UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Žiga Elsner

Uporaba orodja FitNesse za izdelavo sprejemnih testov

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: prof. dr. Viljan Mahnič

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00104/2011
Datum: 04.04.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ŽIGA ELSNER**

Naslov: **UPORABA ORODJA FITNESSE ZA IZDELAVO SPREJEMNIH TESTOV**
USING FITNESSE FOR ACCEPTANCE TESTING

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V agilnih metodah (npr. Scrum, ekstremno programiranje) so zahteve uporabnikov opisane v obliki ti. uporabniških zgodb, katerih pomemben sestavni del so sprejemni testi. V svoji nalogi najprej predstavite metodo Scrum in proučite najpomembnejše značilnosti uporabniških zgodb. Nato opišite orodje Fitnesse in možnosti, ki jih to orodje nudi za pripravo in avtomatsko izvajanje sprejemnih testov. Uporabo tega orodja prikažite na praktičnem primeru.

Mentor:
prof. dr. Viljan Mahnič

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Žiga Elsner,

z vpisno številko 63050029,

sem avtor/-ica diplomskega dela z naslovom:

Uporaba orodja FitNesse za izdelavo sprejemnih testov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom
  prof. dr. Viljan Mahnič
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 28.06.2011 Podpis avtorja/-ice:

Zahvala

Zahvaljujem se mentorju prof. dr. Viljanu Mahniču za vso pomoč in nasvete, ki mi jih je nudil tekom izdelave diplomske naloge, ter vsem ostalim, ki so kakorkoli prispevali k nastanku diplomske naloge.

Posebna zahvala je namenjena celotni družini, posebej očetu Mihi in materi Lilijani, ki sta me ves čas študija moralno in finančno podpirala.

Kazalo

Povzetek

V diplomski nalogi je predstavljen pomen in osnovni principi agilnih metodologij razvoja programske opreme. Med njimi posebno izstopa metodologija Scrum, ki je v zadnjem času najbolj razširjena agilna metodologija razvoja programske opreme. V nadaljevanju je razloženo, kako je možno proces metodologije Scrum še izboljšati z vpeljavo uporabniških zgodb. Opisano je, na kaj moramo biti pozorni pri pisanju zgodb, kaj so to sprejemni testi, ter zakaj so dobri. Na koncu poglavja o uporabniških zgodbah je predstavljeno, kako uporabniške zgodbe uporabimo za planiranje in načrtovanje. Sledi poglavje z opisom orodja FitNesse, vse od namestitve pa do tega, kako lahko spišemo svoj prvi test. Na koncu sledi prikaz uporabe orodja Fitnesse na enostavnem primeru.

Ključne besede:

agilne metodologije, Scrum, uporabniške zgodbe, sprejemni testi,
orodje FitNesse

Abstract

In the thesis we present the importance and basic principles of the agile methodology of software development. The special emphasis is on Scrum methodology, which has been widely used in recent software development. Further on, we describe how the process of Scrum methodology may be further improved by imposing user stories. We point out where we have to pay attention while writing stories, what are so called acceptance tests and why they are important. At the end of this chapter it is described how user stories may be used for planning (and designing). The next chapter describes FitNesse tool, from the installation to the step-by-step instructions to write the first test. At the end we show an example of FitNesse tool in practice.

Key words:

Agile Methodology, Scrum, User Stories, Acceptance Test, FitNesse Tool

Poglavje 1

Uvod

Pri razvoju programske opreme se srečamo z številnimi problemi. Prvi izmed njih je, kako predstaviti zahteve programske opreme, da bodo razumljive vsem vpletenim na projektu, ne glede na njihovo funkcijo. Ker so funkcije sodelujočih različne, je med njimi potrebna komunikacija. Stalna uspešna komunikacija zmanjšuje verjetnost, da bi prišlo do ustavitve projekta zaradi nerazumevanja problema. Ravno zaradi tega se poskušamo izogibati preobsežnemu zbiranju zahtev in preobsežni dokumentaciji ter imamo zbranega le toliko, kot potrebujemo za načrtovanje in planiranje, hkrati pa s tem spodbujamo komunikacijo [3], ki ni pomembna samo pri razvoju programske opreme, temveč tudi pri vsakdanjih človeških odnosih.

Kot drugi problem bi omenil stalen pritok novih idej in sprememb s strani uporabnika. Zaradi tega je težko predvideti čas razvoja programske opreme. Prav tako nastanejo težave pri načrtovanju programske opreme. Zato moramo imeti tak proces, ki se je zmožen prilagoditi na spreminjajoče se zahteve. Tak proces mora hitro in čim bolj pogosto zagotavljati potrebne informacije. Odločitve sprejemamo na podlagi informacij, ki jih trenutno imamo. Te odločitve delamo pogosto [3].

Tretji problem so napake oz. hrošči v programski kodi. Za iskanje napak v programski kodi se uporablja tehnika, ki se ji reče testiranje. Testiranje je proces potrjevanja in preverjanja, da programska koda dosega cilje, ki so bili začrtani med razvojem, in da deluje, kot je to od nje pričakovano [5]. Običajno (pri klasičnem razvoju) je testiranje izvedeno po koncu faze kodiranja, dasiravno se postopki testiranja razlikujejo glede na metodologijo razvoja programske opreme. Najnovejši razvojni modeli, kot so npr. agilne metodologije razvoja programske opreme pa pogosto izvajajo testiranje med samim procesom razvoja. Ker naj bi z zgodnjim odkritjem hroščev prihranili denar za

popravke, so se te metodologije izkazale za zelo uspešne. Njihova uporaba se je v zadnjih 10 letih znatno povečala. Ker je testiranje pomemben del procesa razvoja programske opreme so razvili vrsto orodij za testiranje. Med njimi je seveda odlično orodje za avtomatiziranje sprejemnih testov, FitNesse [4], ki je osrednja tema moje diplomske naloge in je opisan v nadaljevanju. Orodje je postalo zelo popularno, še posebej v kombinaciji z agilnimi metodologijami. Moja naloga je bila uporabiti orodje FitNesse, ki ga najpogosteje srečamo v navezi z metodologijo Scrum [2] in uporabniškimi zgodbami [3]. Te tehnologije in postopki, poskušajo v veliki meri odpraviti zgoraj opisane probleme.

Poglavje 2

Agilne metodologije razvoja programske opreme

2.1 Uvod

Agilne metode razvoja programske opreme so ena od skupin metodologij razvoja programske opreme, ki temeljijo na iterativnem in inkrementalnem procesu razvoja. Agilne metode so nastale z namenom odstranitve problemov, ki jih prinašajo plansko vodene metodologije.

Problemi plansko vodenih metodologij so:

- dolg življenski cikel,
- zahtevnost za učenje in uporabo,
- preveč dokumentacije,
- neprilagodljivost in premajhna fleksibilnost (vključevanje sprememb med razvojem ni možno).

Za agilni pristop pa so značilni:

- inkrementalen razvoj: kratki cikli,
- stalna komunikacija in sodelovanje z uporabniki,
- enostavno učenje in uporaba metod,
- enostavno sprotno vključevanje sprememb zahtev naročnika.

2.2 Manifest agilnosti

Da bi odpravili težave, ki se pojavljajo pri plansko vodenih metodologijah, se je februarja 2001, na območju Snowbird v državi Utah, zbralo 17 razvijalcev, da bi se pogovorili o lažjem in bolj učinkovitev razvoju programske opreme. Nastal je manifest agilnosti (ang. Agile Manifesto) [1].

Na sestanku so prišli do naslednjih spoznanj [1]:

- **Posamezniki in njihova medsebojna komunikacija** so bolj pomembni od procesa in razvojnih orodij.
- **Delujoč program** je bolj pomemben od popolne dokumentacije.
- **Sodelovanje med razvijalci in naročnikom** je bolj pomembno od pogodbeno dogovorjenega razvoja programske opreme.
- **Fleksibilnost pri spremembah zahtev** je bolj pomembna od sledenja začrtanemu načrtu.

2.3 Osnovni principi agilnih metodologij

12 osnovnih principov agilnih metodologij [1]:

1. Zadovoljiti naročnika z neprestano dostavo uporabne programske opreme.
2. Spremembe zahtev so dobrodošle, tudi kasneje v razvoju.
3. Dostava delujoče programske opreme naj bo čim pogostejša.
4. Tesno, vsakodnevno sodelovanje in komuniciranje med naročniki in razvijalci.
5. Projekte je potrebno zastaviti okrog motiviranih, zaupanja vrednih posameznikov in jim nuditi ustrezno okolje in potrebne vire.
6. Osebni pogovor je najboljša oblika komunikacije.
7. Najpomembnejše merilo napredka je količina delujoče programske kode.
8. Težnja k stalnemu in enakomernemu razvoju.

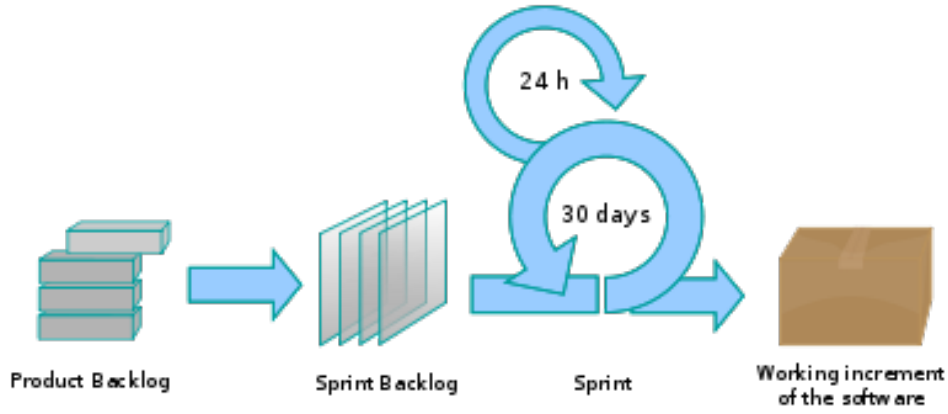
9. Nenehna težnja za tehnično odličnost in optimalen načrt razvijajoče programske opreme pospešuje agilnost.
10. Enostavnost.
11. Na projekte najbolje vplivajo skupine, ki se znajo same organizirati in voditi.
12. Redni pregled procesa in prilagajanje skupine na spreminjajoče se razmere.

Poglavje 3

Agilna metodologija Scrum

3.1 Metodologija Scrum

Scrum [2] je ena izmed agilnih metodologij razvoja programske opreme. Metodologija poudarja prožnost, prilagodljivost in ustvarjalnost. Proces razvoja je iterativen in inkrementalen. Osnovna ideja takega procesa je razviti sistem po delih (inkrementih), pri katerem vsak del izboljšujemo skozi ponavljajoče se cikle (iteracije). Vsak inkrement je v celoti skodiran in stestiran in pričakovano je, da se ne bo potrebno ponovno vračati nanj. Scrum projekti napredujejo skozi serijo 30-dnevnih ciklov oz. iteracij, imenovanih teki (ang. Sprint). Na začetku vsakega teka skupina oceni, koliko dela je možno narediti. To naredimo na sestanku planiranja teka (ang. Sprint Planning Meeting). Delo je izbrano iz prioritetnega seznama zahtev (ang. Product Backlog). Kratki dnevni sestanki (ang. Daily Scrum) omogočajo ekipi, da pregleda napredek in se prilagodi na razmere, če je to potrebno. Na koncu vsakega teka dobimo del delujoče in stestirane programske opreme, ki je pripravljena, da se jo pošlje naročniku. Grafično je Scrum prikazan na sliki 3.1.



Slika 3.1: Scrum proces.

3.2 Vloge

Glavne vloge pri metodologiji Scrumi so skrbnik metodologije (ang. Scrum Master), lastnik izdelka (ang. Product Owner) in razvojna skupina (ang. Team).

3.2.1 Razvojna skupina

Razvojna skupina ponavadi šteje od 5 do 9 razvijalcev. Najbolj idealno je, če je skupina sestavljena iz 7 razvijalcev [2]. Čeprav bi razvojna skupina lahko imela specializirane ljudi na svojih področjih (testerje, administratorje podatkovnih baz, itd.) je ideja Scrum skupin, da so vsi odgovorni za vse. Če je potrebno izvesti testiranje in pravega testerja ni, mora odgovornost prevzeti nekdo drug. Pri razvojni skupini vloge, kot so programer, arhitekt in tester ponavadi ne obstajajo in je vse prepuščeno dogovoru znotraj skupine. Take skupine so samoorganizirane, kar pomeni, da same izberejo način, kako realizirati posamezne zahteve. Poleg tega so same odgovorne za uspeh celotnega projekta. Glavna naloga razvojne skupine je razvoj novih funkcij znotraj vsake iteracije.

3.2.2 Skrbnik metodologije

Skrbnik metodologije skrbi, da se celoten proces izvaja v skladu z metodologijo Scrum. Njegove naloge so učenje metodologije vseh, ki sodelujejo na projektu,

zagotoviti mora, da se upoštevajo vsa pravila, ter poskrbeti, da metodologija daje pričakovane rezultate in se učinkovito vklaplja v samo kulturo organizacije.

3.2.3 Lastnik izdelka

Lastnik izdelka je tisti, ki mu pri ekstremnem programiranju pravimo naročnik (ang. Customer). Lastnik izdelka skrbi za zagotavljanje sredstev in vzdrževanje seznama zahtev (ang. Product Backlog). Odgovoren je za izdelavo začetnega seznama zahtev in določitev prioritete posameznih zahtev. Določa vsebino izdaj (ang. Release).

3.3 Izdelki metodologije Scrum

Izdelki metodologije Scrum so seznam zahtev (ang. Product Backlog), plan teka (ang. Sprint Backlog) in uporaben izdelek z dodano funkcionalnostjo (ang. Increment of Potentialy Shippable Product Functionality).

