak sa niektorý test ukončí neúspechom. Pozastavenie plánovača znamená, že je možné si v špeciálnom režime spúšťať vlastné funkcie alebo celé testy. Interaktívny mód umožňuje zmeniť beh testovania na základe špecifických požiadavkov testera, ktorý túto vlastnosť využíva. Tento mód sa však v regresnom testovaní nepoužíva, nakoľko sa využitím tohoto módu stráca vlastnosť plnej automatizácie.

Spomínaný plánovač testov má ešte niekoľko vlastností, ktoré však nie sú pre zameranie tejto bakalárskej práce dôležité. Tieto vlastnosti v kapitole nie sú uvedené pre ich rozsah.

Každý test má svoj zdrojový kód v spomínanom jazyku Tcl/Tk a svoju vlastnú hlavičku, ktorá obsahuje informácie, potrebné pre identifikovanie testu a jeho úspešné zaradenie do regresnej sady. Zdrojové kódy testov sú logicky rozdelené a umiestnené v súboroch začínajúcich predponou *CL-* a s príponou *.tcl*, napríklad *CL-SMPP.tcl* alebo *CL-GSM.tcl*. Hlavičky testov môžu byť taktiež logicky delené na niekoľko celkov a začínajú predponou *TEST.*, napríklad *TEST.ALL* alebo *TEST.REVIEW_23*. Takýmto spôsobom sú napríklad delené testy v produkte MCO, ktoré prešli kontrolou, a sú úspešne zaradené do regresnej sady (súbor *TEST.ALL*) a testy, ktoré sú do regresnej sady zaradené, ale zatiaľ neboli nikým skontrolované (súbor *TEST.REVIEW_23*).

Hlavička každého testu je štandardizovaná a obsahuje nasledovné informácie:

- **Name** – názov testu
- **Author** – autor testu
- **Description** – stručný popis toho, čo daný test vykonáva
- **Function** – názov funkcie v jazyku Tcl/Tk, ktorá sa daným testom spustí. Každý test je tvorený jednou funkciou, ktorá je typicky rovnaká ako názov testu. Ako nepovinný parameter je možné špecifikovať argumenty funkcie. Príkladom môže byť cluster s názvom `cl_send_sms` ktorý používa ako argument počet sms správ ktoré má poslať, s defaultnou hodnotou 100 sms správ. Takýto cluster má označenú položku *Function* ako `cl_send_sms 100`. V regresnej sade potom môže existovať cluster s názvom `cl_send_sms_long_running`, s položkou *Function* označenou ako `cl_send_sms 10000`, ktorý posiela takýchto správ stonásobne viac.
- **Pre** – množina všetkých prerekvizít, ktoré musia byť v čase spustenia daného testu úspešne „aktivované“. Pod týmto pojmom sa myslia tie prerekvizity, ktorých konfiguračné zmeny neboli odstránené spustením príslušnej odrekvizity. Princíp plánovača

je ten, že prerekvizity sú určené na zmenu stavu testovacieho prostredia. Množina prerekvizít uvedených v tejto položke teda predstavuje prerekvizity, ktoré je nutné úspešne spustiť, aby sme testovacie prostredie dostali do stavu, v ktorom bude možné spustiť aktuálny test.

- **Preopt** – množina všetkých clusterov, ktoré musia byť úspešne otestované v čase, keď sa púšťa aktuálny test. Ak aspoň jeden cluster z tejto množiny dopadol neúspechom, aktuálny cluster sa preskočí. Táto položka je určená iba pre clustre.
- **Restore** – názov prerekvizity, ktorá patrí je určená pre nastavenie testovacieho prostredia pre potreby testu. Táto položka je určená iba pre odrekvizity.
- **Undo** – názov odrekvizity, ktorá je určená pre obnovu testovacieho prostredia. Táto položka je určená iba pre prerekvizity.
- **Functionalities** – množina všetkých názvov funkcionalít, ktorými je možné daný test spustiť. Táto položka umožňuje logicky rozdeliť všetky testy do podmnožín, ktoré je možné spúšťať samostatne. Plánovač umožňuje pomocou prepínača **-f** určiť, ktoré funkcionality sa pri testovaní spustia. Pri plánovaní testov sa potom pracuje iba s testami, ktoré sú týmito funkcionalitami označené. Štandardná hodnota pre všetky clustre je hodnota *all*, a pre všetky prerekvizity a odrekvizity je to hodnota *notall*. Typickým príkladom použitia je nastavenie testov, ktoré sa v nočných regresných testoch bežne nespúšťajú, nakoľko je ich beh príliš časovo náročný, na hodnotu *long_running*. V plánovači testov je potom možné spustiť funkcionalitu *all* bez testov, ktoré sú označené funkcionalitou *long_running*.
- **Sets** – umožňuje ďalší spôsob logického rozdeľovania testov na podmnožiny, podobne ako položka *Functionalities*.
- **Versions** – dolné a prípadne aj horné ohraničenie hodnoty verzie softvéru, na ktorej je daný test možné pustiť (Napríklad test určený pre softvér od verzie 2.3-0.0 vrátane, až do verzie 2.3-04.06 môžeme označiť ako: >= 2.3-00.00 ∼ >= 2.3-04.06).
- **CR** – popisok slúžiaci na označenie čísla ticketu, alebo čísla zmeny požiadavkov systému[8], ktoré sú do systému implementované na vyžiadanie zákazníka. Pomocou prepínača **-cr** je v plánovači testov možné naplánovať a spustiť všetky testy, ktoré majú dané označenie v tejto položke.
- **CRS** – informačný popisok slúžiaci na označenie konkrétnej časti naimplementovanej zmeny systému.

Hlavička testu zo súboru *TEST.REVIEW_23* je zobrazená na príklade 2.4.

[8] angl. change request

```
Cluster
Name: cl_smpp_truncate_025
Author: Filip Meszaros
Description: Priklad hlavicky testu zo suboru TEST.REVIEW_23
Function: cl_smpp_truncate_025
Pre: p_mobiles p_sol_default p_dc_default p_gbg_default p_sol_sf p_smpp_
    truncate
Preopt: cl_smpp_truncate_001
Restore:
Undo:
Functionalities: smpp truncate licensed gsm
Sets:
Versions: >= 2.3-05.01
CR: MCRE0112412 CR00025702 MCRE0112535
CRS:
```

Príklad 2.4: Hlavička testu

Táto sekcia obsahovala zhrnutie toho, akým princípom funguje plánovač testov, vytvorený firmou Acision. Na záver si ešte uvedieme zoznam výhod a nevýhod použitia tohto nástroja.

Výhody:

- jednoduché zaradenie nového testu do regresnej sady
- možná znovu-použiteľnosť prerekvizít
- združovanie prerekvizít
- preskakovanie testov o ktorých vieme, že by skončili neúspechom
- možné označovanie testov do kategórie známych chýb (súbor KNOWN.BUGS)
- interaktívny mód pre testovanie
- označovanie testov do logických celkov a ich spúšťanie
- oddelený kód pre zmenu konfigurácie a pre samotné testy
- veľká možnosť zmien chovania pomocou prepínačov

Nevýhody:

- náhodný neúspech nejakej prerekvizity môže znamenať vynechanie väčšiny testov v regresnom testovaní
- jedna chybne vyplnená hlavička testu môže znefunkčniť celý plánovač
- jednotlivé testy sa v prípade chyby systému nemusia úspešne z tejto chyby zotaviť, a môžu spustiť lavínu chýb, ktoré by inak nenastali
- v prípade, že sa konfiguračné zmeny v odrekvizitách úspešne neobnovia, dochádza k neodpovedajúcim výsledkom z testovania
- plánovanie testov je relatívne pomalé
- prehľad v testoch a ich údržba je náročná

Kapitola 3

Návrh riešenia

Táto kapitola sa zaoberá základným návrhom implementácie rozšírenia plánovača testov pre podporu distribúcie testov na viaceré systémy. Kapitola popisuje rôzne možnosti distribúcie testov, ich vzájomné porovnanie a taktiež výhody a nevýhody jednotlivých prístupov. V kapitole si popíšeme návrh spúšťania testov a prístup k centralizovanej správe plánovača testov. Ďalej si popíšeme návrh riešenia komunikácie medzi jednotlivými regresnými behmi, vykonávaných na samostatných testovacích strojoch, na spôsob čakania na dokončenie testovania. V kapitole je taktiež popísaný návrh zbierania výsledkov a ich interpretácia. Posledná časť kapitoly poukazuje na nové typy problémov, ktoré vznikli pri vytváraní rozšírenia plánovača testov, a navrhuje možnosti ich riešenia.

3.1 Možnosti distribúcie testov

Pri návrhu distribúcie testov som vychádzal z informácii o počte testov a prerekvizít produktov, ktoré daný plánovač testov používajú. Informácie som čerpal z dvoch najväčších produktov (MCO a SMSCv5), ktoré tento plánovač testov každodenne používajú. Pri každom regresnom behu sa zbierajú štatistiky o tom, ako dlho sa každý test vykonával. V tabuľkách 3.1 a 3.2 sú zobrazené informácie o testoch z týchto dvoch najväčších produktov.

Z tabuliek vyplýva, že produkt MCO má priemerné dĺžky trvaní clusterov, prerekvizít a odrekvizít veľmi podobné. V produkte SMSCv5 je priemerná dĺžka vykonávania clusterov niekoľkonásobne vyššia, ako priemerná dĺžka vykonávania prerekvizít alebo odrekvizít. Tieto tabuľky ukazujú celkový počet všetkých testov. Niektoré testy sú však z regresnej sady vynechané. Navyše z tabuliek nie je jasné, koľko prerekvizít sa skutočne naplánuje pre jedno regresné testovanie. Z konceptu plánovača totiž vyplýva, že niektoré prerekvizity je nutné spúšťať viacnásobne. Je to preto, lebo každý cluster môže mať inú množinu prerekvizít a všetky tieto kombinácie možných prerekvizít pre každý cluster je potrebné naplánovať.

typ testu	počet testov	min. čas [s]	max. čas[s]	priem. čas [s]	modus	medián
cluster	1791	1	931	18,47	8	7
prerekvizita	323	1	82	19,54	18	1
odrekvizita	323	1	63	19,56	18	7

Tabuľka 3.1: Prehľad testov dostupných pre najnovšiu verziu produktu MCO a štatistiky o dĺžke trvania testov v sekundách

typ testu	počet testov	min. čas [s]	max. čas [s]	priem. čas [s]	modus	medián
cluster	1810	1	1963	26,19	12	3
prerekvizita	247	1	120	9,82	6	1
odrekvizita	247	1	78	6,96	5	1

Tabuľka 3.2: Prehľad testov dostupných pre najnovšiu verziu produktu SMSCv5 a štatistiky o dĺžke trvania testov v sekundách

typ testu	počet testov pre produkt MCO	počet testov pre produkt SMSCv5
cluster	1710	1716
prerekvizita	707	661
odrekvizita	707	661

Tabuľka 3.3: Prehľad počtu naplánovaných testov pri regresnom testovaní v produktoch MCO a SMSCv5

Tabuľka 3.3 zobrazuje počet clusterov a prerekvizít, ktoré sa plánujú pri každodennom spúšťaní nočných regresných testov pomocou funkcionality zvanej *all* a v prípade produktu SMSCv5 aj pomocou funkcionality *2nd*. Z tabuľky vyplýva, že približne 5% testov sa z regresných testov vynecháva. Túto množinu tvoria napríklad testy, ktoré nie sú plne automatické, a vyžadujú tak interakciu s používateľom, alebo testy, o ktorých sa vie, že je nevhodné ich spúšťať spolu s veľkou množinou testov, nakoľko môžu negatívne ovplyvniť výsledky týchto testov. Zároveň z obrázku vyplýva to, že sa spustí viac ako dvojnásobok prerekvizít, ako existuje v testovacej sade. Každá prerekvizita sa teda spustí v priemere viac ako dvakrát. Vo svetle týchto informácii je teda vhodné zameriavať sa pri rozdeľovaní testov na clustre.

V nasledujúcej časti sa pozrieme na navrhnuté možnosti distribúcie testov, ich výhody a nevýhody. Treba podotknúť, že plánovač testov si uchováva dĺžky trvaní behu každého naposledy spusteného testu. Táto štatistika sa môže využiť pri distribúcii testov, aby sme vedeli rovnomerne rozložiť záťaž pre každý testovací systém.

Distribúcia clusterov na základe dĺžky behu clusterov

Tento princíp distribúcie by spočíval v rozdeľovaní clusterov do n podmnožín na základe súčtu dĺžok trvaní všetkých clusterov v danej podmnožine. Každý cluster by bol zaradený do jednej podmnožiny, pričom by platilo, že každá podmnožina by sa spúšťala na samostatnom testovacom systéme. Pri rozdeľovaní testov na podmnožiny by sa bral zreteľ na rozloženie záťaže medzi jednotlivými testovacími systémami.