3.3.1 Seznam zahtev

Seznam zahtev predstavlja spisek vseh želenih funkcionalnosti končnega izdelka. Seznam ni nikoli dokončen in se med delom dopolnjuje in spreminja. Obstaja toliko časa, dokler je izdelek v uporabi. Na začetku projekta še ni potrebno, da so predvidene in zapisane vse funkcije izdelka.

Za stalno spreminjanje in dopolnjevanje seznama zahtev je odgovoren lastnik izdelka. Priporočeno je, da je lastnik izdelka poleg osnovnih nalog zadolžen tudi za razvrščanje zahtev po prioriteti.

V zadnjem času prihaja v veljavo, da so zahteve prikazane v obliki uporabniških zgodb. Več o uporabniških zgodbah v poglavju 4.

3.3.2 Plan teka

Plan teka predstavlja seznam nalog, ki morajo biti opravljene v enem teku. Plan teka izdela razvojna skupina na sestanku planiranja teka (ang. Sprint Planning Meeting), potem ko izberejo zahteve, ki jih bodo realizirali v tem teku. Obseg dela za vsako nalogo mora znašati med 4 in 16 delovnih ur. Plan teka lahko spreminja samo razvojna skupina.

3.3.3 Uporaben izdelek z dodano funkcionalnostjo

Na koncu vsakega teka dobimo kot rezultat uporaben izdelek, ki ima dodatno funkcionalnost. Tak izdelek je dobro strukturiran, dobro spisan in stestiran. Poleg izdelka mora biti priložena dokumentacija v obliki navodil za uporabnika.

3.4 Sestanki metodologije Scrum

Med procesom Scrum se izvajajo številni sestanki. Mednje sodijo sestanek planiranja teka (ang. Sprint Planning Meeting), dnevni sestanek (ang. Daily Scrum), sestanek pregleda teka (ang. Sprint Review Meeting) in sestanek ocene teka (ang. Sprint Retrospective Meeting). Vsi ti sestanki obstajajo z namenom, da članom omogočajo sproten pregled in prilagoditev procesa.

3.4.1 Sestanek planiranja teka

Lastnik izdelka in razvojna skupina se na sestanku planiranja teka dogovorijo, katere zahteve bodo realizirali v tem teku. Sestanek se izvaja na začetku vsakega teka in traja predvidoma 8 ur oz. dvakrat po 4 ure. Ko so zahteve izbrane, razvojna skupina izdela plan teka.

3.4.2 Dnevni sestanek

Dnevne sestanke se izvaja vsak dan. Tipično se skličejo čimbolj zgodaj, vendar šele, ko prispejo vsi odgovorni člani skupine. Sestanki se začno vedno ob istem času, ponavadi je to med 9:00 in 9:30. So kratki in trajajo 15 minut. Da bi poskrbeli, da so sestanki res kratki, nekatere skupine zahtevajo, da udeleženci stojijo.

Med sestankom vsak član skupine odgovori na tri vprašanja:

1. Kaj si naredil včeraj?

2. Kaj boš naredil danes?

3. Ali si se pri delu srečal s kakšnimi težavami?

3.4.3 Sestanek pregleda teka

Sestanek pregleda teka se izvaja po koncu vsakega teka. Traja največ 4 ure. Na njem razvojna skupina predstavi rezultate teka lastniku izdelka in ostalim zainteresiranim. Prikaže se nove lastnosti izdelka in se pogovori o morebitnih spremembah, ki bi jih želeli v novi verziji izdelka.

Na sestanku se pregleda, če so bili doseženi cilji, ki so bili določeni na sestanku planiranja teka. Idealno je, če so končane vse zadane točke teka, še bolj pomembno pa je, da je dosežen nek splošen cilj.

3.4.4 Sestanek ocene teka

Sestanek se izvede na koncu vsakega teka in traja največ 3 ure. Na njem skrbnik metodologije in razvojna skupina ocenijo potek teka in predlagajo morebitne izboljšave procesa.

3.5 Prednosti in slabosti metodologije Scrum

Prednosti metodologije Scrum [3]:

- Majhno število razvijalcev omogoča učinkovitejšo in hitrejšo komunikacijo med njimi, ustvari se prijetna klima in občutek pripadnosti, znanje naraste.
- Boljše poznavanje trenutnega stanja projekta, zaradi pogoste komunikacije in sestankov.
- Zaradi kratkih razvojnih ciklov se manjša možnost frustracij ob neuspehih, očiten je napredek, čeprav se zahteve pogosto spreminjajo.
- Boljši odnos z naročnikom, naročnik dobi občutek, da se na projektu dejansko dela, zaradi stalne dostave programske opreme, ter sodelovanja pri projektu.

Slabosti metodologije Scrum [3]:

- Neučinkovitost pri večjih in bolj zapletenih projektih.
- Potrebno znanje o uporabi metodologije.
- Problematična izbira vlog, posamezniki morajo biti prilagodljivi in ustvarjalni, morajo se zavedati dolžnosti in pravic.
- Čeprav je metodologija učinkovita pri prikazu trenutnega stanja projekta nam ne da nadzora nad celoto projekta.

Poglavje 4

Uporabniške zgodbe in metodologija Scrum

4.1 Opis uporabniške zgodbe

Uporabniške zgodbe (ang. user story) [3] opisujejo funkcionalnosti, ki so pomembne tako za uporabnika, kot za naročnika sistema ali programske opreme.

Uporabniške zgodbe so lahko shranjene elektronsko, bolj pogosto pa jih srečamo v fizični obliki, napisane na roko, na kartice papirja. Na sprednji strani kartice so napisani podatki kot so: besedilo uporabniške zgodbe, zabeležka (ang. note), prioriteta (ang. priority) in ocena (ang. estimate) uporabniške zgodbe. Na zadnji strani so izpostavljeni sprejemni testi (ang. acceptance tests).

Primeri uporabniških zgodb:

- Uporabnik lahko napiše komentar.
- Uporabnik lahko plača s kreditno kartico.
- Podjetje lahko doda novo prosto delovno mesto.

Glede na to, da uporabniške zgodbe pišejo uporabniki in so namenjene razvijalcem, še primeri neprimernih uporabniških zgodb:

- Programska oprema bo napisana v Javi.
- Program se bo povezal s podatkovno bazo MySQL.

Uporabnika ne zanima, ali bo program napisan v Javi ali Pythonu. Za uporabnika je pomembno, da končni izdelek deluje.

Trije pogledi na uporabniške zgodbe:

- Opis zgodbe je uporabljen za planiranje in kot opomnik, o čem se je treba pogovarjati z naročnikom.
- Iz pogovorov o zgodbi lahko izluščimo podrobnosti o zgodbi.
- Testi nam povedo, kdaj je zgodba končana.

4.2 Zajemanje uporabniških zgodb

Poznamo različne tehnike zajemanja zgodb:

- intervjuji,
- vprašalniki,
- opazovanje,
- delavnice pisanja zgodb.

Odgovornost programerja je, da razume vse tehnike zajemanja zgodb in jih zna pravilno uporabiti. Pri intervjujih je pomembno, da pozna razliko med odprtimi, zaprtimi ter kontekstno neodvisnimi vprašanji. Pri zaprtih vprašanjih vprašanemu ne pustimo veliko prostora za nič drugega, kot le za enostaven odgovor z da ali ne. Obraten učinek dosežemo z odprtimi vprašanji, s katerimi vprašanemu dovolimo večjo svobodo pri odgovorih. Problem takih vprašanj je, da lahko vprašani prehitro zaide od bistva. Prav tako je pomembno poznavanje kontekstno neodvisnih vprašanj. To so splošna vprašanja o projektu in okolju na katerem bo tekel končni produkt. Pri teh vprašanjih je pomembno, da se izogibamo takih vprašanj, ki že vnaprej ponujajo odgovor.

Odgovornosti naročnika so razumevanje tehnik zajemanja zgodb, pisanje zgodb, razumevanje opcij pri komunikaciji z uporabnikom. Poleg tega mora razumeti, kako postaviti vprašanja ter poskrbeti za delavnice pisanja zgodb. Paziti mora, da so vse vloge, v katerih lahko nastopajo uporabniki, pravilno izbrane.

4.3 Pisanje uporabniških zgodb

Pri pisanju uporabniških zgodb se poskušamo osredotočiti na 6 lastnosti:

- neodvisnost,
- prilagodljivost,
- pomembnost za uporabnika oz. naročnika,
- predvidljiv obseg oz. kompleksnost,
- majhnost,
- omogočajo testiranje.

Bill Wake predlaga za te lastnosti kratico INVEST (Independent, Negotiable, Valuable to users and customers, Estimatable, Small, Testable) [3].

4.3.1 Neodvisnost

Če je le možno se poskušamo izogniti odvisnosti med zgodbami, saj to lahko privede do problemov pri prioritiziranju zgodb in načrtovanju. Na primer, da imamo zgodbo z visoko prioriteto, ki je odvisna od zgodbe z nizko prioriteto. V takem primeru bi bilo nujno implementirati tudi zgodbo z nizko prioriteto. Drugi problem, ki se pojavi, je pri ocenjevanju časa razvoja. Če imamo več zgodb, ki so si zelo podobne po funkcionalnosti, bo potrebnega več časa za razvoj prve, preostale pa bodo razvite hitreje.

Dve možni rešitvi za zgoraj navedene probleme:

- združitev odvisnih zgodb v eno večjo, toda neodvisno zgodbo,
- najti drugačen način, kako razdeliti zgodbo na več manjših

4.3.2 Prilagodljivost

Zgodbe so prilagodljive. Ker so zgodbe napisane na kartice, ki so omejene z prostorom in ne vsebujejo preveliko podrobnosti o zgodbi, puščajo možnost, da se o zgodbi pogovorimo. Kartice so samo opomnik za programerje in naročniško skupino in naj ne bi vključevale pomembnih podrobnosti o zgodbi. Če so podrobnosti kljub temu že znane na začetku pisanja zgodbe, jih vključimo

kot opombo na samo kartico. Pri tem moramo paziti, da smo kratki in jedrnati, tako kot pri zgodbah.

Komponenti kartic, če nanje gledamo kot opomnik, ki je osnova za nadaljni pogovor:

- stavek ali dva o problemu, kot opomnik za nadaljni pogovor,
- zapiski oz. opombe o problemu, katerih rešitve se razrešijo na sestanku.

Podrobnosti o zgodbi, ki se jih razreši na sestanku skozi pogovor, postanejo testi. Teste se zapiše na zadnjo stran kartice. Testi so samo druga oblika podrobnosti o sami zgodbi in nam povedo, kdaj nek sistem deluje kot pričakovano.

4.3.3 Pomembnost za uporabnika oz. naročnika

Pomembno je, da imamo zgodbo, ki je pomembna za uporabnika ali za naročnika. Izogibati se moramo zgodb, ki so pomembne za razvijalce in naročnikom ter uporabnikom ne povedo ničesar.

Primer take zgodbe:

- Vsi podatki so shranjeni v mySQL podatkovni bazi.

Take zgodbe je težko razvrstiti po prioriteti, zato jih je potrebno spremeniti ali odstraniti.

4.3.4 Predvidljiv obseg oz. kompleksnost

Pomembno je, da so razvijalci zmožni oceniti velikost zgodbe ali čas razvoja, ki je potreben, da zgodbo spremenimo v delujočo kodo.

Trije vzroki zakaj zgodbi ne moremo podati ocene:

- Razvijalcem primankuje znanja o domeni/problemu.
- Razvijalcem primanjkuje tehničnega znanja.
- Zgodba je prevelika.

Pri drugem vzroku gre za problem, ko razvijalci ne poznajo dovolj tehnologije, na kateri delajo. To privede do nezmožnosti za ocenitev zgodbe. V tem primeru razvojna skupina lahko definira kratek eksperiment (ang. spike), v okviru katerega pridobi potrebno znanje oziroma oblikuje ustrezno rešitev. Na ta način razvijalci dobijo ravno prav informacij, da so zmožni podati oceno o nalogi.

4.3.5 Majhnost

Velikost zgodbe je pomembna. Zgodba ne sme biti prevelika ali premajhna, saj jo težko uporabimo za planiranje. Z velikimi zgodbami je težko delati, ker pogosto vsebujejo več zgodb. Odločitev, ali je zgodba dovolj velika, je odvisna od ekipe, njenih zmožnosti in tehnologije.

4.3.6 Omogočajo testiranje

Zgodbe morajo biti napisane tako, da jih lahko testiramo. Uspešno izvedeni testi dokazujejo, da je zgodba uspešno razvita. Poleg tega so testi dober opomnik za programerje, kaj morajo še postoriti oz. kdaj lahko zaključijo s kodiranjem.

Obstajajo zgodbe, ki jih ni možno testirati. Kadar je možno, naj bodo testi avtomatizirani. Obstajajo tudi testi, ki jih ni moč izvesti.

Primer, imamo naslednjo zgodbo: *Uporabnik nikoli ne čaka predolgo na prikaz zaslona.* Take zgodbe ni možno testirati, zaradi besede *nikoli* in besede *predolgo*. Ne moremo definirati, kaj beseda predolgo pomeni. Prav tako je nemogoče dokazati, da se nekaj ne zgodi nikoli. Boljši način bi bil, da dokazujemo, da se nekaj dogaja redko.

4.4 Ocenjevanje uporabniških zgodb

Eno od pomembnejših vprašanj, ki se nam pojavi, ko delamo na projektu, je, koliko časa bomo porabili zanj. Da bi kar čimbolj učinkovito ocenili čas razvoja, moramo oceniti, koliko časa bomo porabili za kodiranje posameznih zgodb. Uporabniške zgodbe ocenjujemo z uporabniškimi točkami. Lepa lastnost uporabniških točk je, da lahko vsaka skupina sama definira njihov pomen. Najboljša praksa je, da se točko vzame kot idealen dan (brez sestankov, elektronske pošte, telefonov). Redko imamo take dneve [3].