Výhody:

- jednoduché na implementáciu

Nevýhody:

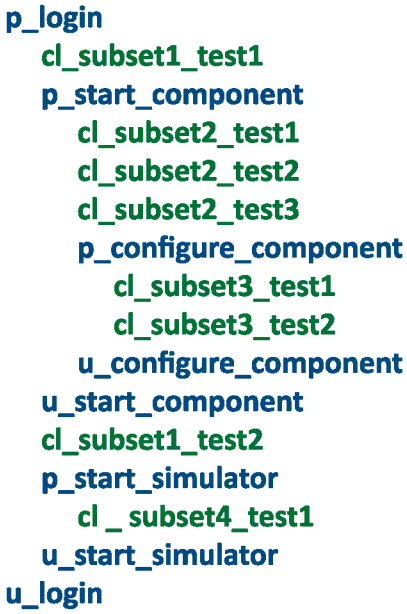
- odhadovaný čas trvania každej podmnožiny clusterov by nereflektoval stav po naplánovaní, pretože by neobsahoval čas potrebný pre spustenie prerekvizít a odrekvizít
- neefektívne združovanie prerekvizít

- nerovnomerné rozloženie záťaže medzi testovacími systémami

Distribúcia clusterov na základe spoločných prerekvizít

Distribúcia na základe spoločných prerekvizít využíva fakt, že v regresnej sade existujú testy, ktoré majú rovnakú množinu prerekvizít. Celú regresnú sadu môžeme týmto spôsobom rozdeliť na n podmnožín, pričom by platilo, že všetky clustre v jednej takejto množine by mali rovnakú množinu prerekvizít. Pri distribúcii testov na m častí, ktoré by sa spúšťali samostatne, by sa do jednotlivých m častí priraďovali vždy celé podmnožiny n. Obrázok 3.1 ilustruje delenie clusterov do štyroch n podmnožín. Pri distribúcii testov by sa potom každá takáto podmnožina brala ako celok a distribuovali by sme tak jednotlivé podmnožiny. Obrovskou výhodou tohto prístupu je to, že vo výsledku by sme ušetrili spúšťanie niekoľkých prerekvizít, nakoľko by sa využila vlastnosť plánovača, a tou je združovanie testov.

Príkladom výsledku môže byť distribúcia testov na dva testovacie systémy, pričom prvý testovací systém by spúšťal clustre z podmnožín 1 a 4 (clustre `cl_subset1_test1`, `cl_subset1_test2` a `cl_subset4_test1`) a druhý testovací systém by spúšťal zvyšné clustre z podmnožín 2 a 3 (clustre `cl_subset2_test1`, `cl_subset2_test2`, `cl_subset2_test3`, `cl_subset3_test1` a cluster `cl_subset3_test2`).



Obrázok 3.1: Príklad rozdeľovania clusterov na podmnožiny na základe spoločných prerekvizít

Výhody:

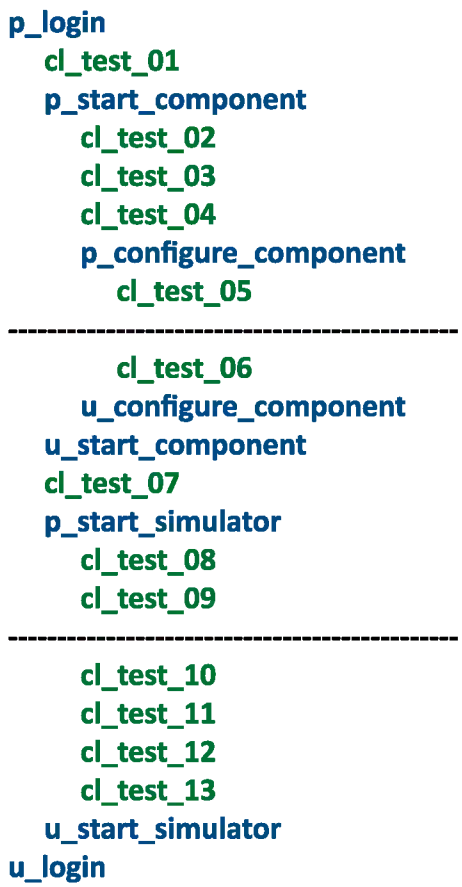
- efektívnejšie združovanie prerekvizít
- efektívnejšie rozloženie záťaže medzi testovacími systémami

Nevýhody:

- odhadovaný čas trvania každej podmnožiny clusterov by nereflektoval stav po naplánovaní, pretože by neobsahoval čas potrebný pre spustenie prerekvizít a odrekvizít

Distribúcia clusterov na základe častí plánu

Idea tohoto prístupu spočíva v tom, že by sa všetky testy naplánovali ako obvykle, a po naplánovaní by sa celý plán rozdelil na m častí. Plánovač by spočítal odhadované časy trvania jednotlivých častí a snažil by sa tieto časti rozdeliť rovnomerne. Po rozdelení by sa do každej časti museli pridať prerekvizity a odrekvizity, o ktoré by sme daným delením prišli. Princíp rozdelenia plánu na 3 časti je znázornený na obrázku 3.2.



Obrázok 3.2: Príklad rozdeľovania clusterov na základe častí plánu

Po rozdelení je dôležité, aby sme každej časti pridali prerekvizity a odrekvizity, o ktoré sme prišli pri delení. Ak sa tieto testy pridajú do plánu, celý plán pre 3 systémy by vyzeral tak, ako je znázornené na obrázku 3.3. Z obrázku vyplýva, že pre vysoký počet testov by bol tento prístup najvhodnejší. Problémom tohto prístupu je však to, že požiadavkom na funkcionalitu plánovača bolo to, aby sa princíp plánovania testov nezmenil. Znamená to, že by sa museli rešpektovať hlavičky testov napríklad na položku *Preopt*, čo znamená, že každému clusteru môžeme nastaviť množinu clusterov, ktoré musia byť úspešne otestované ešte predtým, ako sa spustí test aktuálneho clusteru. Príkladom tohto problému je prípad, kedy by cluster `cl_test_13` mal nastavený *Preopt* na cluster `cl_test_01` a `cl_test_06`. V praxi by to znamenalo, že celá tretia časť rozdeleného plánu by sa musela preplánovať a spomínané clustre by sa do nej museli pridať. Ďalšou nevýhodou je to, že tento prístup by vyžadoval väčší zásah do funkčnosti plánovača testov, nakoľko by sa celá logika plánovača

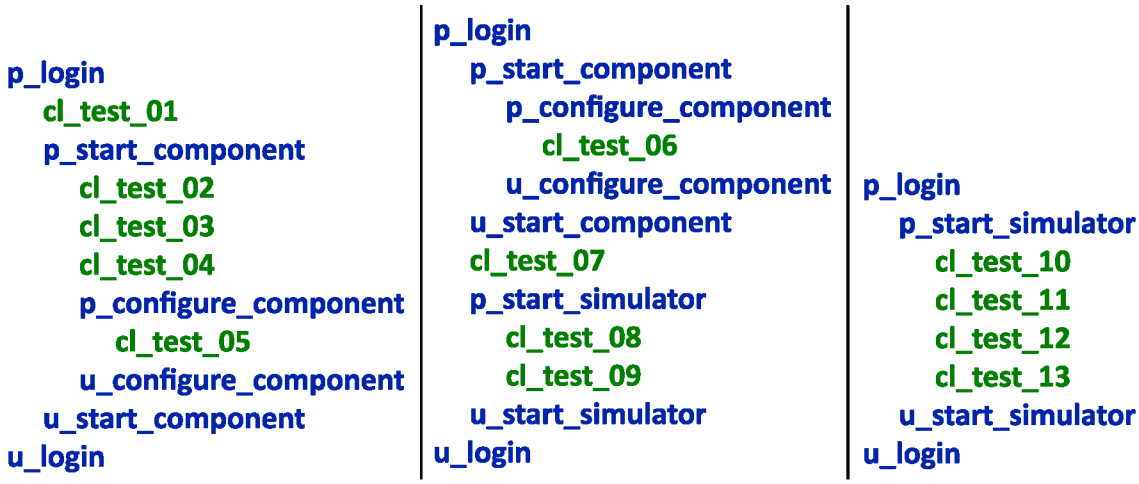
musela upraviť. V prípade, že by ale neexistovala požiadavka na zachovanie funkčnosti, tento prístup by bol najvhodnejší. Týmto problémom, spôsobeným zachovaním funkcionality plánovača, trpia aj predošlé 2 spôsoby distribúcie testov. Pre povahu testov v regresnej sade by však najviac na toto doplácal práve tento princíp distribúcie testov.

Výhody:

- efektívne združovanie prerekvizít
- veľmi presné rozloženie záťaže medzi testovacími systémami (v prípade nezachovania požiadavkov na funkcionalitu)
- presný odhad trvania testovania (v prípade nezachovania požiadavkov na funkcionalitu)

Nevýhody:

- náročné na implementáciu, pretože by sa musela zmenila logika plánovača



Obrázok 3.3: Príklad rozdeľovania clusterov na základe častí plánu - po preplánovaní

3.2 Centralizovaná správa plánovača

Jedným z hlavných vlastností, spomínaných v dokumente [6], ktoré by mal mať nástroj pre distribúciu testov, je vlastnosť centralizovanej správy všetkých testov. Pri návrhu rozšírenia plánovača sa táto vlastnosť brala do úvahy. Rozšírenie je navrhnuté tak, aby bolo z jedného stroja možné spustiť rozšírený plánovač, ktorý by rozdelil množinu testov na m podmnožín, v závislosti na počtu strojov, na ktorých chceme regresné testy spustiť. Jednotlivé podmnožiny testov sa potom predajú novým inštanciám plánovačov testov, ktoré si tieto testy preplánujú a začnú ich vykonávať. Tieto nové inštancie plánovačov sú spustené ako podprocesy hlavného procesu, ktorým je plánovač testov s podporou distribúcie testov. Plánovač testov je navrhnutý tak, aby dokázal rozdistribuovať testy na teoreticky nekonečný počet neprázdnych podmnožín. V prípade, že je počet testovacích systémov väčší ako počet clusterov, sa niektoré testovacie systémy nevyužijú. Centralizovaná správa plánovača umožňuje

zobraziť rozdelený plán testovania pre každý testovací systém zvlášť. Zároveň umožňuje vypísať odhadované dĺžky trvania testovania pre každý systém zvlášť, a umožňuje porovnanie efektivity distribúcie testov voči stavu, kedy by testy neboli distribuované, a spúšťali by sa iba na jednom systéme.

Pri navrhovaní riešenia pre centralizovanú správu testov bolo nutné vymyslieť, akou formou sa budú spúšťať jednotlivé inštancie plánovača testov pre distribuované systémy. Plánovač testov totiž potrebuje konfiguračný súbor, zadávaný parametrom **-i**, ktorý je uvedený pri spúšťaní regresných testov v príkladoch 2.1 a 2.2. Tento konfiguračný súbor obsahuje informácie ako napríklad IP adresa systému, na ktorom sa budú spúšťať testy, číslo verzie softvéru, názov produktu, atď. Príklad 3.1 znázorňuje takýto konfiguračný súbor využívaný v produkte MCO. Rozšírenie plánovača pre distribuované systémy však potrebuje informácii obsiahnutých v konfiguračnom súbore viac. Pre každý distribuovaný systém potrebuje napríklad jednu IP adresu obsiahnutú v položke *Host*. Jedným riešením by bolo dať všetky informácie pre každý distribuovaný systém do jedného konfiguračného súboru. V tomto jednom súbore by sa potom dalo napríklad nastavovať, na koľkých testovacích systémoch sa regresné testy spustia. Pre toto riešenie by však bolo treba upraviť parser, ktorý informácie z týchto konfiguračných súborov vyhodnocuje. Mnou navrhnuté riešenie počíta s jedným konfiguračným súborom pre každý testovací systém. Takýmto spôsobom môžeme pomocou použitých prepínačov jednoducho meniť počet testovacích systémov, na ktoré sa budú distribuovať testy. Pri spúšťaní regresných testov na troch systémoch je preto potrebné mať pripravené 3 konfiguračné súbory, každý pre jeden testovací systém. Spoločné informácie v každom konfiguračnom súbore je možné uložiť do samostatného súboru. V každom z použitých konfiguračných súborov potom môžeme takýto súbor zahrnúť pomocou známej direktívy `#include` známej z ostatných programovacích jazykov.

```
Product: MCO
Username: root
Host: 10.10.10.206
Outerhost: 10.10.110.206
Version: 2.3-05.04
```

Príklad 3.1: Konfiguračný súbor pre produkt MCO

Nová funkcionalita, ktorú som navrhol pre pridanie do rozšíreného plánovača testov, je možnosť sledovať aktuálny stav testovania pomocou progress baru. Táto funkcionalita v starej verzii plánovača testov chýbala a bolo preto ťažké sledovať aktuálny stav testovania. Vo vytvorenom rozšírení plánovača testov by sa mal zobrazovať aktuálny stav testovania pre každý testovací systém zvlášť, ako aj celkový stav testov zo všetkých testovacích systémov.