Ocenjevanje zgodb se izvede s celotno ekipo. Izkaže se, da je tak pristop bolj učinkovit, kot če bi ocene podali posamezniki. Načinov ocenjevanja je več. Sledi opis tehnike ocenjevanja zgodb, ki se imenuje poker planiranje (ang. Planning Poker) [6]. Metodo je prvi opisal James Grenning.

Na sestanku se zberejo naročniška in razvojna skupina. Zraven prinesejo uporabniške zgodbe. Vsakemu razvijalcu se razdeli kup vnaprej pripravljenih kart z veljavnimi ocenami, npr.: 0, 1, 2, 3, 5, 8, 13, 20, 40 in 100. Naročnik izbere uporabniško zgodbo in jo prebere razvijalcem. Nato razvijalci postavljajo vprašanja o zgodbi. Naročnik odgovarja po svojih zmožnostih. Ko ni več

vprašanj, razvijalci ocenijo zgodbo in istočasno obrnejo karte. To je pomembno, saj ne želimo, da so ocene posameznika podane na podlagi ocen drugih članov. Najverjetneje se bodo ocene razlikovale. Tista dva, z največjo in najmanjšo oceno razložita, zakaj sta podala take ocene. Nato sledi pogovor celotne skupine in ponovni postopek ocenjevanja. Postopek se ponavlja toliko časa dokler ocene ne konvergirajo. Ponavadi je to že v drugem ali tretjem krogu. Ni nujno, da so ocene popolnoma enake. Če so, na primer, razvijalci podali naslednje ocene: 3, 3, 3, 2, se razvijalca, ki je podal najmanjšo oceno vpraša, če se strinja z oceno 3. Če se strinja, zaključimo z ocenjevanjem zgodbe. Pomembno je, da se vsi ocenjevalci dogovorijo glede končnega rezultata. Argumenti, zakaj smo podali tako oceno, so ključni za uspešno izvedbo tehnike.

4.5 Sprejemni testi uporabniških zgodb

Sprejemno testiranje je proces preverjanja, da sprogramirane zgodbe delujejo natanko tako, kakor si je zamislil naročnik izdelka. Ob začetku teka programerji začnejo programirati, naročniška skupina pa začne pisati teste. Testi naj bodo napisani čim preje v iteraciji. To je v veliko pomoč programerjem, da se jih spomni na zadeve, ki bi jih lahko morda pozabili.

Testiranje je proces, ki je sestavljen iz dveh korakov. Prvi korak je, da se zapiske testov zapiše na zadnjo stran kartice. V drugem koraku se zapisane teste realizira do te mere, da so zmožni pokazati, da je zgodba pravilno in v celoti skodirana.

Za zgodbo *Podjetje lahko za objavo dela plača s kreditno kartico*, bi na zadnji strani kartice lahko pisalo [3]:

- Preveri plačilo s karticami Visa, MasterCard in American Express (test mora uspeti).
- Preveri plačilo s kartico Diners Club (test mora pasti).
- Preveri plačilo z veljavnimi, neveljavnimi številkami kartic.
- Preveri s kartico, ki ji je potekla veljavnost.
- Preveri plačilo z različnimi zneski (tudi tistimi nad limitom).

Ti zapiski testov so predpostavke, narejene s strani naročnika.

Poznamo veliko načinov testiranja. Naloga naročnika in razvojne skupine je, da razmislijo kateri način testiranja je najbolj primeren za trenutni problem. Za večino sistemov je najbolj v uporabi testiranje zgodb (funkcionalno testiranje, ki zagotovi pravilno delovanje funkcij aplikacije). Seveda obstajajo tudi drugi načini, vsak za svoj domenski problem [3]:

- Testiranje uporabniških vmesnikov.
- Testiranje uporabnosti.
- Testiranje zmogljivosti.
- Stresno testiranje (aplikacijo se podvrže ekstremnim razmeram, npr. veliko število uporabnikov, transakcij, itd.).

4.5.1 Pisanje testov

Priporočljivo je, da so sprejemni testi spisani pred začetkom kodiranja zgodbe, na začetku vsakega teka. Ponavadi so testi napisani, ko pride do pogovora med naročnikom in razvojno skupino o zgodbi in njenih podrobnostih. Ker se velikokrat zgodi, da se nove teste odkrije ravno med kodiranjem, so lahko testi napisani tudi po kodiranju.

Na začetku vsakega teka bi moral iti naročnik skozi zgodbe in napisati dodatne teste, ki se jih spomni. Dober način, da to stori je, da pogleda vsako zgodbo in si postavi naslednja vprašanja:

- Kaj morajo programerji še vedeti o tej zgodbi?
- Kaj jaz mislim o tem, kako naj bo zgodba implementirana?
- Ali obstajajo okoliščine, ko se ta zgodba lahko obnaša različno?
- Kaj gre lahko narobe med samo zgodbo?

Pomembno je, da je testiranje del razvojnega procesa, ne nekaj, kar je storjeno po koncu kodiranja.

Naročnik naj piše teste le tako dolgo, dokler še dajejo vrednost in pojasnilo sami zgodbi. Dobra razvojna skupina ima že spisane teste za večino nizkonivojskih primerov, na primer test, ki zazna 30. februar in 31. junij, kot nepravilna datuma. Naročnik ni odgovoren za identificiranje vsakega testa. Naročnik naj piše samo teste, ki razjasnijo bistvo zgodbe. Namenjene naj bodo razvijalcem.

Ker je programska oprema napisana, da izpolni vizijo naročnika, je prav da nekaj osnovnih sprejemnih testov določi naročnik sam. Pri tem lahko sodeluje z

programerjem ali testerjem. Razvojna skupina lahko obogati zgodbe s svojimi testi.

4.5.2 Orodja za integrirane teste

Sprejemni testi so namenjeni za prikaz, da je aplikacija dosegla zahteve, ki so bile podane s strani lastnika izdelka. Lastnik izdelka je tisti, ki naj bi izvršil sprejemne teste na koncu vsake iteracije. Na koncu vsake iteracije je potrebno izvršiti ne samo trenutni test, ampak vse teste do prvega nazaj. Ker je to lahko časovno potratno, je potrebno teste avtomatizirati.

Odlični orodji za avtomatiziranje sprejemnih testov sta Ward Cunningham's Framework for Integrated Test (FIT) in FitNesse, katerega avtorja sta Bob in Micah Martin. Drugo orodje je nadgradnja prvega. FitNesse je hitro postal zelo popularno orodje za pisanje sprejemnih testov na agilnih projektih.

4.6 Naročniška skupina

Pri agilni metodologiji Scrum imamo samo eno osebo, ki skrbi za seznam zahtev ter določitev prioritete posameznih zahtev [2]. Ko v proces metodologije Scrum vpeljemo uporabniške zgodbe s tem na projekt vključimo tudi uporabnike same ali njihove predstavnike (ang. user proxies), lahko pa tudi testerje ter upravitelje projekta. Taki skupini pravimo naročniška skupina (ang. Customer Team). Njihova naloga je zadovoljiti bodoče uporabnike z dostavo kvalitetne programske opreme.

4.7 Planiranje izdaj

Pri planiranju izdaj gre za določitev časa razvoja, v katerem bo možno izdelati aplikacijo z željenimi funkcionalnostmi. Večina projektov naj bi imelo čim bolj pogoste izdaje. To je pomembno, saj s tem dobimo pogoste odzive uporabnikov, kaj jim je pri programu všeč in kaj jim ni. Sodelujoči na projektu dobijo pregled nad tem, kaj bo potrebno spremeniti do naslednje izdaje. Če seveda želimo splanirati, koliko tednov ali mesecev bo preteklo do izdaje, mora naročnik najprej prioritizirati zgodbe. Pri tem se pogosto uporablja tehnika, vzeta iz metodologije DSDM - Dynamic System Development Method. Več o metodologiji DSDM v knjigi DSDM:Business Focused Development (Stapleton 2003) [3].

Tehnika deluje po pravilih MoSCoW. MoSCoW je akronim za:

- Must have - Obvezno.
- Should have - Potrebno.
- Could have - Zaželeno.
- Won't have this time - Nepotrebno v tej izdaji.

Naročnik ne more prioritizirati zgodb, če nima potrebnih informacij, ki mu jih priskrbijo razvijalci. Najmanj, kar mora naročnik vedeti o zgodbi je, koliko časa bo potrebnega za njeno implementacijo. Velikokrat je želja razvijalcev, da se najprej naredijo težje zgodbe. Končna odločitev mora priti s strani naročnika. Potem, ko je naročnik prioritiziral vse zgodbe, skupina sešteje točke vseh zgodb. Če želimo ugotoviti, koliko tekov bomo potrebovali, za implementacijo teh točk, moramo vpeljati še drugo mero ocenjevanja, ki ji pravimo hitrost razvoja (ang. velocity) [3]. Hitrost je skupna vsota vseh točk uporabniških zgodb, ki jih skupina konča v eni iteraciji. Za določitev hitrosti obstajajo trije načini: uporabimo zgodovinske podatke, izmerimo po prvi iteraciji, ali pa si hitrost izmislimo. Hitrost in točke se nato uporabi za napoved trajanja projekta. Poleg tega razvojna skupina z naročnikom izbere dolžino teka (1 - 4 tedni). Sledi opis dveh možnih scenarijev, kako zgodbe porazdeliti v posamezne teke.

$$število\ tekov = vsota\ vseh\ točk / hitrost \tag{4.1}$$

$$čas\ razvoja = število\ tekov * dolžina\ enega\ teka \tag{4.2}$$

4.7.1 1. scenarij

Recimo, da razvojna skupina skupaj z naročnikom izbere, da naj bo dolžina enega teka 2 tedna. Vsota vseh točk je 100. Skupina oceni hitrost na 20. Hitrost se določi na tri načine, in sicer z uporabo zgodovinske vrednosti, izvedbo začetnega teka in uporabo njene hitrosti ali z ugibanjem. Hitrost nam pove število končanih točk v eni iteraciji. Po enačbi 4.1 izračunajo število tekov, in dobijo 5. Nato naročnik z razvojno skupino izbere 20 točk vredne najvišje prioritizirane zgodbe in jih razporedijo v prvi tek. Postopek ponovijo še za ostale teke. Sedaj lahko po formuli 4.2 izračunajo predviden čas celotnega razvoja in dobijo rezultat potrebnega časa 10 tednov.

4.7.2 2. scenarij

Naročnik želi, da je projekt končan v 5 tednih. Skupaj z razvojno skupino se odločijo, da bo dolžina enega teka 1 teden. Vsota vseh zgodbovnih točk je 100. Po formuli 4.2 izračunajo, da je število možnih tekov 5. Nato po formuli 4.1 dobijo hitrost, ki jim pove, da morajo implementirati 20 točk na tek. Začne se postopek razporeditve zgodb, kot pri scenariju 1.

4.8 Planiranje tekov

Za planiranje vsakega teka se zbere celotna ekipa na sestanku planiranja tekov. Sestanek je sestavljen iz dveh delov. V prvem delu gre za izbiranje uporabniških zgodb, ki bodo vključene v trenutno iteracijo, v drugem delu sledi razbitje zgodbe na manjše naloge.

Drugi del sestanka poteka na sledeč način. Naročnik prebere zgodbo z največjo prioriteto ostalim razvijalcem. Ti nato postavljajo vprašanja, dokler zgodbe ne razumejo v celoti. Naročnik jim ponudi odgovore. Sledi razbitje zgodbe na manjše naloge. Nato razvijalci prevzamejo odgovornosti za naloge. Priporočljivo je, da so naloge in imena razvijalcev napisana na beli tabli. Postopek se ponovi, dokler niso vse zgodbe obdelane in sprejete. Razvijalci individualno ocenijo naloge, ki so jih dobili.

Razbitje zgodb je pomembno iz dveh vidikov. Prvi je, da v večini primerov zgodba ne bo implementirana zgolj samo z enim razvijalcem, ampak bo razdeljena med razvijalce, ali zaradi specializacije na določenih tehnologijah ali pa zaradi hitrosti dokončanja naloge. Drugi vidik, zakaj razbiti zgodbo je, da programerjem izpostavimo naloge, ki jih morajo postoriti in bi jih morda lahko pozabili.

Poglavje 5

Opis orodja FitNesse

5.1 Kaj je FitNesse?

FitNesse [4] je odprtokodno avtomatizirano ogrodje (ang. framework), ustvarjeno z namenom testiranja programske opreme. Program spodbuja sodelovanje pri razvoju programske opreme, saj vsebuje strežnik z vgrajenim Wiki sistemom, ki naročniku, testerjem in programerjem omogoča enostavno dodajanje in urejanje testov na način, neodvisen od platforme. FitNesse je osnovan na Ward Cunningham-ovem ogrodju za integrirane teste (FIT)[4].

FitNesse omogoča naročniku, testerjem ter programerjem, da se naučijo, kaj mora njihov program narediti in to avtomatsko primerjati s tem, kar program dejansko naredi. FitNesse primerja naročnikova pričakovanja z dejanskimi rezultati.

FitNesse je zasnovan tako, da podpira funkcionalno testiranje (poznano tudi kot sprejemno testiranje).

Če povzamem. FitNesse je:

- orodje, ki podpira sodelovanje pri razvoju programske opreme,
- orodje testiranja programske opreme,
- wiki (enostavno dodajanje in urejanje strani),
- spletni strežnik.

5.2 Arhitektura ogrodja FitNesse

FitNesse deluje tako, da izvaja Wiki strani, na katerih so testne tabele. FitNesse nato vzame tabelo in zažene testno okolje (ang. fixture), ki ustreza imenu tabele. Testno okolje nato pokliče ustrezen del aplikacije s podatki iz tabele in vrne rezultate. Testna okolja so most med Wiki stranmi in sistemom, ki ga testiramo (System Under Test - SUT). Ta testna okolja lahko spišemo v več programskih jezikih.