3.3 Komunikácia medzi jednotlivými plánovačmi testov

V distribuovaných systémoch nie je spoločná pamäť, a preto je forma komunikácie založená na princípe zasielania správ. Princíp komunikácie je taký, že hlavný proces, ktorý spustí m plánovačov testov vo forme podprocesov, čaká na ukončenie všetkých týchto podprocesov. Ukončenie podprocesu znamená, že ak nenastala žiadna systémová chyba, tak je testovanie jednej podmnožiny testov dokončené. Ak by náhodou nastala nejaká kritická chyba, podproces by sa mohol ukončiť, a my by sme mohli prísť o výsledky z tejto podmnožiny regresných testov. Z tohto dôvodu by sa výsledky z testovania mali posielať ihneď, ako ich máme k dispozícii, a nemalo by sa čakať na dokončenie testovania danej podmnožiny re-

gresných testov. Každý podproces teda asynchrónne zasiela údaje o výsledkoch aktuálneho testu hlavnému procesu, ktorý si tieto výsledky uchováva. Tento prístup má výhodu, že sa výsledky testovania uchovávajú na dvoch miestach, a v prípade zlyhania systému, či už hlavného procesu alebo podprocesov, by sme sa k daným výsledkom vedeli dopracovať.

3.4 Interpretácia výsledkov

Medzi jednu z najdôležitejších vlastností, ktoré musí každý testovací nástroj mať, je vlastnosť zobrazovania výsledkov z testovania. Pri návrhu rozšírenia plánovača testov bolo nutné riešiť situáciu, ako zobraziť výsledky z každého testovacieho systému jednotlivo, a takisto aj súhrn celkových výsledkov.

Rozšírenie plánovača testov disponuje niekoľkými výsledkami, ktorými môže každý test skončiť. Jednotlivé výsledky platia len v prípade, že sa daný test spúšťal na všetkých testovacích systémoch len raz. Sú to výsledky:

- **passed** – test, ktorý sa skončil úspechom (návratová hodnota je rovná nule).
- **failed** – test, ktorý sa ukončil neúspechom (nenulová návratová hodnota).
- **skipped** – test, ktorý sa vďaka neúspechu nejakého predošlého testu vynechal, nakoľko by jeho spustenie skončilo neúspechom.
- **passed on second attempt** – cluster, ktorý sa vďaka prepínaču plánovača **-tw** spustil po neúspechu znova, pričom na druhýkrát skončil úspechom. Test označený týmto výsledkom je zároveň označený výsledkom *passed*.
- **known bug - failed** – cluster, ktorý je pre aktuálnu verziu softvéru označený ako *known bug*, ktorý skončil úspechom. Cluster označený týmto výsledkom, nie je označený výsledkom *failed*.
- **known bug - passed** – cluster, ktorý je pre aktuálnu verziu softvéru označený ako *known bug*, ktorý skončil neúspechom. Cluster označený týmto výsledkom je zároveň označený aj výsledkom *passed*.

V prípade, že sa nejaký test spúšťal na viacerých systémoch, bolo nutné riešiť stav, v ktorom sa test mohol ocitnúť. Problémom je situácia, kedy nejaký test, ktorý sa spúšťal na viacerých systémoch, skončil na týchto rozdielnych systémoch s rôznymi výsledkami. Pre riešenie tejto situácie bol zavedený stav **unclear**. Ak test skončí s rôznymi výsledkami, jeho finálny výsledok sa určí podľa prevodovej tabuľky 3.4. Stavy **passed on second attempt**, **known bug - failed** a **known bug - passed** nie sú zaznačené z dôvodu, že tieto špeciálne stavy sa mapujú na stavy **passed** a **failed**.

Spomínané stavy výsledkov testov platia pre celkové výsledky zo všetkých testovacích systémov. Rozšírený plánovač testov navyše umožňuje zobraziť výsledky všetkých testov pre každý testovací systém jednotlivo. Výsledky všetkých testov sa zobrazujú v rovnakom poradí, v akom sa testy spúšťali. Výsledky zobrazované z každého testovacieho systému majú však pre jednoduchosť len stavy **passed**, **failed** a **skipped**.

3.5 Riešenie nových typov problémov

Pri vytváraní rozšírenia plánovača je potrebné myslieť na nové problémy, ktoré sa objavili až pri riešení distribúcie testov. Jedným z problémov je zobrazovanie výsledkov testov,

výsledok 1	výsledok 2	konečný výsledok
passed	passed	passed
failed	failed	failed
skipped	skipped	skipped
passed	skipped	passed
failed	skipped	failed
passed	failed	unclear
unclear	*	unclear

Tabuľka 3.4: Mapovanie rozdielnych výsledkov testu na konečný výsledok

spomínané v predchádzajúcej kapitole. Pre riešenie tohto problému bol zavedený výsledok testu *unclear*.

Nezávislosť medzi testami

Ďalším z problémov, ktorého riešenie bolo treba navrhnúť, bol problém, ktorý porušoval vlastnosť nezávislosti medzi testami a vlastnosť integrity testovacej sady, popísaných v dokumente [8]. Jednalo sa o problém, že niektoré testy používali pevne stanovený sieťový port, ktorý slúžil napríklad pre komunikáciu so simulátorom, alebo pre napojenie sa na nejakú časť testovaného systému, atď. Problémom tohto pevného portu bola distribúcia testov, ktoré používali rovnaký port. V prípade, že sa takéto testy spúšťali naraz, dochádzalo k obsadeniu portu prvým testom, pričom druhý test už k portu nemal prístup, a tak končil neúspechom.

Riešením tohto problému je parameter port offset **-po**. Každý test, ktorý používal pevne stanovenú hodnotu portu sa musel upraviť tak, aby sa ako port použila táto pevná hodnota, pripočítaná o hodnotu zadanú parametrom **-po**. Jednotlivé podprocesy plánovača testov sa spúšťajú s parametrom **-po**, ktorý sa navyšuje práve o hodnotu **-po** s každým podprocesom. V praxi to znamená, že ak nastavíme hodnotu offsetu na 1000, tak sa prvý podproces spustí s parametrom **-po 1000**, druhý podproces sa spustí s parametrom **-po 2000**, tretí s **-po 3000** atď. V teste sa potom používa štandardné číslo portu, ktorý sa využíval v teste pred úpravou, s pridanou hodnotou port offsetu. Týmto spôsobom sa zabraňuje problému, že niektoré testy pristupovali k rovnakému portu súčasne. Implicitnou hodnotou parametru **-po**, ktorá sa používa napríklad pri spustení plánovača testov bez potreby distribúcie testov je hodnota 0.

Systém logovania

Ďalším novým problémom sa stalo aj spravovanie logovacích súborov. Plánovač testov totiž pri vykonávaní testov zapisuje informácie o aktuálne prevádzaných testoch do viacerých logovacích súborov. Pri distribúcii testov sa však spúšťajú viaceré podprocesy plánovača testov, pričom každý podproces si tieto súbory vytvára zvlášť. V rámci centrálnej správy bolo nutné navrhnúť prístup, akým sa budú logovať testy z viacerých strojov na jedno centrálne miesto. Pre riešenie tohto problému bolo možné implementovať 2 možné spôsoby:

- **Zapisovanie údajov do logov po každom teste** – týmto spôsobom by sa do logovacích súborov, vytváraných hlavným procesom, postupne zapisovali informácie, vždy po aktuálne dokončenom teste, nezávisle na testovacom systéme. Tento prístup je

však nevyhovujúci, lebo by podprocesy museli posielať logovacie informácie hlavnému procesu, čo je náročné na zdroje. Ďalšou nevýhodou je to, že by sa poradie jednotlivých testov z testovacích systémov pomiešalo.

- **Parsovanie údajov do logov po dokončení testovania** – toto riešenie spočíva v tom, že sa najprv počká na dokončenie testovania. V momente, ak je testovanie ukončené, môžeme počítať so situáciou, že si každý podproces plánovača testov vytvoril vlastné logovacie súbory, v ktorých je zachované poradie testov. Ak máme tieto súbory k dispozícii, môžeme si z nich vytiahnuť potrebné informácie a skopírovať ich do logovacích súborov, vytvorených rozšíreným plánovačom testov. Toto riešenie má výhodu, že je menej náročné na zdroje, a zároveň, že sa v logoch dodržiava poradie testov, vykonávaných pre každý systém zvlášť. Pri implementácii rozšírenia plánovača testov sa využíva práve toto riešenie.

Neželaná distribúcia testov

V regresnej sade existujú množiny testov, ktoré chceme spúšťať vždy ako celok. Príkladom takejto množiny testov je funkcionalita *env*, ktorá slúži na nainštalovanie rôznych produktov. Táto funkcionalita obsahuje niekoľko clusterov, ktoré tvoria jeden funkčný celok slúžiaci na preinštalovanie produktu. V prípade produktu MCO táto funkcionalita obsahuje clustre, ktoré:

1. vymažú zo systému produkt MCO
2. stiahnu verziu produktu MCO zadanú v konfiguračnom súbore
3. nainštalujú stiahnutú verziu produktu
4. nakonfigurujú systém
5. nainštalujú externé súčasti systému

V prípade regresných testov sa pred samotným testovaním vždy vykoná táto funkcionalita, aby sa zaistilo, že sa bude testovať najnovšia verzia produktu, so správnou konfiguráciou. Pri využívaní viacerých systémov pre distribúciu testov je nutné na každom takomto systéme preinštalovať testovaný produkt. Pre tento účel bol naimplementovaný parameter **-wd**[1], ktorý zapríčiní, že sa testy nebudú distribuovať, ale po naplánovaní sa všetky spustia na každom testovacom systéme. Príkaz pre nainštalovanie produktu MCO na troch testovacích systémoch je uvedený v príklade 3.2.

```
meszarosf@mco-v156#./Run_test.tcl -i cfg.ini -mi cfg-vm1.ini cfg-vm2.ini
   cfg-vm3.ini -f env -wd
```

Príklad 3.2: Inštalácia produktu MCO na troch testovacích systémoch súčasne

[1] skratka pre angl. without distribution

Kapitola 4

Implementácia

Kapitola popisuje základné implementačné detaily jednotlivých častí rozšírenia pre plánovač testov. Prvá kapitola popisuje, aké technológie boli pre implementáciu rozšírenia plánovača použité. Nakoľko sa jedná o implementáciu rozšírenia pre nástroj, ktorý už bol vytvorený, kapitola taktiež popisuje rozdelenie zdrojového kódu na časť vytvorenú firmou Acision, a časť vytvorenú mnou. Kapitola obsahuje popis implementácie distribúcie testov so zameraním na rozloženie záťaže a popis toho, a akou formou sa spúšťajú testy na jednotlivých testovacích systémoch. Ďalej kapitola popisuje implementáciu komunikácie medzi jednotlivými procesmi plánovača testov. Posledná kapitola popisuje implementáciu novej funkcionality plánovača, ktorou je progress bar.

4.1 Použité technológie a členenie práce

Rozšírenie plánovača testov je napísané v jazyku Tcl/Tk (Tool Command Language), pre verziu 8.4.19. Pre správny beh aplikácie je taktiež nutné mať nainštalované rozšírenie tohoto jazyka zvané *Tclx*[1]. Ďalej potrebuje plánovač testov pre správne fungovanie interaktívneho režimu, ktorý sa však v regresných testoch nepoužíva, rozšírenie *tclreadline*[2]. Plánovač testov je vo firme známy pod názvom TTT (Tcl Testing Tool).

Testovacia sada pre produkty je delená na 2 časti, pričom každá časť sa udržiava v samostatnom repozitáry vo verzovacom systéme. Jedná sa o časti:

- **ttt-core** – Je logická časť plánovača testov, zodpovedná predovšetkým za plánovanie, zbieranie výsledkov, spúšťanie testov, vytváranie logov a pod. Táto časť je spoločná pre všetky produkty.
- **ttt-product** – Je časť plánovača testov, ktorá obsahuje všetky testy, simulátory a nástroje potrebné pre otestovanie špecifického produktu. Nakoľko sa plánovač testov využíva pre testovanie širokej škály produktov, každý produkt má vlastný repozitár pre svoje testy, ktorý si udržiava oddelene.

Pri vytváraní rozšírenia pre plánovač testov som zasahoval hlavne do časti *ttt-core*, ktorá je spoločná pre každý produkt. Táto časť je rozdelená na niekoľko súborov. Pri implementácii som všetky mnou vytvorené funkcie a zdrojový kód vkladal do mnou vytvoreného súboru **prlmng.tcl**. Tento súbor obsahuje všetky funkcie potrebné pre správnu funkcionalitu rozšírenia. Pri vytváraní rozšírenia sa však niektoré časti kódu museli upraviť. Jednalo

[1] dostupný na stránke http://tclx.sourceforge.net/
[2] http://tclreadline.sourceforge.net/

sa hlavne o hlavný súbor plánovačky testov, súbor **Run_test.tcl** a o súbor zodpovedný pre plánovanie testov, súbor **tstmng.tcl**. Zmeny prevedené v týchto dvoch hlavných súboroch sú zobrazené v prílohe C.

Nakoľko firma Acision vyvíja komerčný softvér, pri odovzdávaní zdrojových kódov som pre ochranu duševného vlastníctva firmy Acision odovzdal len súbory nutné pre beh plánovača testov. Pri odovzdávaní som dodržal logickú štruktúru testov, ktoré sa v produkte MCO používajú, avšak zdrojové kódy testov som musel nahradiť jednoduchou funkciou, ktorá počká náhodný čas a potom ukončí test úspechom. Odovzdané súbory obsahujú štatistický súbor *TTT_stat.txt*, ktorý uchováva dĺžku vykonávania každého testu spúšťaného v regresnej sade produktu MCO. Tento súbor obsahuje presné informácie o dĺžke behu testov v tomto produkte a je použitý pre porovnanie výsledkov. Ďalej je medzi odovzdanými súbormi súbor *KNOWN.BUGS* z produktu MCO. Tento súbor obsahuje testy ohraničené verziami, v ktorých sa vyskytujú odhalené chyby, ktoré nemusia priamo súvisieť s funkcionalitou, ktorá sa v danom teste testuje. Viac sa o tomto súbore píše v kapitole 2.4. Z časti *ttt-core* som odovzdal len súbory nutné pre beh plánovača.

4.2 Distribúcia testov

Pri implementovaní distribúcie testov som použil distribúciu clusterov na základe spoločných prerekvizít, popísanú v kapitole 3.1. Implementácia distribúcie testov je obsiahnutá vo funkcii `prl_divide_test_groups`.

Táto funkcia sekvenčne prejde všetky testy a rozdelí ich do skupín, kde každý cluster v rovnakej skupine má rovnakú množinu prerekvizít. Pre každú takúto skupinu sa potom spočíta odhadovaná dĺžka trvania behu tejto skupiny. Tento čas sa ráta na základe poslednej dĺžky trvania behu daného clusteru, zaznamenanom v súbore *TTT_stat.txt*. Tieto skupiny sa potom zoradia podľa odhadovaných dĺžok trvania celej skupiny zostupne. V cykle sa potom prechádzajú tieto skupiny od najväčšieho odhadovaného času behu po najmenší a postupne sa priraďujú do jedného z m zoznamov spúšťaných clusterov na distribuovaných systémoch. Daná skupina clusterov sa priradí vždy do toho zoznamu spúšťaných clusterov, ktorý má najmenšiu odhadovanú dĺžku trvania behu. Týmto spôsobom sa zaisťuje rozloženie záťaže.

Zoznam clusterov pre každý distribuovaný systém sa potom vloží do samostatného súboru. Pri spustení plánovača testov na jednotlivých distribuovaných systémoch si potom každý plánovač zoberie jeden takýto súbor. Z tohto súboru si vyčíta zoznam clusterov, ktoré sa majú spustiť. Tento zoznam clusterov sa potom nanovo preplánuje a začne postupne vykonávať.

Preplánovanie testov je nutné z rôznych dôvodov. Jedným z nich je, že do zoznamu clusterov sa musia pridať *Preopty* kvôli splneniu požiadavku na zachovanie funkcionality, spomínaného na konci kapitoly 3.1. Ďalším dôvodom je, že rozšírenie plánovača využíva distribúciu testov na základe spoločných prerekvizít. Vďaka tomuto distribuovaniu sa pre jeden systém môžu zvoliť clustre z rôznych častí test plánu, ktorý by vznikol bez distribúcie testov. Tieto časti teda treba vhodne „prepojiť“ pomocou prerekvizít a odrekvizít.

Ďalším z dôvodov, prečo sa clustre musia preplánovať je fakt, že plánovač testov neobsahuje funkcionalitu, ktorou by sme mohli spustiť už predpripravený plán testov. Táto funkcionalita by sa musela naimplementovať. Regresné testovanie trvá približne 15 hodín a vytvorenie test plánu pre takéto testovanie trvá menej ako minútu. Na základe tohto faktu môžeme čas potrebný na vytvorenie test plánu považovať za zanedbateľný a môžeme si dovoliť testy takýmto spôsobom preplánovávať.

Posledným a nemenej dôležitým dôvodom, prečo sa preplánovanie testov používa je fakt, že plánovač testov sa vo firme Acision každodenne používa už niekoľko rokov. Jeho funkcionalita je teda rokmi overená, a preto sa môžeme spoľahnúť, že sa všetky testy úspešne vykonajú a že sa žiadny nevynechá. V prípade implementovania funkcionality, ktorá by zmenila logiku plánovania tak, aby preplánovanie testov nebolo potrebné, by sme narazili na problém, že by sme túto funkcionalitu museli veľmi dôkladne otestovať.

Algoritmus distribúcie testov je možné vylepšiť o rôzne druhy optimalizácii a heuristiky. Vylepšením implementácie tohoto algoritmu sa venuje kapitola 5.

4.3 Spúšťanie a komunikácia s podprocesmi

Pre každý testovací systém sa musí spustiť nový podproces plánovača testov, ktorý sa pomocou IP adresy napojí na testovací systém, na ktorom sa bude testovať daný softvér. Hlavný proces plánovača testov potom slúži ako centralizovaná správa testov uvedená v kapitole 3.2. Pri implementácii spúšťania podprocesov som využil procedúru `BgExec`[3], ktorá dokáže spustiť podproces na pozadí. Procedúra taktiež umožňuje udržiavať počet aktuálne bežiacich podprocesov na pozadí. Takýmto spôsobom je možné jednoducho kontrolovať počet bežiacich regresných testov na všetkých testovacích systémoch.

Pre nastavenie počtu systémov, na ktoré chceme distribuovať regresné testy, vznikol nový parameter **-mi**. Za tento parameter sa uvádza zoznam konfiguračných súborov, ktoré sa použijú pre spustenie testov na rôznych testovacích systémoch. Počet konfiguračných súborov udáva počet systémov, na ktoré sa budú testy distribuovať.

Aby sa zabránilo situácii, že by jednotlivé konfiguračné súbory používali inú verziu produktu, ktorá sa používa hlavne pri vyberaní správnych testov, plánovač vyžaduje taktiež zadanie konfiguračného súboru, ktorý sa použije pre výber testov. Tento konfiguračný súbor sa zadáva rovnako ako v plánovači testov bez rozšírenia o možnosti spúšťať testy na distribuovaných systémoch, a to parametrom **-i**. Spustenie regresných testov v produkte MCO na troch distribuovaných systémoch je uvedené na príklade 4.1.

```
meszarosf@mco-v156#./Run_test.tcl -i cfg.ini -mi cfg-vm1.ini cfg-vm2.ini
   cfg-vm3.ini -B TEST.REVIEW_23 -f all -tw -po 1000
```

Príklad 4.1: Príkaz pre distribúciu regresných testov v produkte MCO na 3 testovacie systémy

Pred spustením podprocesov plánovača testov sa najprv zistia parametre, s ktorými sa dané inštancie spustia. Z týchto parametrov sa vylúčia tie, ktoré nie sú vhodné pre distribúciu testov. Jedná sa napríklad o parametre určené pre interaktívny režim spomínaný v kapitole 2.4.

Plánovač testov potom rozdelí testy na m častí, zadaných pomocou prepínača **-mi**. Zoznam všetkých clusterov, určených pre každý testovací systém sa potom vloží do samostatných súborov. Po spustení podprocesov plánovača testov si potom každý podproces zoberie jeden súbor so zoznamom clusterov na otestovanie a tieto clustre naplánuje a otestuje.

Spúšťanie podprocesov je implementované vo funkcii `prl_run_multiple_schedulers`. Po spustení všetkých potrebných podprocesov na pozadí sa vo funkcii hlavného procesu `prl_accept_conn` čaká na dáta od podprocesov. Podprocesy potom asynchrónne zasielajú

[3]dostupná zo stránky http://wiki.tcl.tk/12704

výsledky svojich testov a takisto aj plán testov hlavnému procesu. Každý podproces komunikuje len s hlavným procesom. Jednotlivé podprocesy o sebe navzájom nevedia, nakoľko sa každý podproces správa ako autonómny systém.

Pre komunikáciu s podprocesmi si hlavný proces vyčlení jeden voľný systémový port, na ktorom bude očakávať výsledky z podprocesov. Zisťovanie voľného portu a inicializácia čakania na výsledky od podprocesov je implementované vo funkcii `prl_server_start`. Hlavný proces plánovača testov predá podprocesom číslo portu tak, že podprocesy spustí s parametrom **-lp**[4] so zadaným číslom portu.

Ako prvé sa čaká na test plány od každého podprocesu. Po prijatí týchto test plánov sa vypočítajú informácie o odhade času potrebného pre testovanie, počtu prerekvizít, počtu clusterov a iné. Medzi týmito informáciami je uvedené napríklad aj percentuálne vyjadrenie počtu clusterov alebo prerekvizít, ktoré sa spúšťajú viacnásobne. Počítanie tohto percentuálneho vyjadrenia je naimplementované vo funkcii `prl_count_test_overlapping`. Zároveň sa uvádza odhadovaný čas, ktorý v prípade distribúcie testov na aktuálny počet testovacích systémov ušetríme. Ukážka takýchto informácii je uvedená v príklade 4.2.

```
status: ############################################################
status: #                   Number of TCS: 1487                    #
status: #                  Number of Prereq: 656                   #
status: #  Expected execution time of the node 1: 0 d 09:05:53     #
status: #  Expected execution time of the node 2: 0 d 08:25:29     #
status: #  Expected execution time of the node 3: 0 d 08:59:47     #
status: #             Cluster overlapping ratio: 2.3%              #
status: #               Test overlapping ratio: 3.1%               #
status: #          Expected execution time: 0 d 09:05:53           #
status: #           Approximate time saved: 0 d 15:43:49           #
status: #                   Plan has 2799 nodes.                   #
status: #                Planning lasts: 0 d 00:00:15              #
status: ############################################################
```

Príklad 4.2: Informácie zobrazované po naplánovaní regresných testoch

Tieto informácie sú vypisované vo funkcii `prl_plan_stat` a ich význam je nasledovný:

- **Number of TCS** – celkový počet naplánovaných clusterov.
- **Number of Prereq** – celkový počet naplánovaných prerekvizít.
- **Expected execution time of the node x** – odhadovaná dĺžka trvania testovania pre každý testovací systém na základe štatistického súboru *TTT_stat.txt*.
- **Cluster overlapping ratio** – percentuálne vyjadrenie počtu clusterov, ktoré sa spustia vďaka distribúcii testov viacnásobne.
- **Test overlapping ratio** – percentuálne vyjadrenie počtu clusterov, prerekvizít a odrekvizít, ktoré sa spustia vďaka distribúcii testov viacnásobne.
- **Expected execution time** – odhadovaná dĺžka trvania celého testovania, zistená z maximálnej odhadovanej dĺžky trvania testovania pre každý testovací systém.
- **Approximate time saved** – odhadovaná ušetrená dĺžka trvania testovania v porovnaní s prípadom, ak by sme nevyužili vlastnosti distribúcie testov, a všetky testy by sme spúšťali na jednom testovacom systéme.

[4] angl. skratka pre listening port

- **Plan has x nodes** – celkový počet naplánovaných testov.
- **Planning lasts** – dĺžka trvania plánovania.

Po prijatí všetkých test plánov sa spustí vykonávanie testov. Pre vypísanie test plánov a nespustenie testovania slúži parameter **-p**, ktorý je možné využiť pre rýchle zistenie informácii o plánovaných regresných testoch.

4.4 Sledovanie aktuálneho stavu testovania

Pri spúšťaní veľkého množstva testov bolo v plánovači testov problematické zisťovať, v akej fáze testovania sa práve nachádzame. Pred samotným ukončením testovania nebolo možné napríklad zistiť počet clusterov, ktoré skončili úspechom či neúspechom. Jediný údaj, ktorý bolo možné zistiť, bolo aktuálne poradie testu zo všetkých možných testov. Pri implementovaní rozšírenia pre plánovač testov som sa rozhodol tento prístup vylepšiť formou progress baru.