Trenutno so podprti naslednji jeziki:

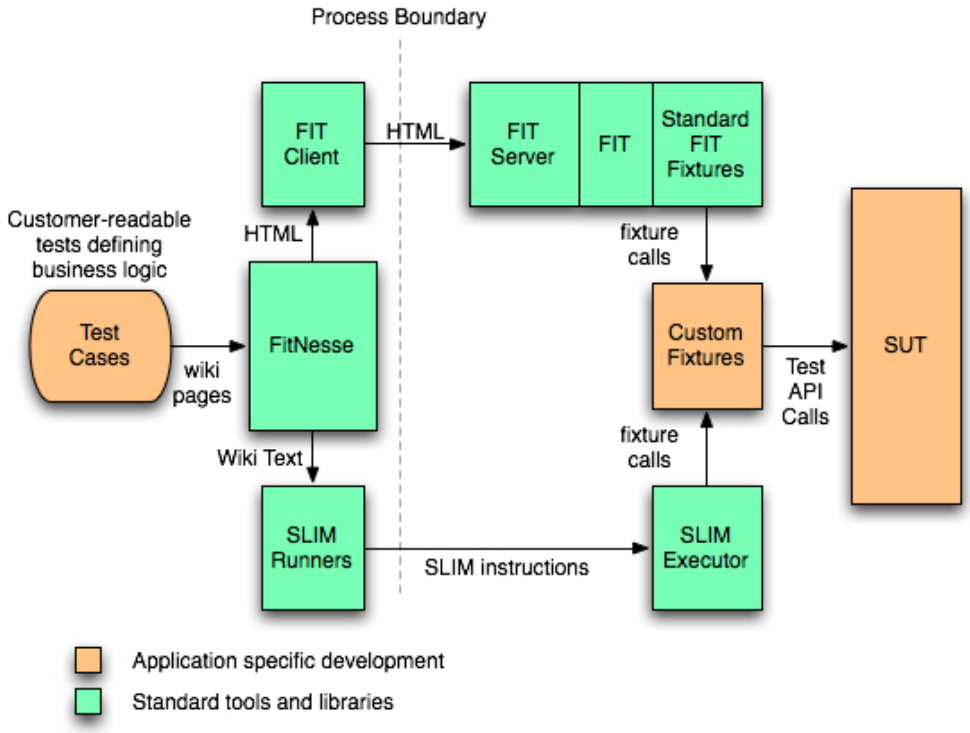
- Java,
- .NET,
- C++,
- Delphi,
- Python,
- Ruby,
- Smalltalk,
- Perl.

FitNesse ima dva sistema za testiranje, SLIM in FIT. FIT je starejši testni sistem in ni več aktivno v razvoju. Zaradi kompleksnosti vzdrževanja in podpore FIT na različnih platformah je bil ustvarjen SLIM. SLIM je lahka različica FIT protokola. Eden od glavnih ciljev oblikovanja SLIM je bil enostaven prenos implementacij za različne jezike. Poleg tega v nasprotju s FIT, SLIM omogoča lažje pisanje testnih okolij.

Na sliki 5.1 je prikazana arhitektura orodja FitNesse. Iz nje je razvidno, da je FitNesse sestavljen iz dveh protokolov, zgornja veja predstavlja protokol FIT, spodnja protokol SLIM. Protokol FIT deluje tako, da ko mi poženemo naš test, aktiviramo strežnik FIT, ki od odjemalca FIT prejme HTML testno stran. Strežnik FIT nato razčleni HTML testno stran in pokliče ustrezno testno okolje. Testno okolje je odgovorno za izvršitev *Parse* drevesa ter interpretacijo rezultatov.

Protokol SLIM deluje nekoliko drugače. Začne se podobno, s to razliko, da se HTML testno stran pošlje tekačem SLIM, ki Wiki strani pretvori v

navodila SLIM (ki niso nič drugega kot funkcijski klici). Izvršitelj SLIM nato prejme ta navodila SLIM oz. ukaze, si jih interpretira in pokliče ustrezno testno okolje. Vsa procesiranja tabel, primerjave in razčlenjevanja so storjena na strani tekačev SLIM. Zaradi tega vse lastnosti protokola SLIM ostanejo enake, ne glede na to, katero platformo izberemo. FitNesse je odgovoren za interpretacijo rezultatov.



Slika 5.1: FitNesse arhitektura.

5.3 Postopek namestitve FitNessa

Koraki namestitve FitNessa:

1. Povleci zadnjo verzijo iz

   ```
   http://www.fitnesse.org/FrontPage.FitNesseDevelopment.DownLoad
   ```

2. Poženi

   ```
   java -jar fitnesse.jar
   ```

3. Opomba: če se FitNesse na portu 80 ne zažene, poskušaj pognati

   ```
   java -jar fitnesse.jar -p 9090
   ```

4. Usmeri spletni brskalnik na `http:/localhost:≪port-number≫`, za prikaz prve strani FitNesse-a.

Nekatere FitNesse opcije ukazne vrstice:

```
Usage: java -jar fitnesse.jar [-pdrleoa]
        -p <port number> {80} or {9123 if -c}
        -d <working directory> {.}
        -r <page root directory> {FitNesseRoot}
        -l <log directory> {no logging}
        -e <days> {14} Number of days before page versions expire
        -o omit updates
        -a {user:pwd | user-file-name} enable authentication.
        -i Install only, do not run fitnesse after install.
        -c <command> Run the command (same as restful url) and then exit.
         Return status is the number of test pages that failed
         as a result of the command.
```

5.4 Testi in zbirka testov

FitNesse ima koncept testov (ang. tests) in zbirke testov (ang. suites). Teste organiziramo v zbirke testov. Ko izvršimo zbirko testov, izvršimo vse teste znotraj zbirke.

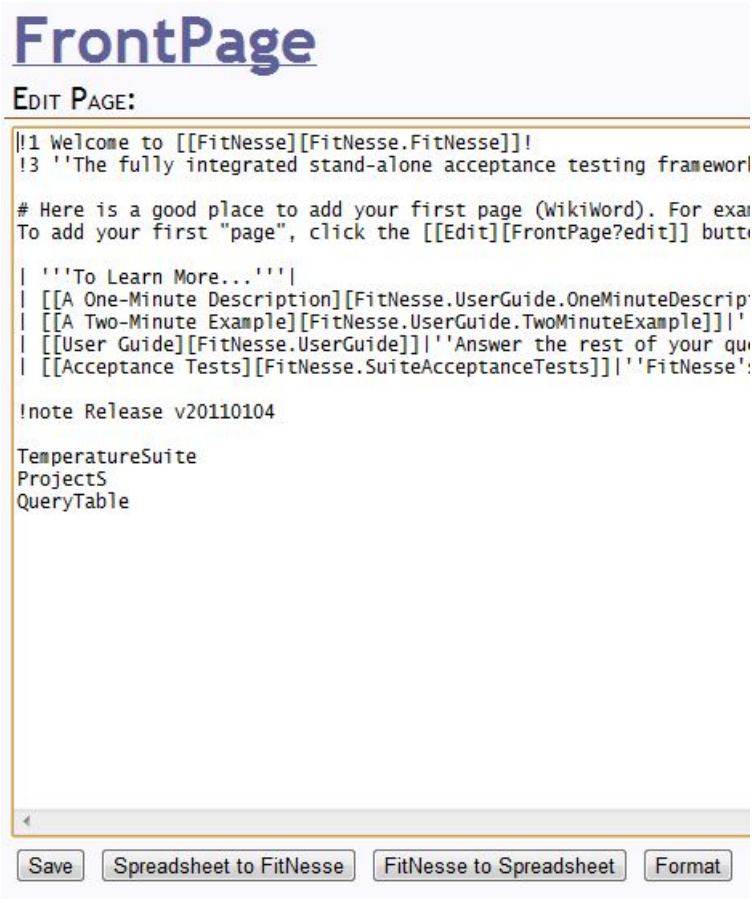
5.4.1 Kreiranje zbirke testov

Koraki kreiranja nove FitNesse zbirke testov:

1. Pojdi na prvo stran `http://localhost:9090`.
2. Klikni povezavo *Edit* v levem meniju. FitNesse meni prikazan na sliki 5.2.
3. Vnesi ime zbirke testov na konec polja z besedilom, npr. TemperatureSuite. Vsebine polja ne briši. Ime zbirke naj se začne z veliko začetnico. Prikaz polja z besedilom in imeni zbirk na sliki 5.3.
4. Klikni na gumb *Save*.
5. Klikni na vprašaj [?] ob imenu zbirke na prvi FitNesse strani.
6. Pritisni gumb *Save*. Da bo naša stran zbirka, moramo v *Properties* izbrati tip strani *Suite*.

Kreira se prazna zbirka testov. To nam pove povezava *Suite* v levem meniju zgoraj. Če želimo priti na prvo stran, kliknemo na *Front Page* povezavo, ki se nahaja v nogi strani.

Slika 5.2: Gumb *Edit* na meniju levo.



Slika 5.3: Imena kreiranih zbirk testov.

5.4.2 Kreiranje novega FitNesse testa

Po uspešno ustvarjeni zbirki testov želimo kreirati svoj prvi test. Na primer, da želimo napisati funkcijo, ki pretvori temperaturo iz Celzijev v Fahrenheite. Zaradi lažjega razvoja želimo imeti test, ki nam bo pokazal, kdaj smo uspešno razvili želeno funkcijo. Pri testu si bomo pomagali z tabelo `Decision`, ki je bolj podrobno opisana v naslednjih poglavjih.

Koraki kreiranja testa so podobni tistemu za kreiranje nove zbirke testov:

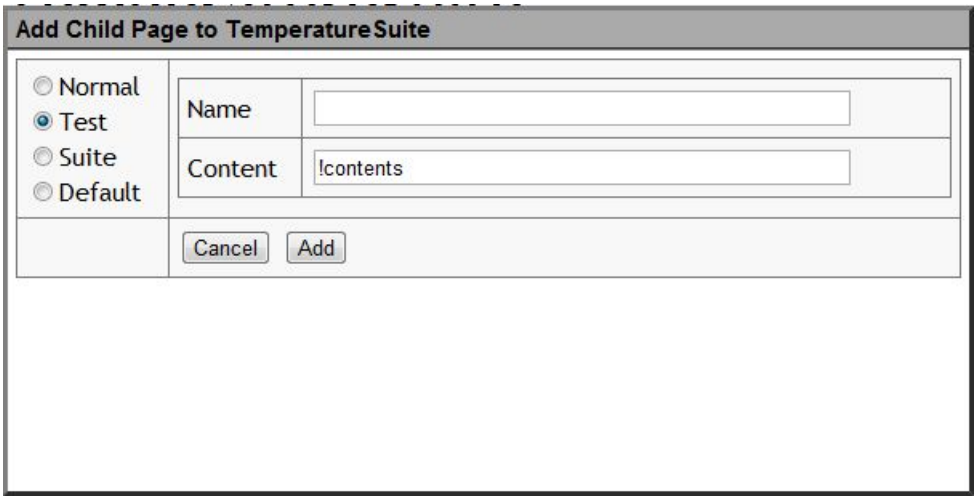
1. Ko si znotraj zbirke testov, klikni na *add child*, ki se nahaja ob imenu zbirke zgoraj.
2. V polje *Name* vnesi ime testa, npr. TemperatureTest in izberi *Test* na levi strani. Slika 5.4 prikazuje obrazec za kreiranje novega testa.

3. Pritisni gumb *Add*.

4. Pod *Contents* izberi pravkar kreiran test in vanj vstavi naslednje besedilo:

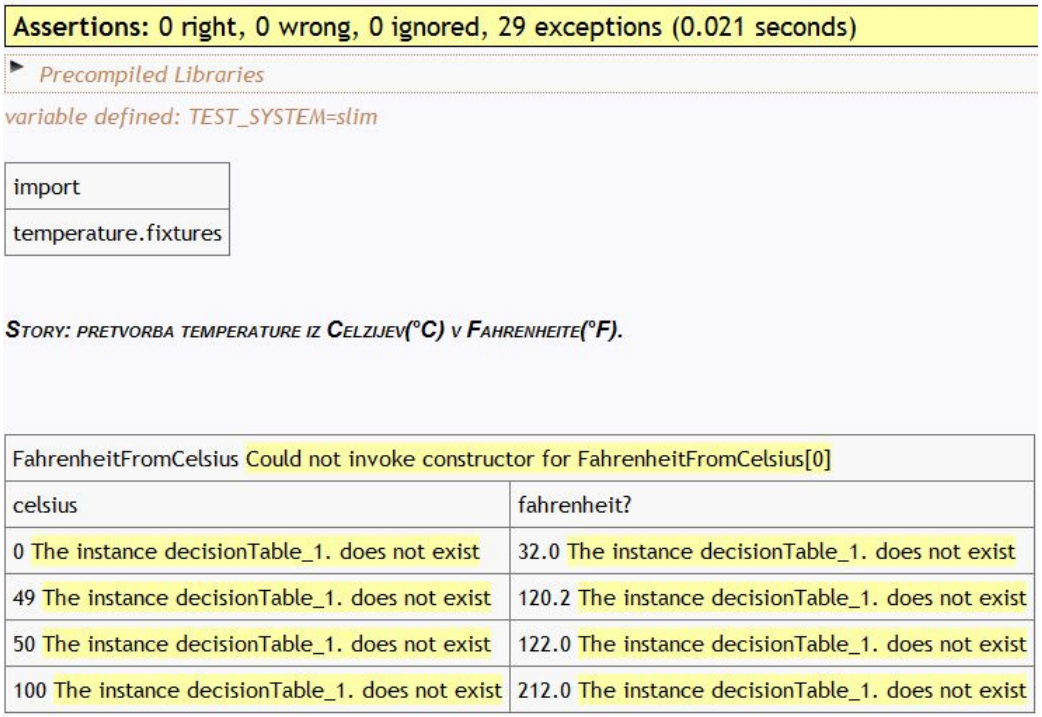
```
!4 Story: pretvorba temperature iz Celzijev v Fahrenheite.

!|fahrenheit from celsius|
|celsius|fahrenheit?|
|0      |32.0  |
|49     |120.2 |
|50     |122.0 |
|100    |212.0 |
```

5. Pritisni gumb *Save*.



Slika 5.4: Dodajanje novega testa.