Progress bar zobrazuje informácie z jednotlivých testovacích systémov a umožňuje tak zlepšiť prehľad o aktuálnom stave testovania. Progress bar si zisťuje šírku terminálu (prípadne to, či máme vôbec terminál k dispozícii, čo neplatí napríklad v prípade použitia utility *cron*[5]) a na základe tejto šírky si upravuje svoju vlastnú veľkosť. Je naimplementovaný tak, aby vždy využíval celú šírku terminálu, a tým umožňoval zobrazovať výsledky, ako napríklad grafické znázornenie podielu dokončených testov čo najpresnejšie. Progress bar je schopný prispôsobovať sa zmene šírky terminálu pri behu testovania.

Minimálna šírka terminálu pre podporu progress baru je 80 znakov. Aktualizácia progress baru prebieha vždy po dokončení určitého testu a prijatia výsledku tohoto testu od podprocesu vo funkcii `prl_accept_conn`. Po obdržaní všetkých plánov testov sa najprv vo funkcii `prl_init_progress_bar` zistia všetky dostupné informácie o vytváranom progress bare, aby sme tieto údaje nemuseli každým prijatým výsledkom testu znova prepočítavať. Aktualizácia progress baru je naimplementovaná vo funkcii `prl_update_progress_bar`.

Progress bar s minimálnou šírkou terminálu a rôznymi typmi výsledkov je znázornený na príklade 4.3.

```
status: ##################################
status: Running parallel testing on 3 nodes
status: ##################################
status:
status: pid of the main process      : 1882
status: pids of the all subprocesses : 1891 1892 1893
status:
status:   Total:  65.59%  |  1836 tests of 2799  |  Time elapsed: 00:02:29
status:
status: Node 1:   2.3% []  |    23 tests of  986
status: Node 2: 100.0% []  |   849 tests of  849  |  Finished in: 00:01:08
status: Node 3: 100.0% []  |   964 tests of  964  |  Finished in: 00:00:56
status:
status: Number of PASSED  test clusters: 0 of 1487
status: Number of FAILED  test clusters: 7 of 1487
status: Number of SKIPPED test clusters: 1007 of 1487
```

Príklad 4.3: Ukážka progress baru so šírkou terminálu 80 znakov

[5]https://help.ubuntu.com/community/CronHowto

4.5 Zbieranie a vyhodnocovanie výsledkov

Zbieranie výsledkov funguje na princípe zasielania správ. Každý podproces zasiela hlavnému procesu (ktorý slúži ako centralizovaná správa testov) výsledky o každom dokončenom teste. Jednotlivé výsledky sa pre každý test zaznamenajú. V praxi sa stáva, že veľká časť prerekvizít a odrekvizít je spúšťaná viacnásobne. Navyše pri dopĺňaní *Preoptov* ako je uvedené v kapitole 3.1 sa stáva, že sa viacnásobne spúšťajú aj niektoré clustre.

Pre každý test sa ukladá množina jeho všetkých výsledkov. Po dokončení testovania sa potom z tejto množiny na základe prevodovej tabuľky 3.4 určí konečný výsledok, ktorý sa zobrazí v štatistikách o prebehnutom testovaní. Určovanie konečného výsledku je implementované vo funkcii `prl_determine_test_result`.

Ak sa nejaký podproces nečakane ukončí, automaticky prichádzame o časť výsledkov. Používateľ musí byť o tomto fakte informovaný, nakoľko sa jedná o kritickú chybu. Pre detekciu tohto problému pred samotným začatím testovania každý podproces zasiela hlavnému procesu plán testov.

Pri detekcii, že sa daný podproces ukončil (či už plánovane alebo nečakane v prípade nejakej chyby) sa porovnajú všetky prijaté výsledky testov s plánom, ktorý sa posielal na začiatku. V prípade, že podproces zaslal menej výsledkov, ako mal v test pláne, používateľ je o tejto situácii informovaný formou chybového výpisu.

4.6 Parsovanie logovacích informácii

V priebehu testovania si každý podproces plánovača testov vytvára vlastné logovacie súbory, do ktorých si ukladá informácie o testoch. Samotný hlavný proces však žiadne testy nespúšťa, a preto v ňom tieto informácie chýbajú. Pre každého testera sú však logovacie informácie dôležitou súčasťou jeho každodennej práce. Navyše sa logovacie súbory používajú aj pri vytváraní dokumentácie zvanej ATP[6] ktorý je súčasťou akceptačného testovania spomínaného v kapitole 2.

Navrhnuté riešenie sa snaží tento problém odstrániť tým, že sa vytvorené logovacie súbory po dokončení testovania skopírujú na jedno centrálne miesto. Princíp spájania logovacích súborov je ten, že sa postupne začnú prechádzať všetky logovacie súbory vytvorené podprocesmi. V každom súbore sa preskočí úvodná časť a prejde sa k časti, ktorá obsahuje výsledky z testov. Tieto výsledky sa potom nakopírujú do logovacieho súboru vytvoreného hlavným procesom. Jednotlivé časti z rôznych podprocesov sa pomocou komentára označia, aby sa vedelo určiť kde jednotlivé logovacie informácie z každého podprocesu začínajú a končia.

Implementácia parsovania logovacích súborov je obsiahnutá v dvoch funkciách, a to `prl_parse_error_logs` a `prl_parse_logs`.

[6] angl. Acceptance Test Protocol

Kapitola 5

Optimalizácie algoritmu distribúcie testov

V nasledujúcej kapitole si popíšeme optimalizácie algoritmu distribúcie testov na základe spoločných prerekvizít, ktorý je v rozšírení plánovača testov naimplementovaný. Jednotlivé optimalizácie boli postupne naimplementované do funkcie `prl_divide_test_groups`, ktorá slúži na distribúciu testov. Tieto optimalizácie na seba naväzujú a je možné ich jednoducho zapínať a vypínať pomocou premennej `g_prl_optimize`. Nastavenie tejto premennej je umiestnené na začiatku súboru *prlmng.tcl*, ktorý obsahuje všetky nové funkcie naimplementované pre potreby rozšírenia plánovača testov. Pre optimalizácie je využívaný štatistický súbor *TTT_stat.txt* spomínaný v predchádzajúcich kapitolách. Plánovač testov je schopný využívať svoj posledný beh testov a zbierať štatistiky tak, aby bolo možné zlepšiť rozloženie záťaže v nasledujúcom testovaní.

5.1 Optimalizácia 1 – distribúcia využitím dĺžky trvania behu prerekvizít a odrekvizít

Základný algoritmus pre distribúciu testov využíva len dĺžky trvaní jednotlivých clusterov. Rozloženie záťaže medzi jednotlivé testovacie systémy je zabezpečované iba štatistikami, zozbieranými zo spúšťania clusterov. Do plánu testov sa však pridávajú aj prerekvizity a odrekvizity, a to môže narušiť optimálne rozloženie záťaže.

Prvá optimalizácia algoritmu distribúcie testov sa snaží minimalizovať tento problém tak, že sú testy distribuované aj na základe dĺžok trvania prerekvizít a odrekvizít. Zapnutie tejto optimalizácie je možné nastavením premennej `g_prl_optimize` na hodnotu väčšiu alebo rovnú 1.

Distribúcia testov na m častí zadaných pomocou parametra **-mi** pracuje tak, že sa pri priraďovaní skupiny clusterov s rovnakými prerekvizitami do jednej z m častí počíta odhadovaná dĺžka trvania danej časti. Tento čas sa však nepočíta len na základe clusterov, ale do tohto času sú prirátané aj jednotlivé dĺžky trvania prerekvizít a odrekvizít pre konkrétnu skupinu testov.

Všetkých m častí pre každý testovací systém si udržiava zoznam a dĺžky trvania jednotlivých testov. Tieto časy sa spočítajú a pomocou tejto hodnoty sa riadi rozloženie záťaže pre každý testovací systém.

Hlavnou nedokonalosťou tohto prístupu je, že skupiny clusterov obsahujú navzájom podobné množiny prerekvizít. Pri priraďovaní takejto skupiny clusterov do jednej z m častí

by sa teda väčšina prerekvizít a odrekvizít pridala viacnásobne. Toto je však neželaný efekt, nakoľko plánovač testov disponuje vlastnosťou združovania testov, popísanou v kapitole 2.4.

Vďaka tejto vlastnosti sa prerekvizity pri vytváraní plánu testov združujú, a preto je viacnásobné pridanie prerekvizít a odrekvizít do spomínaných častí neželané. Pre čiastočné riešenie tohto problému sa teda prerekvizity a odrekvizity pridávajú do každej časti len raz.

Rozloženie záťaže je teda v konečnom výsledku pre každý testovací systém riešené tak, že sa odhaduje čas vykonávania každej časti regresných testov pomocou času vykonávania všetkých clusterov v danej časti. Do tohto času vykonávania sa však ešte pridajú časy vykonávania prerekvizít a odrekvizít pre dané clustre, avšak časy trvania prerekvizít a odrekvizít sa pripočítava len raz.

Takýmto spôsobom môžeme pomerne presne určiť dĺžku trvania testovania pre každý testovací systém zvlášť. Odchýlkami tejto vypočítanej hodnoty trvania testovania oproti skutočnému trvaniu testovaniu po preplánovaní sú len prerekvizity a odrekvizity, ktoré sa spúšťajú viacnásobne, a samozrejme pridané *Preopty*.

5.2 Optimalizácia 2 – distribúcia využitím rozdeľovania skupín clusterov

Rozšírenie plánovača testov pre podporu distribúcie testov využíva distribúciu clusterov na základe spoločných prerekvizít spomínaných v kapitole 3.1. Na začiatku distribúcie testov sa najprv vytvoria skupiny clusterov so spoločnými prerekvizitami a následne sa distribuujú priamo tieto skupiny.

Nevýhodou tohto prístupu je to, že v testovacej sade môžeme mať takéto skupiny, ktorých je málo a zároveň majú príliš veľký čas vykonávania. V prípade, že máme skupín testov menej, ako je počet testovacích systémov, niektoré testovacie systémy ostanú nevyužité, lebo sa im nepriradí žiadna z týchto skupín.

Najnevhodnejšou variantou testovacej sady pre základný algoritmus distribúcie testov je sada, ktorá obsahuje všetky clustre s rovnakou množinou prerekvizít. V prípade takejto sady sa všetky clustre zaradia do jednej spoločnej skupiny a túto skupinu by ďalej nebolo možné distribuovať. Príkladom takejto nevhodnej sady je testovacia sada znázornená na obrázku 5.1. V prípade takejto testovacej sady a bez použitia optimalizácie by sa takáto skupina clusterov vôbec nedistribuovala. Všetky clustre by sa spustili na jednom testovacom systéme a prípadné ostatné testovacie systémy by ostali nevyužité.

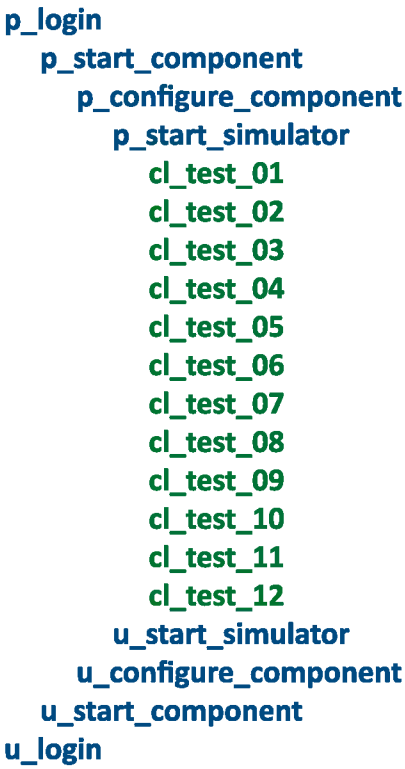
Riešením tohto problému je použitie optimalizácie, ktorá je schopná takúto veľkú skupinu clusterov rozdeliť na menšie. Táto optimalizácia funguje tak, že sa takéto veľké skupiny testov rozdelia na dve menšie. Algoritmus delí takéto veľké skupiny clusterov vždy na dve časti, aby sme v prípade delenia skupín na viac častí zbytočne nezanášali do jednotlivých testovacích systémov nové prerekvizity a odrekvizity. Algoritmus sa snaží odhadnúť, akú veľkú časť skupiny testov potrebujeme pre aktuálny testovací systém (testovací systém s najmenšou odhadovanou dĺžkou trvania testovania), do ktorého vkladáme nové clustre. Zvyšná časť clusterov sa ponechá na distribúciu do ostatných testovacích systémov.

Takýmto spôsobom je zaistené to, že sa v prípade úspechu algoritmu táto veľká skupina clusterov rozdelí tak, že sa aktuálnemu testovaciemu systému priradia clustre práve tak, aby sa tento testovací systém naplno využil s ohľadom na rozloženie záťaže. Následné pridávanie clusterov do tohto testovacieho systému by už nemalo byť potrebné.

Algoritmus rozdeľovania veľkých skupín clusterov na dve menšie je naimplementovaný vo funkcii `prl_split_test_groups`. Detekcia toho, či je nutné skupinu clusterov rozdeliť je

obsiahnutá vo funkcii `prl_divide_test_groups`, ktorá slúži na distribúciu testov.

Tento typ optimalizácie je zapnúť nastavením premennej `g_prl_optimize` na hodnotu väčšiu alebo rovnú 2. V prípade aplikovania tejto optimalizácie sa automaticky zapína aj optimalizácia s využitím dĺžky trvania prerekvizít a odrekvizít.



Obrázok 5.1: Príklad nevhodnej sady testov pre základný algoritmus distribúcie testov

5.3 Optimalizácia 3 – distribúcia využitím podobnosti prerekvizít

Táto optimalizácia využíva pri distribúcii testov skutočnosť, že čím viac skupín clusterov s podobnými prerekvizitami existuje v danom distribuovanom systéme, tým efektívnejšie vieme využiť vlastnosť združovania testov popísanej v kapitole 2.4. Pri rozhodovaní, do ktorej z m skupín testov zadaných pomocou prepínača **-mi** vložíme aktuálnu skupinu clusterov je teda vhodné rozhodovať sa aj na základe tohto faktora.

Táto optimalizácia funguje tak, že sa najprv zistia všetky možné testovacie systémy, do ktorých je možné vložiť skupinu clusterov bez toho, aby sa narušilo rozloženie záťaže na týchto systémoch. V prípade, že je takýchto systémov viac (čo sa v prípade širokej škály clusterov s rôznymi prerekvizitami bežne stáva), aktuálnu skupinu clusterov automaticky nepriradíme testovaciemu systému, ktorý má najmenšiu odhadovanú dĺžku trvania testovania. Namiesto toho si pre každý takýto testovací systém spočítame počet prerekvizít, ktoré má spoločné s aktuálnou skupinou clusterov, ktorú sa snažíme distribuovať.

Skupinu clusterov potom priradíme do toho testovacieho systému, ktorý má najviac podobných prerekvizít. V prípade, že existuje viac testovacích systémov, ktoré majú rovnaký

počet podobných prerekvizít, aktuálnu skupinu testov vložíme do testovacieho systému s najmenšou odhadovanou dĺžkou trvania testovania.

Počítanie podobnosti prerekvizít je jednoduchá operácia, ktorá nám vo výsledku zabezpečí, že sa na jednotlivých testovacích systémoch budú združovať clustre, ktoré majú medzi sebou podobné prerekvizity. Takéto clustre je potom možné efektívnejšie naplánovať a využiť tak vlastnosti združovania testov.

Zapínanie tejto optimalizácie je riadené nastavením premennej `g_prl_optimize` na hodnotu väčšiu alebo rovnú 3. V prípade využitia tejto optimalizácie sa zároveň aktivuje využívanie predošlých dvoch optimalizácii.

Kapitola 6

Zhodnotenie dosiahnutých výsledkov

V nasledujúcej kapitole si porovnáme vytvorené rozšírenie plánovača s jeho pôvodnou verziou, ktorá nedisponovala funkcionalitou distribúcie testov. Ukážeme si niektoré štatistiky, z ktorých bude zrejmý prínos vytvorenia rozšírenia pre tento nástroj. Jednotlivé štatistiky sú zbierané na základe informácii priamo z produktov vo firme Acision, kde sa tento plánovač testov s využitím distribúcie testov každodenne používa. V kapitole si na záver porovnáme vplyv jednotlivých naimplementovaných optimalizácii na regresné testovanie.

6.1 Prínos vytvorenia rozšírenia pre plánovač testov

Prínos vytvorenia rozšírenia pre distribúciu testov na viaceré testovacie systémy je zrejmý. Táto bakalárska práca vznikla z potreby, že sa každodenné regresné testovanie natiahlo na viac ako 15 hodín. Čas potrebný pre tento typ testovania sa navyše každým pridaným testom zvyšoval. Za posledný rok sa v produkte MCO zvýšil tento čas takmer o 2 hodiny.

Z dôvodu, že je regresné testovanie časovo náročná operácia, sa taktiež pri písaní nových testov bral na časovú náročnosť veľký ohľad. V prípade niektorých testov sa tým znížila kvalita testu, nakoľko sa tieto testy museli obmedziť, aby nevykonávali taký veľký počet operácii. Jednalo sa hlavne o niektoré základné typy stress testov.

Postupom času sa musel vymyslieť spôsob, akým by bolo možné skrátiť dĺžku trvania regresných testov. Tým spôsobom bola distribúcia testov.

Tabuľka 6.1 znázorňuje dĺžku trvania regresných testov pre produkty MCO a SMSCv5 využitím rôznych počtov testovacích systémov, na ktoré je možné testy distribuovať. V prípade použitia jedného testovacieho systému je vytvorený plán testov identický s plánom testov pred naimplementovaním rozšírenia pre podporu distribúcie testov. Údaje v tabuľke vznikli naplánovaním regresných testov, ktoré sa v spomínaných dvoch produktoch každodenne spúšťajú. Pri plánovaní testov boli využité všetky naimplementované optimalizácie.

Z tabuľky vyplýva, že použitím dvoch testovacích systémov skráti celkový čas potrebný pre dokončenie testovania na približne 54% z celkového času potrebného pre dokončenie testovania v prípade použitia jedného testovacieho systému. Táto hodnota nového času testovania nie je presná polovica, nakoľko sa pri vytváraní nových test plánov určitá množina prerekvizít spustí vždy. Príkladom takejto prerekvizity je prerekvizita na pripojenie sa na testovací systém, alebo na zapínanie nejakej často používanej komponenty.

V prípade využitia štyroch testovacích systémov efektivita distribúcie mierne klesá. Čas

počet testovacích systémov	čas potrebný pre dokončenie testovania v produkte MCO	čas potrebný pre dokončenie testovania v produkte SMSCv5
1	14:56:43	15:08:32
2	08:06:45	08:07:32
3	05:32:34	05:36:16
4	04:37:08	04:08:06
5	03:29:33	03:37:21
6	03:09:17	03:17:33
7	02:55:42	02:46:58
8	02:34:39	02:41:35
9	02:26:04	02:28:20
10	02:10:16	02:10:00

Tabuľka 6.1: Prínos použitia distribúcie testov pre rôzny počet testovacích systémov

potrebný pre dokončenie testovania tvorí v prípade produktu MCO približne 31% (namiesto ideálnych 25%) z času potrebného pre testovanie na jednom testovacom systéme. V prípade produktu SMSCv5 je táto percentuálna hodnota o trochu lepšia, a to niekde na úrovni 27%.

Ďalším pridávaním testovacích systémov sa efektivita distribúcie postupne znižuje. Je to zapríčinené hlavne prekrývaním clusterov a prerekvizít (hodnota test overlapping ratio použitá v príklade 4.2), čo znamená že sa tieto testy spúšťajú vďaka distribúcii testov viacnásobne. Plánovač testov je schopný túto hodnotu prekrývania testov pri naplánovaní testov vypočítať.

Grafické znázornenia tabuľky 6.1 pre produkty MCO a SMSCv5 sú umiestnené v prílohách E a F.

Pre zhodnotenie prínosu vytvorenia rozšírenia plánovača testov je nutné overiť, ako je zvládnuté rozloženie záťaže medzi jednotlivými testovacími systémami. Pri distribúcii testov na viaceré testovacie systémy je rozloženie záťaže vylepšované zbieraním štatistických informácii o poslednom behu testovania. Vo výsledku je rozloženie záťaže na jednotlivých testovacích systémoch rovnomerné, a pri použití niekoľkých testovacích systémoch sa minimálne a maximálne zaťaženie testovacích systémov líši rádovo v percentách. Dôkazom toho sú napríklad nasledujúce tabuľky.

Tabuľky 6.2 a 6.3 zobrazujú rozloženie záťaže v prípade použitia rôzneho počtu testovacích systémov. Údaje sú zozbierané z distribúcie regresných testov produktu MCO. V prípade použitia dvoch testovacích systémov je rozdiel dĺžky testovania na týchto systémoch prižne 17 minút. Tento čas tvorí približne 3,5% z maximálnej dĺžky testovania. V prípade použitia troch testovacích systémov je toto percento niekde na úrovni šiestich percent.

Zvyšné dve tabuľky zobrazujú rozloženie záťaže pri distribúcii testovacej sady pre produkt SMSCv5. Tabuľka 6.4 zobrazuje, že v prípade použitia dvoch testovacích systémov je rozdiel v dĺžke trvania testovania približne 28 minút, čo tvorí približne 6% z maximálnej dĺžky testovania. V prípade použitia štyroch testovacích systémov ako je zobrazených v tabuľke 6.5, je rozdiel medzi najkratším a najdlhším časom testovania približne 18 minút, čo tvorí približne 7% z maximálnej dĺžky testovania.

testovací systém	dĺžka testovania
1	08:06:45
2	07:49:38

Tabuľka 6.2: Rozloženie záťaže pre produkt MCO v prípade použitia 2 testovacích systémov

testovací systém	dĺžka testovania
1	05:14:40
2	05:11:28
3	05:32:34

Tabuľka 6.3: Rozloženie záťaže pre produkt MCO v prípade použitia 3 testovacích systémov

testovací systém	dĺžka testovania
1	08:07:32
2	07:39:21

Tabuľka 6.4: Rozloženie záťaže pre produkt SMSCv5 v prípade použitia 2 testovacích systémov

testovací systém	dĺžka testovania
1	04:06:11
2	03:49:51
3	03:58:52
4	04:08:06

Tabuľka 6.5: Rozloženie záťaže pre produkt SMSCv5 v prípade použitia 4 testovacích systémov

6.2 Porovnanie jednotlivých optimalizácii

Pri porovnávaní jednotlivých optimalizácii som vychádzal z plánovania testov regresnej sady, ktoré sa spúšťajú denne pre dva najväčšie produkty. Tabuľky 6.6 a 6.7 zobrazujú dĺžku trvania regresných testov pre rôzny počet testovacích systémov a rôzne úrovne optimalizácie. Z tabuliek vyplýva, že využívanie dĺžky trvania prerekvizít a odrekvizít ako je použité v optimalizácii 1, nie je vždy lepšie riešenie. Táto optimalizácia má najväčší efekt, keď sa prerekvizity spúšťajú maximálne raz. V prípade použitia veľkej sady testov sa však bežne niektoré prerekvizity spúšťajú niekoľkonásobne. V takomto prípade má optimalizácia len minimálny, alebo dokonca aj negatívny vplyv. Optimalizácia 2 funguje na princípe rozdeľovania veľkých skupín clusterov so spoločnými prerekvizitami. Táto optimalizácia nadobúda efekt iba v prípade, že je takéto skupiny nutné rozdeliť aby sme zachovali optimálne rozloženie záťaže. V prípade regresných testov sa však takéto veľké skupiny testov dokážu distribuovať aj bez toho, aby to narušilo rozloženie záťaže. Znova teda platí, že táto optimalizácia má efekt skôr na menší počet testov.

Z pohľadu regresného testovania má však veľký vplyv posledná optimalizácia. Táto optimalizácia sa snaží distribuovať clustre na testovacie systémy tak, aby sa na každom

počet testovacích systémov	Dĺžka regresného testovania			
	bez optimalizácii	optimalizácia 1	optimalizácia 2	optimalizácia 3
1	14:56:43	14:56:43	14:56:43	14:56:43
2	08:53:16	08:40:17	08:40:17	08:06:45
3	06:19:57	06:22:37	06:22:37	05:32:34
4	05:03:54	05:05:24	05:05:24	04:37:08
5	04:13:29	04:15:41	04:15:41	03:29:33
6	03:37:58	03:40:10	03:40:10	03:09:17
7	03:12:31	03:26:33	03:26:33	02:55:42
8	03:00:58	02:52:57	02:52:57	02:34:39
9	02:45:07	02:35:39	02:35:39	02:26:04
10	02:40:52	02:41:40	02:41:40	02:10:16

Tabuľka 6.6: Porovnanie jednotlivých optimalizácii pri plánovaní regresných testov v produkte MCO

testovacom systéme združovali clustre s podobnými prerekvizitami. Je to z dôvodu, aby plánovač testov mohol využiť svoju vlastnosť združovania testov. Z tabuliek vyplýva, že tento typ optimalizácie je schopný skrátiť čas potrebný na dokončenie regresného testovania aj o niekoľko desiatok minút.

Pre demonštrovanie optimalizácie 2 si uvedieme naplánovanie funkcionality *addrtran* pre produkt MCO. Táto funkcionalita pozostáva z niekoľko desiatok testov, z ktorých väčšina má rovnakú množinu prerekvizít. Testy v tejto funkcionalite majú len 2 množiny prerekvizít, takže všetky clustre sa rozdelia len do dvoch skupín testov. Tabuľka 6.8 zobrazuje dĺžky trvania testovania tejto funkcionality a efekt spomínanej optimalizácie. Pri plánovaní tejto funkcionality bol využitý parameter **-wp**, ktorý zapríčiní, že sa funkcionalita naplánuje bez použitia *Preoptov*.