Če pritisnemo *Test* v levem meniju, vidimo da nam izpiše napako, prikazano na sliki 5.5. Do napake pride iz dveh razlogov. FitNessu nismo povedali kje lahko najde testno okolje, ali pa naše testno okolje ne obstaja. Da bo testiranje delovalo moramo postoriti še dve stvari: napisati testno okolje in pravilno konfigurirati FitNesse.



Assertions: 0 right, 0 wrong, 0 ignored, 29 exceptions (0.021 seconds)

Precompiled Libraries

variable defined: TEST_SYSTEM=slim

import
temperature.fixtures

Story: pretvorba temperature iz Celzijev(°C) v Fahrenheite(°F).

FahrenheitFromCelsius Could not invoke constructor for FahrenheitFromCelsius[0]	
celsius	fahrenheit?
0 The instance decisionTable_1. does not exist	32.0 The instance decisionTable_1. does not exist
49 The instance decisionTable_1. does not exist	120.2 The instance decisionTable_1. does not exist
50 The instance decisionTable_1. does not exist	122.0 The instance decisionTable_1. does not exist
100 The instance decisionTable_1. does not exist	212.0 The instance decisionTable_1. does not exist

Slika 5.5: Napaka testa TemperatureTest.

5.5 Primer kode

Testno okolje je plast med produkcijsko kodo in FitNesse stranjo. Poznamo več vrst testnih okolij, za vsako testno tabelo, drugačno predlogo kode. Za spodnjo SUT kodo bomo potrebovali testno okolje odločitvene tabele. Na primer, da želimo stestirati naslednjo funkcijo:

```
package temperature.sut;

public class Temperature {
    public double convertCelsiusToFahrenheit(double celsius) {
        return 1.8 * celsius + 32;
    }
}
\end{verbatim}

Testno okolje za testiranje tega razreda izgleda takole:
\begin{verbatim}
package temperature.fixtures;

import temperature.sut.Temperature;
```

```
public class FahrenheitFromCelsius {
    private double celsius;
    private double fahrenheit;

    public void setCelsius(double celsius) { // setter method
        this.celsius = celsius;
    }

    public double fahrenheit() { // returning function because of question
        return this.fahrenheit;
    }

    public void execute() { // executed after each table row
        this.fahrenheit =
        new Temperature().convertCelsiusToFahrenheit(celsius);
    }
}
```

Testno okolje je klicano iz FitNesse strani in za vsako vrstico tabele se izvede naslednje zaporedje dogodkov:

1. Pokličejo se metode *setter*.
2. Pokliče se funkcija *execute*, ki pokliče sistem, ki ga želimo testirati.
3. Testno okolje vrne rezultat in ga primerja s pričakovanim rezultatom.

5.6 Konfiguracija FitNessa

Če želimo namesto protokola FIT, uporabiti protokol SLIM, moramo na vrhu testne strani dodati naslednjo vrstico:

```
!define TEST_SYSTEM {slim}
```

Za pravilno delovanje testiranja moramo postoriti še dve stvari: nastaviti pot do projekta in FitNessu povedati, v katerem paketu lahko najde testno okolje. Za nastavitev poti na testno stran dodamo naslednjo vrstico:

```
!path <your-classpath-here>
```

V našem primeru bi ta izgledala takole:

```
!path C:\Development\CelsiusToFahrenheit\dist\CelsiusToFahrenheit.jar
```

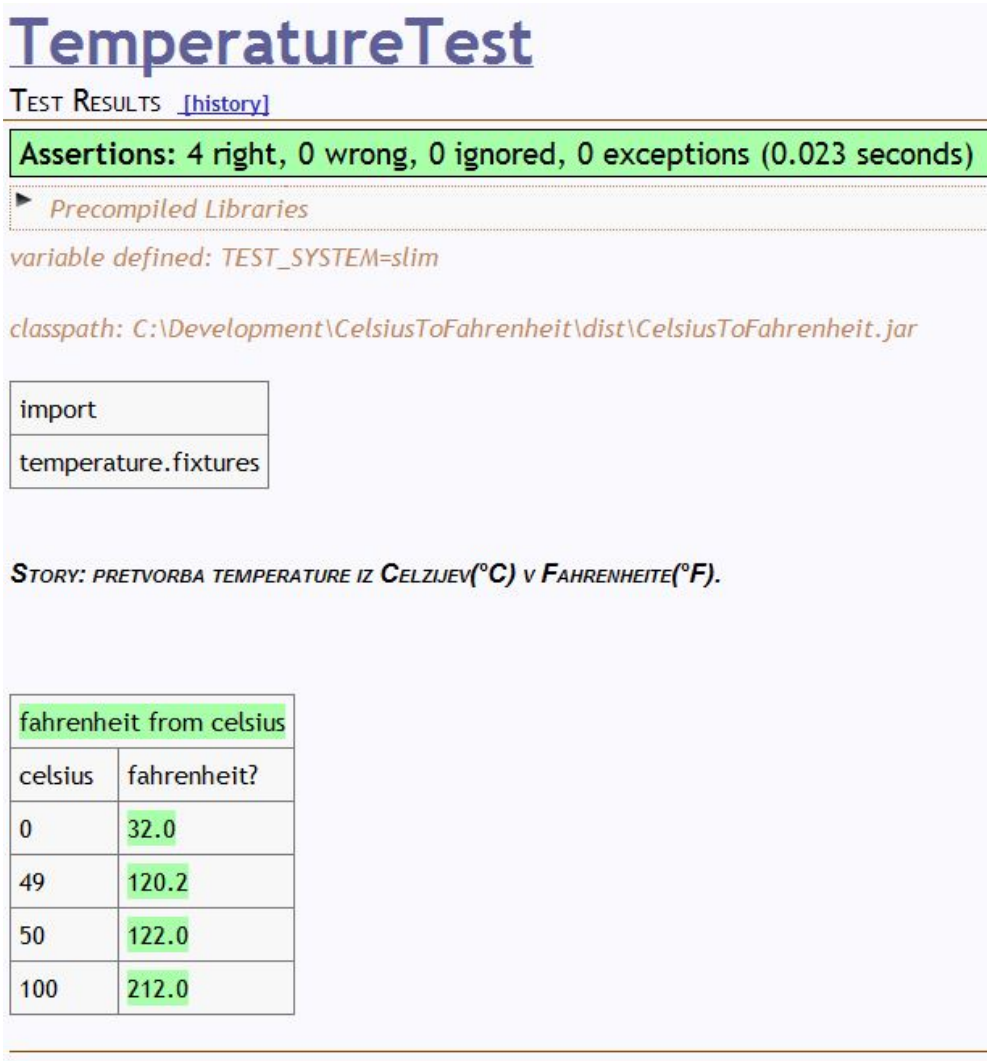
Da FitNessu povemo v katerem paketu najde testno okolje, dodamo na testno stran naslednjo kodo:

```
|import|
|temperature.fixtures|
```

Celotna testna stran mora izgledati takole:

```
!define TEST_SYSTEM {slim}

!path C:\Development\CelsiusToFahrenheit\dist\CelsiusToFahrenheit.jar

|import|
|temperature.fixtures|

!4 Story: pretvorba temperature iz Celzijev v Fahrenheite.

!|fahrenheit from celsius|
|celsius|fahrenheit?|
|0      |32.0  |
|49     |120.2 |
|50     |122.0 |
|100    |212.0 |
```

Ko pritisnemo gumb *T*est v levem meniju, testno okolje pokliče del aplikacije, ki ga želimo testirati. Ta nato obarva Wiki stran glede na testne rezultate. Če so rezultati pravilni, se ti obarvajo zeleno. Rezultati testa prikazani na sliki 5.6.



TemperatureTest

TEST RESULTS [history]

Assertions: 4 right, 0 wrong, 0 ignored, 0 exceptions (0.023 seconds)

▸ *Precompiled Libraries*

variable defined: TEST_SYSTEM=slim

classpath: C:\Development\CelsiusToFahrenheit\dist\CelsiusToFahrenheit.jar

import
temperature.fixtures

STORY: PRETVORBA TEMPERATURE IZ CELZIJEV(°C) V FAHRENHEITE(°F).

fahrenheit from celsius	
celsius	fahrenheit?
0	32.0
49	120.2
50	122.0
100	212.0

Slika 5.6: Rezultati testa tabele `Decision`.

5.7 Vrste testnih tabel SLIM

Prva celica SLIM tabele nam pove kakšne vrste je tabela. Vrste testnih tabel SLIM so podane v tabeli 5.1

Ime tabele	Opis
`Decision`	Tabela pričakovanih izhodov glede na podane vhode.
`Query`	Tabela pričakovanih rezultatov poizvedbe.
`Subset Query`	Tabela podmnožic pričakovanega rezultata poizvedbe.
`Ordered query`	Tabela pričakovanih rezultatov poizvedbe. Vrstice so urejene.
`Script`	Tabela s katero izvedemo serijo akcij in preverjanj.
`Scenario`	Tabela. ki jo lahko kličemo iz drugih tabel.
`Table`	Zelo fleksibilna tabela, ki jo lahko uporabimo za praktično vse.
`Import`	Dodamo pot do razreda testnega okolja.
`Comment`	Ta tabela se ne izvede. Služi kot komentar.
`Library`	Tabela, ki naloži razpoložljiva testna okolja za vse testne strani.

Tabela 5.1: Vrste SLIM testnih tabel.

5.7.1 Tabela `Decision`

Tabela `Decision` je privzeta tabela pri FitNessu. To pomeni, da kadar ne določimo predpone, FitNesse obravnava tabelo kot odločitveno. Kot alternativo lahko pred imenom testnega okolja v testni tabeli zapišemo predpone *decision:* ali *dt:*. Spodnji primer bi izgledal takole: *decision: fahrenheit from celsius*. Primer tabele `Decision`:

```
!4 Story: pretvorba temperature iz Celzijev v Fahrenheite.

!|fahrenheit from celsius|
|celsius|fahrenheit?|
|0      |32.0  |
|49     |120.2 |
|50     |122.0 |
|100    |212.0 |
```

FitNesse ignorira celotno besedilo izven tabele. Besedilo služi samo kot dokumentacija. Klicaj (!) v prvi vrsti preprečuje FitNessu, da bi *Camel Case* besede obravnaval kot povezave strani.

Zaglavje tabele je sestavljeno iz dveh imen, *celsius* in *fahrenheit*. Vsako zaglavje ustreza eni *set* funkciji. Če zaglavje vsebuje vprašaj (?) se vrednosti v tem stolpcu obravnavajo kot izhodi funkcije. Vrnjene vrednosti funkcije *fahrenheit* so primerjane z vrednostmi v tabeli. Celica se obarva zeleno, če je primerjava uspešna in rdeče, če ni.

Ko poženemo ta test, FitNesse poišče razred testnega okolja z imenom `FahrenheitFromCelsius`.
FitNesse bo za vsako vrstico poklical metodo `setCelsius` iz razreda testnega okolja. Po vseh klicih *set* funkcije, FitNesse izvrši `execute` metodo, ki naredi vse delo. Za tem se pokliče `fahrenheit` funkcija, ki prikaže rezultate. Primer testnega okolja za zgornjo tabelo v podpoglavju 5.5.

Tabela `Decision` ima opcijo implementirati še nekatere druge funkcije, ki so klicane, če so definirane v testnem okolju. Funkcije so prikazane v tabeli 5.2.

Funkcija	Opis
reset	Funkcija, ki je klicana za vsako vrstico preden so klicane *set* ali izhodne funkcije.
execute	Funkcija, ki je klicana za vsako vrstico po tem ko so klicane vse *set* funkcije, vendar preden so klicane izhodne funkcije.
table	Funkcija je klicana po izvedbi konstruktorja in preden so procesirane vse vrstice. Sprejme celotno tabelo v obliki strukture seznam seznamov, ki vsebuje vse celice z izjemo prve vrstice.

Tabela 5.2: Dodatne funkcije tabele `Decision`.

5.7.2 Tabela `Query`

Tabela `Query` se uporablja zato, da z njo testiramo, da neka poizvedba deluje, kot smo si zamislili. FitNesse pozna tri vrste tabel `Query`, ki so skoraj identične, z majhnimi izjemami.

Tabele `Query`	Opis
Query	Standardna povpraševalna tabela, ki primerja kompletno množico podatkov na neurejen način.
Subset query	Povpraševalna tabela, ki primerja le podmnožico podatkov z rezultati testnega okolja.
Ordered query	Zaporedje vrstic v tabeli mora biti enako zaporedju rezultatov, ki jih vrne funkcija testnega okolja.

Tabela 5.3: Tri vrste tabel `Query`.

Vrstice tabele `Query` predstavljajo pričakovane rezultate povpraševalne funkcije.