Z tabuľky vyplýva, že v prípade nevyužitia optimalizácie sme schopní distribuovať iba skupiny clusterov s rovnakými prerekvizitami. Nakoľko máme takéto skupiny iba dve, sme schopní využiť iba dva testovacie systémy. V prípade využitia optimalizácie sme schopní rozdeľovať skupiny clusterov na menšie a využiť tým viac testovacích systémov. V prípade využitia dvoch testovacích systémov je vidieť efektivitu tejto optimalizácie, nakoľko sa dve skupiny clusterov rozdelia tak, aby sa dva testovacie systémy využili rovnomerne. Odhadovaná dĺžka testovania sa na týchto dvoch systémoch bude líšiť o 25 sekúnd, namiesto takmer siedmych minút, o ktoré by sa tento čas líšil v prípade nevyužitia optimalizácie.

V prípade použitia viacerých testovacích systémov môžeme na tabuľke jasne vidieť, že sa niektoré testovacie systémy bez použitia optimalizácie nevyužijú. V prípade využitia optimalizácie sa skupiny clusterov rozdelia tak efektívne, že rozdiely dĺžky trvania testovania medzi jednotlivými testovacími systémami sú rádovo v sekundách.

počet testovacích systémov	Dĺžka regresného testovania			
	bez optimalizácii	optimalizácia 1	optimalizácia 2	optimalizácia 3
1	15:08:32	15:08:32	15:08:32	15:08:32
2	08:19:32	08:23:25	08:23:25	08:07:32
3	05:54:31	05:53:21	05:53:21	05:36:16
4	04:40:18	04:51:22	04:51:22	04:08:06
5	03:52:44	03:50:06	03:50:06	03:37:21
6	03:15:13	03:18:23	03:18:23	03:17:33
7	02:53:37	03:00:35	03:00:35	02:46:58
8	02:38:00	02:46:31	02:46:31	02:41:35
9	02:27:12	02:27:25	02:27:25	02:28:20
10	02:21:39	02:08:54	02:08:54	02:10:00

Tabuľka 6.7: Porovnanie jednotlivých optimalizácii pri plánovaní regresných testov v produkte SMSCv5

testovací systém	dĺžka testovania bez využitia optimalizácie	dĺžka testovania s využitím optimalizácie
1	11:31	11:31
1	03:46	07:27
2	10:54	07:52
1	03:46	06:00
2	10:54	06:32
3	-	06:35
1	03:46	05:35
2	10:54	05:34
3	-	05:52
4	-	05:54
1	03:46	05:14
2	10:54	05:02
3	-	05:28
4	-	05:29
5	-	05:30

Tabuľka 6.8: Porovnanie vplyvu optimalizácie 2 na testovacie systémy pri plánovaní funkcionality *addrtran*

Kapitola 7

Záver

Táto diplomová práca je zameraná na návrh a implementáciu rozšírenia existujúceho nástroja pre plánovanie a spúšťanie testov tak, aby bolo možné testy distribuovať na viaceré testovacie systémy. Záver tejto diplomovej práce je venovaný zhrnutiu odvedenej práce a dosiahnutých výsledkov. Súčasne na záver uvádzam návrh možností ďalšieho vývoja implementovaného nástroja.

7.1 Zhrnutie

Pre riešenie práce som potreboval naštudovať existujúci nástroj pre plánovanie testov a teoretické znalosti o distribúcii testov. Medzi tieto teoretické znalosti patrili hlavne základné princípy distribúcie testov a požiadavky pre nástroj schopný takejto distribúcie.

Po naštudovaní potrebných teoretických znalostí som sa zameral na vytvorenie základného konceptu fungovania vytváraného nástroja. Pri jeho tvorbe som sa zameriaval hlavne na centralizovanú správu testov, na rozloženie záťaže medzi jednotlivými testovacími systémami, a na transparentnú a automatickú distribúciu testov. Naimplementované rozšírenie nástroja pre plánovanie a spúšťanie testov týmito vlastnosťami disponuje.

Pri implementovaní som sa zameriaval na optimalizáciu algoritmov tak, aby boli čo najviac efektívne pre testovanie priamo v praxi. Aj keď je teoreticky možné distribuovať testy na obrovský počet testovacích systémov, v praxi je problém zohnať čo i len niekoľko testovacích systémov navyše. Distribúcia testov je teda optimalizovaná pre rádovo niekoľko testovacích systémov a použitím viacerých testovacích systémov efektivita distribúcie testov klesá.

Po dokončení implementácie distribúcie testov som ešte nad rámec začal s implementáciou niekoľkých optimalizácii. Tieto optimalizácie slúžia hlavne na zníženie času potrebného pre dokončenie testovania a pre rovnomernejšie rozloženie záťaže medzi testovacími systémami.

Hotová práca sa používa pri testovaní niekoľkých produktov vo firme Acision, nakoľko testovanie týchto produktov sa postupom času začalo stávať časovo náročné a túto časovú náročnosť bolo potrebné znížiť.

7.2 Možnosti ďalšieho vývoja

Analýza výsledkov vytvárania plánov testov na viacerých testovacích systémoch odhalila niektoré nedokonalosti, ktoré otvárajú priestor pre ďalšiu prácu.

Medzi hlavnú nedokonalosť vytvoreného rozšírenia patrí viacnásobné spúšťanie niektorých testov. Aj keď tento nedostatok nie je možné z povahy plánovača testov úplne odstrániť, je možné ho aspoň zmierniť. Ako možnosť ďalšieho vývoja je teda vhodné popremýšľať o novom algoritme, ktorý by takéto viacnásobné spúšťanie testov minimalizoval.

Ďalším zaujímavým nápadom je možnosť komunikácie medzi jednotlivými testovacími systémami. Pomocou tohoto návrhu by sme totiž mohli dosiahnuť to, aby sa minimalizovalo doplnenie *Preoptov* na každom testovacom systéme. Pri vhodnom naplánovaní testov by sme totižto mohli prvý výsledok každého *Preoptu* preposlať na ostatné testovacie systémy, ktoré by sa potom týmto výsledkom mohli riadiť. Takýmto spôsobom by sme mohli zabrániť tomu, aby sa niektoré clustre spúšťali na každom testovacom systéme zvlášť.

Posledným návrhom možného ďalšieho vývoja je prerobenie algoritmu plánovača testov tak, aby bol schopný distribuovať testy na základe častí plánu. Tento algoritmus je pre distribúciu testov najvhodnejší, no pre jeho implementáciu by bolo nutné zmeniť logiku celého plánovača testov.

Literatúra

[1] *IEEE Standard for Software Unit Testing*. IEEE Computer Society, 1986, iSBN 1-55937-672-4.

[2] *IEEE Standard Glossary of Software Engineering Terminology*. IEEE Computer Society, 1990, iSBN 1-55937-067-X.

[3] *IEEE Guide for Software Verification and Validation Plans*. IEEE Computer Society, 1993, iSBN 0-7381-0410-8.

[4] Abdul, F.; Fhang, M.: Implementing Continuous Integration towards rapid application development. In *Innovation Management and Technology Research (ICIMTR), 2012 International Conference on*, 2012, s. 118–123, iSBN 978-1-4673-0655-3.

[5] Abram, A.; Moore, J.; Bourque, P.; aj.: Guide to the Software Engineering Body of Knowledge [online]. http://www.inf.ed.ac.uk/teaching/courses/seoc/2006_2007/resources/SWEBOK_Guide_2004.pdf, 2004 [cit. 2014-04-16], iSBN 0-7695-2330-7.

[6] Fonseca, F. M. M.: A Parallel Execution Approach for Efficient Regression Testing in the OutSystems Test Infrastructure. Universidade de Lisboa, 2009.

[7] Fowler, M.: Continuous Integration [online]. `http://www.martinfowler.com/articles/continuousIntegration.html`, 2006-05-01 [cit. 2013-03-27].

[8] Kapfhammer, G. M.: Automatically and transparently distributing the execution of regression test suites. *Proceedings of the 18th International Conference on Testing Computer Software*, 2001.

[9] Pressman, R. S.: *Software engineering (7th ed.): a practitioner's approach*. McGraw-Hill, Inc., 2010, iSBN 978-0-07-337597-7.

Príloha A

Obsah CD

Na priloženom CD sa nachádzajú nasledovné súbory a adresáre:

- **doc** – zdrojové súbory tohoto textu v LaTeX-ovom formáte spolu s elektronickou verziou vo formáte PDF.
- **TTT** – zdrojové súbory plánovača testov spolu s hlavičkami testov používaných v praxi. Tento adresár takisto obsahuje súbory so známymi chybami *KNOWN.BUGS* a funkcie pre simuláciu testovania pre potreby demonštrácie.
- **working_dir** – pracovný adresár s konfiguračnými súbormi pre produkty MCO a SMSCv5 a štatistickými súbormi pre tieto dva produkty. Do tohoto adresára sa v prípade spustenia plánovača testov budú zapisovať logovacie informácie.
- **rpms** – *rpm* súbory pre nainštalovanie skriptovacieho jazyka Tcl/Tk vo verzii 8.4.19, pre ktorú je plánovač testov napísaný a otestovaný.
- **README.txt -** textový súbor obsahujúci príklady spustenia.

Príloha B

Ukážka jednoduchého plánu testov z praxe

Na nasledujúcej strane je zobrazený plán testov z produktu MCO, ktorý pozostáva z desiatich clusterov. Z obrázku vyplýva, že pre potreby týchto clusterov je potrebné spustiť celkovo 19 prerekvizít a 19 odrekvizít. Pre otestovanie produktu týmito desiatimi clustermi, je potrebné vykonať ďalších 38 testov, ktoré slúžia na zmenu konfigurácie systému, a na obnovu týchto zmien.

Uvedený plán obsahuje celkovo 48 testov, a je relatívne jednoduchý. V regresných testoch sa aktuálne používa plán testov, ktorý v prípade produktu MCO pozostáva z viac ako 3100 testov. Plán testov každodenne používaný v regresnom testovaní produktu SMSCv5 pozostáva z približne 3000 testov.

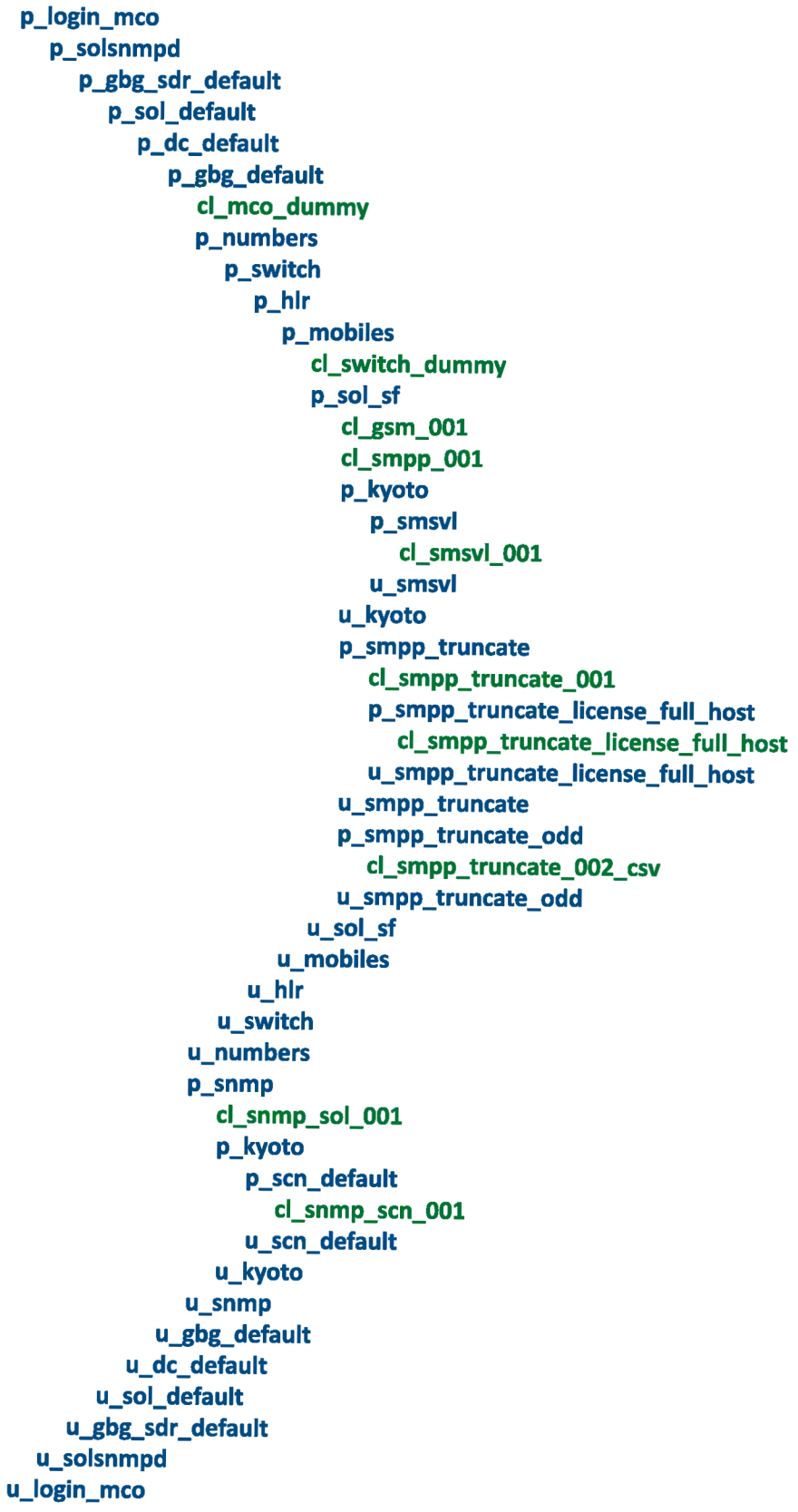
V produktoch MCO a SMSCv5 sa každodenne spúšťajú regresné testy, ktoré tvoria veľkú časť zo všetkých dostupných testov. Spúšťanie regresných testov pre produkty MCO a SMSCv5 je zabezpečované nasledujúcimi dvomi príkazmi.

```
meszarosf@mco-v156#./Run_test.tcl -i mco.ini -B TEST.REVIEW_23 -f all -tw
```