Na primer, da imamo tako tabelo `Query`:

```
|Query:employees hired before|10-Dec-1980                    |
|employee number             |first name|last name|hire date  |
|1429                        |Bob       |Martin   |10-Oct-1975|
|9924                        |Bill      |Mitchell |19-Dec-1966|
```

Lahko tudi pustimo celice prazne in s tem dovolimo, da se jih zapolni:

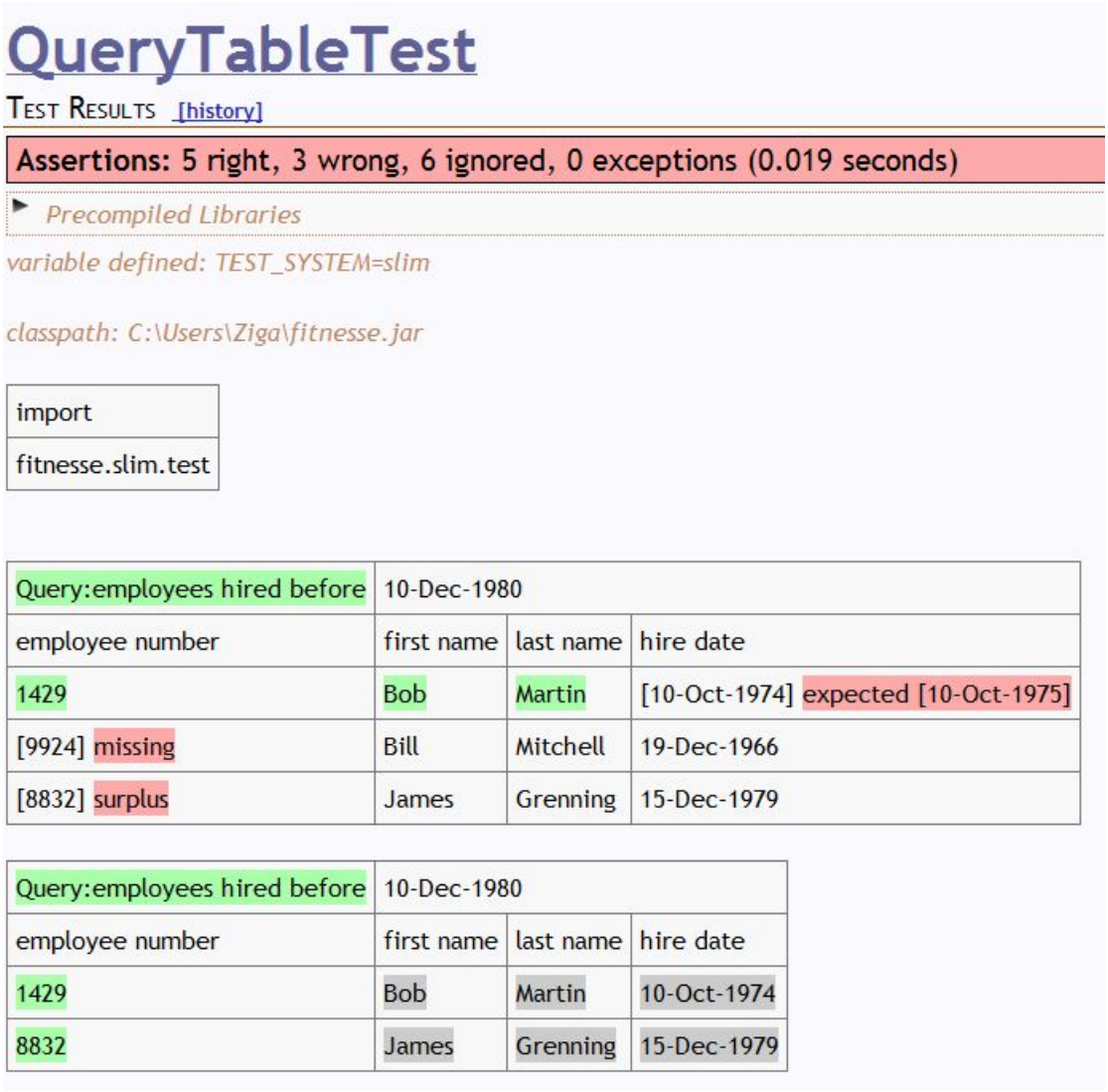
```
|Query:employees hired before|10-Dec-1980                  |
|employee number             |first name|last name|hire date|
|1429                        |          |         |         |
|8832                        |          |         |         |
```

Koda testnega okolja bi bila naslednja:

```
package fitnesse.slim.test;

import static fitnesse.util.ListUtility.list;

import java.util.Date;
import java.util.List;

public class EmployeesHiredBefore {
  private Date date;

  public EmployeesHiredBefore(Date date) {
    this.date = date;
```

```
  }

  public void table(List<List<String>> table) {
    //optional function
  }

  public List<Object> query() {

    return
      list(
        list(
          list("employee number", "1429"),
          list("first name", "Bob"),
          list("last name", "Martin"),
          list("hire date", "10-Oct-1974")
        ),
        list(
          list("employee number", "8832"),
          list("first name", "James"),
          list("last name", "Grenning"),
          list("hire date", "15-Dec-1979")
        )
      );
  }
}
```

Če poženemo test za zgornje podatke, dobimo rezultate, prikazane na sliki 5.7.



QueryTableTest

TEST RESULTS [history]

Assertions: 5 right, 3 wrong, 6 ignored, 0 exceptions (0.019 seconds)

▶ *Precompiled Libraries*

variable defined: TEST_SYSTEM=slim

classpath: C:\Users\Ziga\fitnesse.jar

import
fitnesse.slim.test

Query:employees hired before	10-Dec-1980		
employee number	first name	last name	hire date
1429	Bob	Martin	[10-Oct-1974] expected [10-Oct-1975]
[9924] missing	Bill	Mitchell	19-Dec-1966
[8832] surplus	James	Grenning	15-Dec-1979

Query:employees hired before	10-Dec-1980		
employee number	first name	last name	hire date
1429	Bob	Martin	10-Oct-1974
8832	James	Grenning	15-Dec-1979

Slika 5.7: Rezultati testa tabele `Query`.

Iz slike 5.7 je razvidno, da je test prinesel 5 pravilnih rezultatov, 3 nepravilne rezultate in 6 ignoriranih rezultatov. Najbolj so zanimivi nepravilni rezultati. Prvi nam pove, da je bil pričakovani datum zaposlitve za osebo Bob Martin, 10. oktober 1975. Pravi datum je 10. oktober 1974. Drugi nepravilen rezultat, ki je označen z besedo *missing*, nam pove, da oseba Bill Mitchell ne obstaja v naši podatkovni strukturi, čeprav smo pričakovali, da je. Pri tretjem nepravilnem rezultatu nismo pričakovali James Grenninga, čeprav obstaja v

podatkovni strukturi. Sive celice predstavljajo polja, ki so napolnjena s podatki oseb, zaposlenih pred 10. decembrom 1980 in s številkami zaposlenega 1429 in 8832.

5.7.3 Tabela `Script`

Tabele `Script` so najbolj fleksibilne tabele in se jih uporablja predvsem za testiranje zgodb ali scenarijev. Pri tabelah `Script` se vsaka vrstica nanaša na metodo testnega okolja ali na predhodno definiran scenarij. Pred vsakim stavkom v testni tabeli lahko dodamo po eno od predpon opisanih v tabeli 5.4.

check	Za besedo *check* sledi klic funkcije v drugi celici, v tretji pa pričakovana vrednost te funkcije.
check not	Deluje podobno kot zgornja beseda, le da se pričakuje vse vrednosti razen vrednosti, ki je navedena v tretji celici.
ensure	Za besedo *ensure* sledi klic funkcije, ki vrne boolean vrednost (zelena barva za true, rdeča za false).
reject	Deluje obratno kot *ensure* (zelena barva za false, rdeča barva za true).
note	Uporablja se za komentarje. Vse druge celice v vrstici so ignorirane. Alternativa besedi *note* sta znaka # ali .
show	Besedi *show* sledi klic funkcije. Po testu se v vrstico doda nova celica, ki vsebuje vrednost, ki jo vrne funkcija.
start	Vrstica vsebuje ime konstruktorja ter njegove argumente za nov objekt, ki nadomesti trenutnega.

Tabela 5.4: Ključne besede tabele `Script`.

Primer tabele `Script` in testnega okolja[4]:

```
!|script |login dialog driver|Bob |xyzzy        |
|login;  |Bob                |xyzzy             |
|check   |login message      |Bob logged in.    |
|reject  |login              |Bob |bad password |
|check   |login message      |Bob not logged in.|
|ensure  |login              |Bob |xyzzy        |
|note    |this is a comment                     |
|show    |number of login attempts              |
|$symbol=|login message                         |
```

```
public class LoginDialogDriver {
  private String userName;
  private String password;
  private String message;
  private int loginAttempts;

  public LoginDialogDriver(String userName, String password) {
    this.userName = userName;
    this.password = password;
  }

  public boolean login(String userName, String password) {
    loginAttempts++;
    boolean result = this.userName.equals(userName) &&
                     this.password.equals(password);
    if (result)
      message = String.format("%s logged in.", this.userName);
    else
      message = String.format("%s not logged in.", this.userName);
    return result;
  }

  public String loginMessage() {
    return message;
  }

  public int numberOfLoginAttempts() {
    return loginAttempts;
  }
}
```

Iz zgornjega primera je razvidno, da gre za tabelo `Script`, kar pove beseda *script* v prvi vrstici. Sledi konstruktor `LoginDialogDriver` z argumentoma

Bob in *xyzzy*. Nato sledi klic funkcije `login`. Sledi klic funkcije `loginMessage` z predpono *check*. Želimo, da funkcija vrne niz *Bob logged in.*. Nato test preveri `login` funkcijo z napačnimi argumenti, in ker smo uporabili predpono *reject* pričakujemo, da se vrstica obarva zeleno, če funkcija vrne false. Sledita še dva testa funkcij `loginMessage` in `login`, komentar in klic funkcije `numberOfLoginAttempts` z predpono *show*. *Show* zgradi novo celico in vanj vstavi vrednost funkcije `numberOfLoginAttempts`. V tem primeru vrednost 3.

5.7.4 Tabela `Scenario`

Tabele `Scenario` so tabele, ki se jih kliče iz drugih tabel, npr. tabele `Script` ali tabele `Decision`. Format tabele `Scenario` je enak formatu tabele `Script`. Njihov namen je eliminirati podvajanje v testih.

5.7.5 Tabela `Import`

Tabela `Import` ni testna tabela. Sistemu pove, katero predpono poti do razredov naj vzame, ko išče razrede testnega okolja. Tako lahko celotno ime testnega okolja (vključno z imenom paketa) zamenjamo samo z imenom razreda testnega okolja.

Primer:

```
|import|
|com.path.to.fixture|

|ClassNameOfFixture|
```

5.7.6 Tabela `Comment`

Podobno kot tabela `Import`, tudi tabela `Comment` ni testna tabela. Tabele `Comment` se uporabljajo samo kot komentar in se med testiranjem ne izvršijo.

Primer:

```
|comment|
|this table| is not|executed|
```

5.7.7 Tabela `Library`

Z tabelo `Library` povemo FitNessu, naj pogleda za vsemi funkcijami na vseh straneh, ki so pod stranjo, kjer je definirana tabela `Library`. Kadarkoli se kliče metoda, ki je ni v testnem okolju, so poklicana vsa testna okolja, ki so definirana kot knjižnica, da bi se našla želena metoda.

Tabela `Library` je ustvarjena kot vsaka druga, pri kateri prva vrstica vsebuje rezervirano besedo *Library*. Vse naslednje vrstice predstavljajo imena testnih okolij.

Primer tabele `Library`:

```
|Library|
|song creation|

|script|jukebox fixture|
|create song|Stairway to Heaven|
|play song|Stairway to Heaven|
```

Primer testnega okolja:

```
public class SongCreation {
    public void createSong (String songName) {
       // song creation logic here
    }
}

public class JukeboxFixture {
   public void playSong (String songName) {
      playService.playSong (songName);
   }
}
```

Strategija iskanja funkcij:

1. Sistem išče funkcijo v trenutnem razredu testnega okolja in ko jo najde jo izvrši.
2. Če funkcije ne najde jo začne iskati v SUT.

3. Če funkcije ne najde v SUT, pogleda spisek definiranih knjižnic. Spisek se pregleda od spodaj navzgor, kar pomeni, da se najprej vzame knjižnica, ki je bila definirana kot zadnja.

5.8 Oblikovanje Wiki strani

V uvodu tega poglavja sem omenil, da je FitNesse *Wiki spletni strežnik*. Wiki omogoča spreminjanje ali kreiranje novih Wiki strani vsem pooblaščenim uporabnikom. Če imamo kopijo FitNessa lokalno na svojem računalniku, potem imamo vsa dovoljenja za spreminjanje teh strani. Te spremembe na Wiki straneh delamo enostavno, samo z uporabo spletnega brskalnika.V podpoglavju 5.4 smo uspešno napisali dve Wiki strani, zbirko testov `TemperatureSuite` ter test `TemperatureTest`. Poleg strani *zbirka testov* in *test*, lahko kreiramo tudi navadno Wiki stran. FitNesse ponuja širok nabor možnosti pri oblikovanju besedila na Wiki straneh. Vse možnosti oblikovanja besedila so prikazane v tabeli 5.5.

Comment	#text
Italics	”text”
Bold	’ ”text”’
Strike-through	–text–
Header	!1 Title !2 Header !3 Small header
Bullet Lists	[space]* Item one [space][space]* Sub item one [space][space]* Sub item two
Numbered lists	[space]1 Item one [space]1 Item two [space]1 Item three
Centering	!c center this text
”As-is”/escaping	!-text-!
”As-is”	!<text>!
Formatted ’as is”	{{{text}}}

Tabela 5.5: Oblikovanje besedila.

Nekaj primerov:
Če želimo **krepko odebeljeno besedilo**, postavimo po tri enojne narekovaje na začetek in konec besedila.
Če želimo *poševno besedilo*, postavimo po dva narekovaja na začetek in konec besedila.

Če želimo centrirati besedilo, pred njim postavimo znak !.

Vrstično in bločno oblikovanje

Horizontal Line	—-
Note	!note text
Collapsible section expanded	!*[title] multi-line Wiki text *!
Collapsible section collapsed	!*>[title] multi-line Wiki text *!
Hidden collapsible section	!*<[title] multi-line Wiki text *!
Plain text table	![my simple table]!
Literalized table	![: first:Erik last:Pragt]!

Tabela 5.6: Vrstično in bločno oblikovanje.

Z simboli iz tabele 5.6 lahko začasno skrijemo ali zložimo del besedila. Prav tako lahko sestavimo preproste tabele.

Z rezerviranimi besedami iz tabele 5.7 se lahko po želji premikamo po straneh, katerokoli besedo spremenimo v povezavo (!anchor), vstavimo sliko (!img), dobimo spisek podstrani (!contects), ali celo na trenutno stran vstavimo vsebino iz druge strani (!include).

Povezave in sklicevanja

Page links - from root	.RootPage[.Childpage]
Page links - sibling	SameLevelPage[.ChildPage]
Page links - child or symbolic	>ChildPage[.ChildPage]
Page links - from parent	<ParentPage[.ChildPage]!
Cross-reference	!see AnyPagePath
”In page”label	!anchor label-name
Jump to anchor -in-line Jump to anchor -left-justified Jump to anchor - in an alias	text #label-name text #label-name [[text][#label-name]]
External links -web -local -alias -alias	http://url-path http://files/somePath [[text][/files”’/localPath]] [[text][AnyPagePath#label-name]]
Picture -clickable	!img url-to-image-file [[!img url-to-image-file][some-link]]
Contents List	!contents
Contents Tree	!contents -R
Content Sub-tree	!contents -R2
Include page	!include AnyPagePath

Tabela 5.7: Povezave in sklicevanja.

Uporaba spremenljivk

Variable Definition	!define name {value}
Variable Usage	$name
Expression Evaluation	$=expression=

Tabela 5.8: Uporaba spremenljivk.

Na primer, da imamo tabelo `Decision`:

```
|DT:odlocitvena tabela|
|vhod     |izhod?     |
|3        |$V=        |
|$V       |8          |
|9        |$V         |
```

Prva vrstica tabele naloži vrnjeno vrednost izhoda v spremenljivko V. V naslednji vrstici se vzame shranjena vrednost spremenljivke V in se jo uporabi za vhod. V tretji vrstici pričakujemo, da bo izhod funkcije vrednost spremenljivke V.

Globalne spremenljivke

Primer definiranja globalne spremenljivke:

```
!define TEST_SYSTEM { slim }
```

Poglavje 6

Primer uporabe

6.1 Opis problema

Trener kluba borilnih veščin želi imeti aplikacijo, s katero bo lahko imel pregled nad podatki o svojih članih, lahko bo dodal ali odstranil člana iz kluba. Prav tako bo lahko iskal člane po imenu in priimku ter prijavil člana na izpit. Pogoj, da se člana prijavi na izpit je, da je član plačal članarino, ima 80% udeležbo na seminarjih, ter da je sporočil prihod preko elektronske pošte ali telefona.

6.2 Uporabniške zgodbe

Uporabnik bi spisal naslednje zgodbe:

```
#1 Trener lahko v klub vpiše novega člana.
#2 Trener lahko izpiše člana iz kluba.
#3 Trener lahko išče člane po imenu, priimku.
#4 Trener lahko izpiše podatke o članih na zaslon.
#5 Trener lahko prijavi člana na izpit.
```

Zaradi lažjega prikaza primera bodo vse zgodbe izdelane v prvem teku. Lastnik izdelka in razvojna skupina se na sestanku planiranja teka pogovorijo o zgodbah in pridejo do naslednjih podrobnosti:

```
    Zabeležka
#1 Podatki o članu bodo:  ime, priimek, naslov, poštna številka in kraj,
šola/fakulteta, email, telefon, teža, stopnja(začetnik, modra, zelena,
...), datum rojstva.  Potrebujemo razred član, pomožni razred datum in
statični razred klub, katerega naloga bo dodajanje, brisanje in urejanje
članov.  Vsa polja so obvezna, razen polje stopnja.
```

6.2.1 Testiranje veljavnosti datuma

Čeprav FitNesse vsebuje že svoj razred `Date`, sem za raziskovalne namene napisal svojega. Pri datumu je potrebno preveriti, da so dan, mesec in leto v pravilnih intervalih ter, da ima prestopno leto v mesecu februarju, 29 dni. Nastanejo naslednji sprejemni testi:

```
Preveri s pravilnim datumom, 3.8.1986.
Preveri s pravilnim datumom, 1.1.2000.
Preveri z napačnim datumom, 0.1.1999.
Preveri z napačnim datumom, 1.1.0.
Preveri z napačnim datumom, 29.2.1900.
Preveri s pravilnim datumom, 29.2.2000.
```

Na podlagi teh sprejemnih testov se spiše naslednja tabela `Script`.

```
|script|date fixture;                                              |
|ensure|check date;         |3           |8         |1986         |
|check |get date            |3. Avgust 1986                       |
|ensure|check date;         |1           |1         |2000         |
|reject|check date;         |0           |1         |1999         |
|reject|check date;         |1           |1         |0            |
|reject|check date;         |29          |2         |1900         |
|ensure|check date;         |29          |2         |2000         |
|note  |leto 1900 ni prestopno, zato februar nima 29 dni, 2000 je|
|show  |is leap year        |1900                                 |
|show  |is leap year        |2000                                 |
|show  |get date                                                  |
```

In seveda testno okolje `DateFixture`.

```
package myclasscollection.fixtures;
```

```
import myclasscollection.sut.Date;

public class DateFixture {
    private Date date;

    public DateFixture() {
        date = new Date();
    }

    public boolean checkDate(int day, int month, int year) {
        return date.setDate(day, month, year);
    }

    public String getDate() {
        return date.getDate();
    }

    public boolean isLeapYear(int year) {
        return date.isLeapYear(year);
    }
}
```

Če je naš razred Date skodiran pravilno in poženemo FitNesse test, dobimo rezultate, prikazane na sliki 6.1

DateTest

TEST RESULTS [history]

Assertions: 7 right, 0 wrong, 0 ignored, 0 exceptions (0.023 seconds)

▶ *Precompiled Libraries*

Contents:

variable defined: TEST_SYSTEM=slim

classpath: C:\Development\MyClassCollection\dist\MyClassCollection.jar

import
myclasscollection.fixtures

script	date fixture;				
ensure	check date;	3	8	1986	
check	get date	3. Avgust 1986			
ensure	check date;	1	1	2000	
reject	check date;	0	1	1999	
reject	check date;	1	1	0	
reject	check date;	29	2	1900	
ensure	check date;	29	2	2000	
note	leto 1900 ni prestopno, zato februar nima 29 dni, 2000 je				
show	is leap year	1900			false
show	is leap year	2000			true
show	get date				29. Februar 2000

Slika 6.1: Rezultati testa Date.

Celoten razred `Date` v dodatku A.

6.2.2 Testiranje veljavnosti podatkov o članu

Prvo kar je potrebno narediti, če želimo nekega člana dodati, je mehanizem za preverjanje veljavnosti podatkov o članu. Za preverjanje veljavnosti podatkov napišemo nekaj sprejemnih testov. Ti nam povedo, kaj je naš cilj, kdaj lahko končamo z kodiranjem, saj smo dobili želeno funkcionalnost. V tem primeru, želimo imeti funkcijo, ki bo preverila veljavnost vhodnih podatkov, zato spišemo naslednje sprejemne teste:

```
Preveri za primer, ko so vsi podatki pravilni.
Preveri s praznim poljem stopnja.
Preveri z napačno poštno številko.
Preveri z napačnim elektronskim naslovom.
Preveri z napačnim naslovom.
Preveri z napačnim imenom ali priimkom.
Preveri z napačno težo.
Preveri z napačno telefonsko številko.
Preveri z napačnim datumom rojstva.
```

Sprejemne teste preslikamo v FitNesse testno tabelo. Odločil sem se za tabelo `Decision`. Z njo sem želel, da se za vsako vrstico v tabeli pokliče funkcija `checkMember`, katera preveri podatke o članu, tabela `Decision` pa nato s pomočjo funkcije `status` izpiše, pri katerih podatkih je prišlo do napake. Tabela bi izgledala takole (zaradi prevelike tabele sem odstranil nekaj podatkov):

```
!|member fixture                                                    |
|name|surname|address|...|...|...|...|...|...|...|status?          |
|Robi|Roker  |...    |   |   |   |   |   |   |   |                 |
|Luka|Ficek  |49     |   |   |   |   |   |   |   |Neveljaven naslov.|
```

Pri članu Robi Roker pričakujem, da bodo vsi podatki veljavni, zato sem pustil polje celice prazno. Pri članu Luka Ficek pa pričakujem, da je naslov neveljaven. Isti rezultat pričakujem od sistema, da ugotovi, da je zgradba naslova napačna in vrne ustrezno opozorilo.

Koda testnega okolja :

```
package myclasscollection.fixtures;

import myclasscollection.sut.Member;
```

```
public class MemberFixture {
    private String name;
    private String surname;
    private String address;
    private String postal;
    private String faculty;
    private String email;
    private String phone;
    private int weight;
    private String level;
    private String date;

    private String status;

    public void setName(String name) {
        this.name = name;
    }
    public void setSurname(String surname) {
        this.surname = surname;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    public void setPostal(String postal) {
        this.postal = postal;
    }
    public void setFaculty(String faculty) {
        this.faculty = faculty;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
    public void setWeight(int weight) {
        this.weight = weight;
    }
    public void setLevel(String level) {
        this.level = level;
    }
    public void setDate(String date) {
        this.date = date;
    }

    public String status() {
```

```
        return this.status;
    }

    public void execute() {
        Member m = new Member();
        m.checkMember(name, surname, address, postal, faculty,
email, phone, weight, level, date);
        status = m.errors();
    }
}
```

Rezultati testa, so prikazani na sliki 6.2.

member fixture

name	surname	address	postal	faculty	email	phone	weight	level	date	status?
Žiga	Elsner	Sostrska cesta 49	1000 Ljubljana	Fakulteta za računalništvo in informatiko	ziga.elsner@hotmail.com	040 476 341	65	rdeča	3.8.1986	BLANK
Robi	Roker	Viška cesta 11	4000 Kranj	FRI	robi.roker@hotmail.com	040 111 241	70		1.1.1980	BLANK
Rok	Roker	Viška cesta 11	4000 Kranj	FRI	rok.roker@hotmail.com	041 121 241	71		1.1.1980	BLANK
Miha	Mohar	Wertska cesta 300	1000 Ljubljana	Srednja frizerska	miha.mohar@gmaill.com	040 116 999	90		1.1.1995	Neveljaven poštni naslov.
Franci	Brodar	Deška 7	999 Ljubljana	Srednja frizerska	franci.brodar@gmail.com	040 111 222	70	modra	2.2.1990	Neveljavna poštna številka in kraj.
Janez	Novak	Ižanska 6	1000 Ljubljana	Srednja frizerska	janez.novakgmail.com	041 123 123	120		1.10.1986	Neveljaven poštni naslov.
Luka	Ficek	49	1000 Ljubljana	Srednja gradbena	luka.ficek@hotmail.com	040 476 341	65	rdeča	11.8.1999	Neveljaven naslov.
andraž	Grof666	Savska cesta 49	1000 Ljubljana	Filozofska Fakulteta	andražgrof@hotmail.com	040 476 341	90		5.6.1970	Neveljaven priimek.
Peter	Pan	Nije 0	1000 Ljubljana	Filozofska Fakulteta	peter.pan@hotmail.com	040 476 222	-40		5.5.1987	Neveljavna teža.
Zvone	Zvon	Nista 3	1000 Ljubljana	Fakulteta za družbene vede	zvone.zvon@hotmail.com	040 476 2222	40		1.3.19887	Neveljaven telefon.Neveljaven datum.

Slika 6.2: Rezultati testa za validacijo podatkov o članih.

Seveda je možnih več načinov, kako testirati naš sistem. Zgornji primer testa verjetno ni najboljši. Morda bi bilo bolje, da bi najprej napisali tako testno tabelo, ki bi nam omogočala hitro iskanje regularnih izrazov za posamezne nize in jasen prikaz, kateri člani imajo neveljavne podatke. Napisal sem tabelo `Scenario`, ki pobira podatke iz tabele `Decision`, nato pa pokliče funkcijo `matches` z dvema vhodnima parametroma: podatkom, ki ga želimo validirati, in regularnim izrazom, s katerim povemo, kakšna naj bo struktura našega podatka. Slika 6.3 prikazuje tabelo `Scenario` za validacijo podatkov o članih.

```
|script|member fixture1|

|scenario|member fixture _|name, surname, address, postal, faculty, email, phone, weight, level, date|
|ensure  |matches;        |@name            |[A-ZŠČŽ][a-zščž]{2,12}                               |
|ensure  |matches;        |@surname         |[A-ZŠČŽ][a-zščž]{2,12}                               |
|ensure  |matches;        |@address         |[A-ZŠČŽ][a-zščž ]+[0-9]{1,3}                         |
|ensure  |matches;        |@postal          |[0-9]{4} [A-ZŠČŽ][a-zščž]+                           |
|ensure  |matches;        |@faculty         |[A-ZŠČŽ]+[a-zščž ]+                                  |
|ensure  |matches;        |@email           |!- .+@(hotmail|gmail)[.](com) -!                     |
|ensure  |matches;        |@phone           |(04)[01][ ]+[0-9]{3}[ ]+[0-9]{3}                     |
|ensure  |weight;         |@weight                                                                 |
|ensure  |matches;        |@level           |[a-zščž ]*                                           |
|ensure  |matches;        |@date            |[0-9]{1,2}[.][0-9]{1,2}[.][0-9]{4}                   |

!|member fixture                                                                                                                                   |
|name  |surname|address           |postal        |faculty                                   |email                   |phone       |weight|level|date      |
|Žiga  |Elsner |Sostrska cesta 49|1000 Ljubljana|Fakulteta za računalništvo in informatiko|ziga.elsner@hotmail.com|040 476 341 |65    |rdeča|3.8.1986  |
|Robi  |Roker  |Viška cesta 11   |4000 Kranj    |Fakulteta za računalništvo in informatiko|robi.roker@hotmail.com |040 111 241 |70    |     |1.1.1980  |
|Rok   |Roker  |Viška cesta 11   |4000 Kranj    |Fakulteta za računalništvo in informatiko|rok.roker@hotmail.com  |041 121 241 |71    |     |1.1.1980  |
|Miha  |Mohar  |Wertska cesta 300|1000 Ljubljana|Srednja frizerska                         |miha.mohar@gmaill.com  |040 116 999 |90    |     |1.1.1995  |
|Franci|Brodar |Deška 7          |999 Ljubljana |Srednja frizerska                         |franci.brodar@gmail.com|040 111 222 |70    |modra|2.2.1990  |
|Janez |Novak  |Ižanska 6        |1000 Ljubljana|Srednja frizerska                         |janez.novakgmail.com   |041 123 123 |120   |     |1.10.1986|
|Luka  |Ficek  |49               |1000 Ljubljana|Srednja gradbena                          |luka.ficek@hotmail.com |040 476 341 |65    |rdeča|11.8.1999|
|andraž|Grof666|Savska cesta 49  |1000 Ljubljana|Filozofska fakulteta                      |andražgrof@hotmail.com |040 476 341 |90    |     |5.6.1970  |
|Peter |Pan    |Nije 0           |1000 Ljubljana|Filozofska fakulteta                      |peter.pan@hotmail.com  |040 476 222 |-40   |     |5.5.1987  |
|Zvone |Zvon   |Nista 3          |1000 Ljubljana|Fakulteta za družbene vede                |zvone.zvon@hotmail.com |040 476 2222|40    |     |1.3.19887|
```

Slika 6.3: Tabela `Scenario` za validacijo podatkov o članih.