Príklad B.1: Spúšťanie regresných testov v produkte MCO

```
meszarosf@smsc-v156#./Run_test.tcl -i smsc.ini -f all 2nd -tw
```

Príklad B.2: Spúšťanie regresných testov v produkte SMSCv5



Obrázok B.1: Ukážka jednoduchého plánu testov

Príloha C

Zmeny prevedené do zdrojových súborov a odovzdané súbory

Pri implementovaní rozšírenia plánovača testov som musel upraviť zdrojové kódy dvoch hlavných súborov tohoto nástroja. Nakoľko boli tieto dva súbory vytvorené firmou Acision, v tejto prílohe sú uvedené zmeny do týchto dvoch súborov vykonané pri vytváraní tejto práce. Približné zmeny týchto dvoch zdrojových súborov sú nasledovné:
Zmeny prevedené v súbore **Run_test.tcl**:
Riadky: 1433–1437, 1461–1464, 1467–1493 a ďalej riadky 1499 až 1507.
Celkovo približne 50 riadkov kódu (bez komentárov).

Zmeny prevedené v súbore **tstmng.tcl**:
Riadky: 125–133, 2257–2268, 2301–2322, 2551–2582, 2622–2624, 3095–3097, 3199–3202, 3238–3242 a riadky 3252 až 3260.
Ďalej riadky: 2360, 2365, 2369, 2374, 2379, 2384, 2410, 2415, 2424, 2457, 2477, 2482, 2488, 2504, 2528, 2533, 2538, 2543, 2548 a riadok 2642.
Celkovo približne 123 riadkov kódu (bez komentárov).

Pri odovzdávaní zdrojových súborov som navyše odovzdal len súbory, potrebné pre beh plánovača testov. Jednotlivé testy pre produkty MCO a SMSCv5 boli odstránené a nahradené jednoduchou funkciou `cl_wait_random_ok` umiestnenou v súbore *CL-MY.tcl*.

Hlavičky testov v súboroch *TEST.ALL* a *TEST.REVIEW_23* boli ponechané priamo z produktov MCO a SMSCv5. V týchto súboroch som však zmenil popis testov (informácia *Description*) a funkcionalitu (informácia *Function*) popísané v kapitole 2.4. Jednotlivé závislosti medzi testami ostali ponechané, a je teda možné vytvoriť plány testov, ktoré sa používajú priamo v praxi. Súbory *KNOWN.BUGS* ostali ponechané priamo z týchto dvoch produktov bez zmeny.

Medzi odovzdanými súbormi sa taktiež nachádzajú dva štatistické súbory slúžiace napríklad pre výpočet celkovej dĺžky trvania testovania. Medzi tieto súbory patria dva súbory *TTT_stat_MCO.txt* (pre produkt MCO) a súbor *TTT_stat_SMSC.txt* (pre produkt SMSCv5). Tieto súbory obsahujú presné informácie o dĺžkach trvania jednotlivých testov v týchto dvoch produktov. Pre využitie týchto súborov pre konkrétny produkt je potrebné jeden z týchto súborov vložiť do logovacieho adresára *Log/Stat* a premenovať ho na súbor *TTT_stat.txt*. V prípade použitia týchto štatistických súborov plánovač testov počíta odhadované časy dĺžky trvania testovania na základe údajov priamo z praxe.

Príloha D

Ukážka progress baru

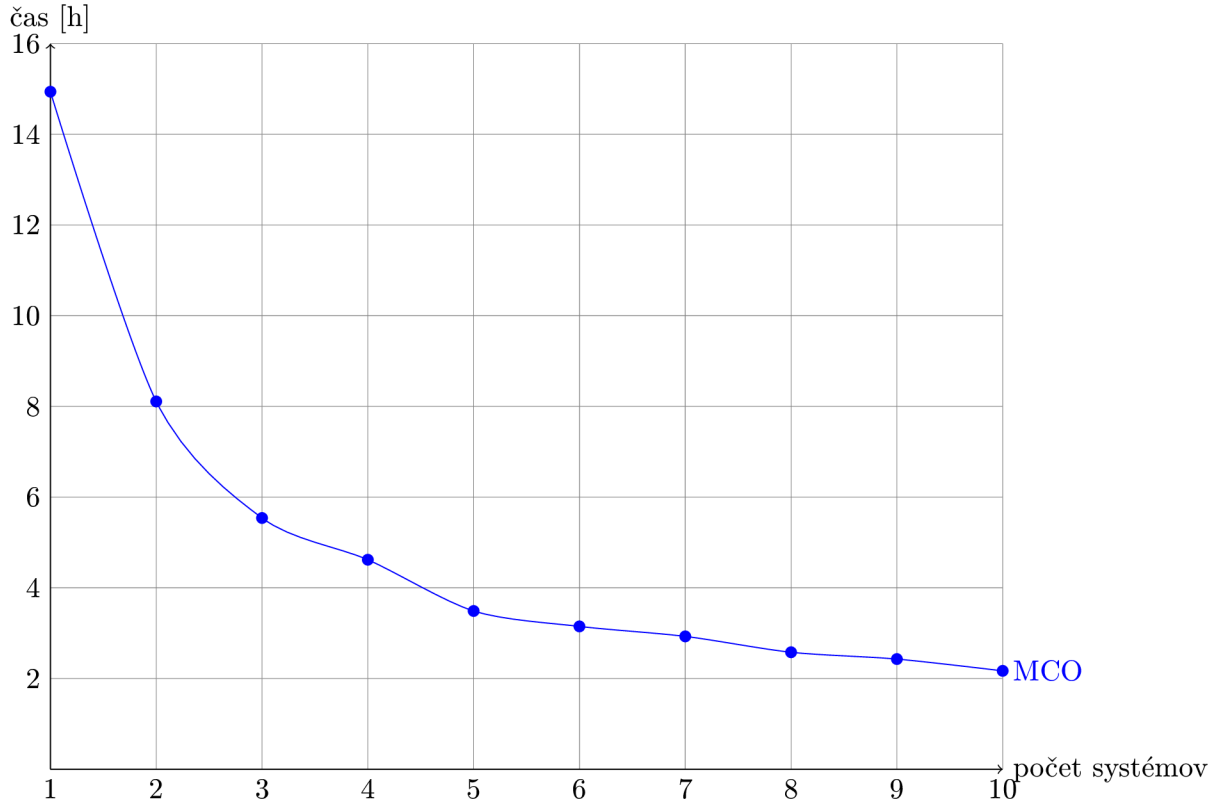
```
status: ##################################
status: Running parallel testing on 3 nodes
status: ##################################
status:
status: pid of the main process      : 1882
status: pids of the all subprocesses : 1891 1892 1893
status:
status: ===============  Total:  50.52%  |  1414 tests of 2799  |  Time elapsed: 00:00:56  ===============
status:
status: Node 1:   0.7% [#.............................]  |     7 tests of  986
status: Node 2:  52.2% [################..............]  |   443 tests of  849
status: Node 3: 100.0% [##############################]  |   964 tests of  964  |  Finished in: 00:00:56
status:
status: Number of PASSED  test clusters: 0 of 1487
status: Number of FAILED  test clusters: 7 of 1487
status: Number of SKIPPED test clusters: 770 of 1487
```

Obrázok D.1: Ukážka progress baru s väčšou šírkou terminálu

Príloha E

Prínos práce pre produkt MCO

Nasledujúci graf zobrazuje závislosť medzi počtom použitých testovacích systémov a časom potrebným pre otestovanie produktu MCO regresnou sadou testov. Uvedené hodnoty sú vygenerované zo štatistík získaných priamo z praxe pri testovaní produktu MCO.

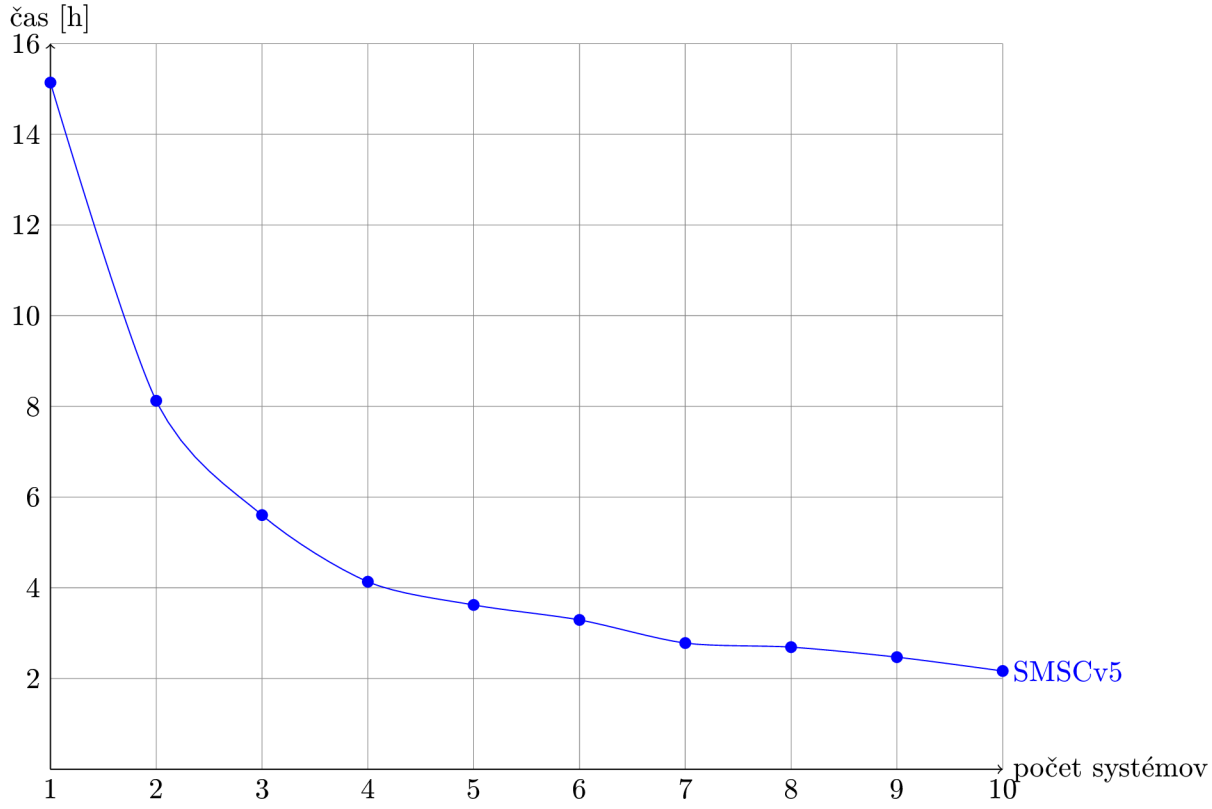


Obrázok E.1: Závislosť medzi počtom testovacích systémov a dĺžkou testovania v produkte MCO

Príloha F

Prínos práce pre produkt SMSCv5

Nasledujúci graf zobrazuje závislosť medzi počtom použitých testovacích systémov a časom potrebným pre otestovanie produktu SMSCv5 regresnou sadou testov. Uvedené hodnoty sú vygenerované zo štatistík získaných priamo z praxe pri testovaní produktu SMSCv5.



Obrázok F.1: Závislosť medzi počtom testovacích systémov a dĺžkou testovania v produkte SMSCv5