Funkcija `matches` je definirana v testnem okolju `MemberFixture1`. Koda testnega okolja `MemberFixture1`:

```
package myclasscollection.fixtures;

import myclasscollection.sut.Member;

public class MemberFixture1 {
    Member member;

    public MemberFixture1() {
            member = new Member();
    }

    public boolean matches(String name, String regex) {
        if(name.matches(regex))
           return true;
        return false;
    }

    public boolean weight(int w) {
        if(w <= 0 || w >= 200)
                return false;
        return true;
    }
}
```

Celotna koda razreda `Member` v dodatku B.

6.2.3 Testiranje akcij dodajanje, brisanje, iskanje ter izpis članov

Za testiranje zgodb *Trener lahko v klub vpiše novega člana*, *Trener lahko izpiše člana iz kluba*, *Trener lahko išče člane po imenu, priimku*, *Trener lahko izpiše podatke o članih na zaslon* sem si zamislil naslednje zaporedje akcij:

```
preveri, da je število članov 0
dodaj novega člana
dodaj novega člana
dodaj člana, ki že obstaja
preveri, da je število članov 2
izpiši člane na zaslon
išči člana po imenu in priimku
izbriši člana glede na njegov id
izpiši člane na zaslon
```

Sedaj te akcije preslikam v tabelo `script` na FitNesse wiki testno stran. Tabela je prikazana na sliki 6.4.

```
|!|script|club fixture;                                                                                                    |
|check  |count members;|0                                                                                                 |
|ensure |add member;   |Žiga,Elsner,Sostrska cesta 49,1000 Ljubljana,FRI,ziga.elsner@hotmail.com,040 476 341,65,rdeča,3.8.1986          |
|ensure |add member;   |Miha,Mohar,Wertska cesta 300,1000 Ljubljana,Srednja frizerska,miha.mohar@gmail.com,040 116 999,90,rdeča,1.1.1995|
|reject |add member;   |Žiga,Elsner,Sostrska cesta 49,1000 Ljubljana,FRI,ziga.elsner@hotmail.com,040 476 341,65,rdeča,3.8.1986          |
|check  |count members;|2                                                                                                 |
|show   |show members;                                                                                                    |
|show   |find member;  |Žiga                                                  |Elsner                                                    |
|ensure |remove member;|0                                                                                                 |
|show   |show members;                                                                                                    |
```

Slika 6.4: Tabela zaporedja akcij.

Da bo zadevo možno testirati potrebujemo testno okolje `ClubFixture` ter razred `Club`, ki se nahaja v dodatku C.

```
package myclasscollection.fixtures;

import fitnesse.slim.Slim;
import java.util.ArrayList;
import myclasscollection.sut.Club;
import myclasscollection.sut.Member;

public class ClubFixture {

    public ClubFixture() {
        Club.members = new ArrayList<Member>();
        Slim.addConverter(Member.class, new MemberConverter());
    }
```

```
    public boolean addMember(Member m) {
        return Club.addMember(m);
    }

    public boolean removeMember(int id) {
        return Club.removeMember(id);
    }

    public int countMembers() {
        return Club.countMembers();
    }

    public String showMembers() {
        return Club.showMembers();
    }

    public String findMember(String name, String surname) {
        return Club.findMember(name, surname);
    }
}
```

Kot je razvidno iz slike 6.4, so podatki o članih ločeni z vejico. Vsi podatki v SLIM tabelah so nizi. Čeprav SLIM prihaja že s standardnimi podatkovnimi tipi, lahko naredimo tudi svoje. Kar je potrebno narediti, je svoj pretvornik in ga dodati v prvo testno okolje, ki uporablja ta naš novi podatkovni tip.

Pretvornik, ki pretvori niz v razred `Member` in obratno, izgleda takole:

```
package myclasscollection.fixtures;

import fitnesse.slim.Converter;
import myclasscollection.sut.Member;

public class MemberConverter implements Converter {

    public String toString(Object o) {
        Member m = (Member) o;
        return m.getName()+","+m.getSurname()+","+m.getAddress()+","+
               m.getPostal_code_and_city()+","+m.getSchool_faculty() +","+
               m.getEmail()+","+m.getPhone()+","+m.getWeight() +","+
               m.getLevel()+","+m.getDate_of_birth();
    }

    public Object fromString(String string) {
        String[] n = string.split(",");
        return new Member(n[0],n[1],n[2],n[3],n[4],n[5],n[6],
```

```
     Integer.parseInt(n[7]),n[8],n[9]);
    }

}
```

6.2.4 Testiranje poizvedb

V tekstovni datoteki *members* hranimo podatke o članih. Trener v svoji aplikaciji želi funkcionalnost iskanja članov po imenu in priimku. Testirati želimo, da bo naša še ne realizirana funkcija `findMemberByNameAndSurname` delovala pravilno. Argumenta funkcije bosta ime in priimek. Odločim se za test v obliki tabele `Query`. Sestavim naslednji test, prikazan na sliki 6.5.

```
|Query: find member by name and surname|Peter  |Pan                                                                  |
|name                                  |surname|address|postal|school|email|phone|weight|level|date_of_birth|
|Peter                                 |Pan    |       |      |      |     |     |      |     |              |
```

Slika 6.5: Tabela Query iskanje člana po imenu in priimku.

V tem testu želim, da testno okolje v podatkovni bazi najde člane z imenom *Peter* in priimkom *Pan* in jih s pomočjo funkcije `query` vrne testni strani. Podatki so vrnjeni kot seznam vrstic. Vrstice v tabeli `Query` predstavljajo pričakovane rezultate poizvedbe. Test, kot pričakovano uspe, kar je razvidno iz slike 6.6.

Query: find member by name and surname	Peter	Pan							
name	surname	address	postal	school	email	phone	weight	level	date_of_birth
Peter	Pan	Nije 0	1000 Ljubljana	Filozofska fakulteta	peter.pan@hotmail.com	040 476 222	40		5.5.1987

Slika 6.6: Rezultat testa iskanje člana po imenu in priimku.

Testno okolje za zgornji primer bi izgledalo takole:

```
package myclasscollection.fixtures;

import myclasscollection.sut.Queries;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import myclasscollection.sut.Member;

public class FindMemberByNameAndSurname {
```

```
    ArrayList<Member> members = new ArrayList<Member>();
    private String name;
    private String surname;

    public FindMemberByNameAndSurname(String name, String surname)
 throws FileNotFoundException {
        this.name = name;
        this.surname = surname;
        Queries.loadMembers();
    }

    public List<Object> query() {
      return Queries.findMemberByNameAndSurname(name, surname);
    }
}
```

Pri tabeli `Query` lahko pustimo prazne vrstice, kar pomeni, da ne pričakujemo nobenih specifičnih rezultatov poizvedbe. Če rezultati so, se vrstice kljub temu zapolnijo. Na primer, da želimo izpisati vse člane iz tekstovne datoteke *members*. Potrebujemo testno okolje `ListOfMembers` in tabelo `Query`, ki je zelo podobna prejšni tabeli, le da je brez vhodnih argumentov ter pričakovanih vrednosti.

```
package myclasscollection.fixtures;

import java.io.FileNotFoundException;
import java.util.List;
import myclasscollection.sut.Queries;

public class ListOfMembers {

    public ListOfMembers() throws FileNotFoundException {
       Queries.loadMembers();
    }

    public List<Object> query() {
      return Queries.listOfMembers();
    }
}
```

Če poženemo test, dobimo rezultat, prikazan na sliki 6.7. Na sliki opazimo, da so se polja obarvala rdeče z rezervirano besedo surplus, kar pomeni, da jih nismo pričakovali, pa vendar so v naši datoteki.

Query:list of members

name	surname	address	postal	school	email	phone	weight	level	date_of_birth
[Žiga] surplus	Elsner	Sostrska cesta 49	1000 Ljubljana	Fakulteta	ziga.elsner@hotmail.com	040 476 341	65	rdeča	3.8.1986
[Robi] surplus	Roker	Viška cesta 11	4000 Kranj	Fakulteta za računalništvo in informatiko	robi.roker@hotmail.com	040 111 241	70		1.1.1980
[Rok] surplus	Roker	Viška cesta 11	4000 Kranj	Fakulteta za računalništvo in informatiko	rok.roker@hotmail.com	041 121 241	71		1.1.1980
[Miha] surplus	Mohar	Vertska cesta 300	1000 Ljubljana	Srednja frizerska	miha.mohar@gmail.com	040 116 999	90		1.1.1995
[Franci] surplus	Brodar	Deška 7	1000 Ljubljana	Srednja frizerska	franci.brodar@gmail.com	040 111 222	70	modra	2.2.1990
[Janez] surplus	Novak	Ižanska 6	1000 Ljubljana	Srednja frizerska	janez.novak@gmail.com	041 123 123	120		1.10.1986
[Luka] surplus	Ficek	Stritarska 49	1000 Ljubljana	Srednja gradbena	luka.ficek@hotmail.com	040 476 341	65	rdeča	11.8.1999
[Andraž] surplus	Grof	Savska cesta 49	1000 Ljubljana	Filozofska fakulteta	andrazgrof@hotmail.com	040 476 341	90		5.6.1970
[Peter] surplus	Pan	Nije 0	1000 Ljubljana	Filozofska fakulteta	peter.pan@hotmail.com	040 476 222	40		5.5.1987
[Zvone] surplus	Zvon	Nista 3	1000 Ljubljana	Fakulteta za družbene vede	zvone.zvon@hotmail.com	040 476 282	40		1.3.1988
[Zvone] surplus	Huj	Nista 7	1000 Ljubljana	Fakulteta za družbene vede	zvone.huj@hotmail.com	040 111 382	40		1.8.1995
[Micka] surplus	Zvon	Nista 3	1000 Ljubljana	Filozofska fakulteta	micka.zvon@hotmail.com	040 777 282	40		6.3.1989
[Maja] surplus	Brodar	Deška 55	1000 Ljubljana	Srednja frizerska	maja.brodar@gmail.com	040 117 777	70	modra	2.11.1990
[Janez] surplus	Dero	Krakova cesta 6	1000 Ljubljana	Srednja frizerska	janez.dero@gmail.com	041 333 123	100		10.10.1986
[Petra] surplus	Ficek	Rečna 49	1000 Ljubljana	Srednja gradbena	petra.ficek@hotmail.com	040 476 341	65	rdeča	8.8.1998

Slika 6.7: Rezultat testa spisek članov.

Za konec še končni sistem, katerega funkcije sem testiral:

```
package myclasscollection.sut;

import java.io.FileNotFoundException;
import java.io.FileInputStream;
import java.util.Scanner;
import java.io.File;
import java.util.ArrayList;
import static util.ListUtility.list;

public class Queries {

    public static void loadMembers() throws FileNotFoundException {
        Club.members = new ArrayList<Member>();

        File file =
                new File("C:/Development/MyClassCollection/members.txt");
        Scanner sc =
                new Scanner(new FileInputStream(file), "UTF-8");

        try {
            while(sc.hasNextLine()){
                String[] m = sc.nextLine().split(",");
                Club.addMember(new Member(m[0], m[1], m[2], m[3],
                                          m[4], m[5], m[6],
                        Integer.parseInt(m[7]), m[8], m[9]));
            }
        } finally {
            sc.close();
```

```
        }
    }

    public static ArrayList<Object>
findMemberByNameAndSurname(String name, String surname) {
      ArrayList<Object> objects = new ArrayList<Object>();

      for(Member m:Club.members) {
        if(m.getName().compareTo(name) == 0 &&
m.getSurname().compareTo(surname) == 0){
        objects.add(list(
                    list("name", m.getName()),
                    list("surname", m.getSurname()),
                    list("address", m.getAddress()),
                    list("postal", m.getPostal_code_and_city()),
                    list("school", m.getSchool_faculty()),
                    list("email", m.getEmail()),
                    list("phone", m.getPhone()),
                    list("weight", m.getWeight()),
                    list("level", m.getLevel()),
                    list("date_of_birth", m.getDate_of_birth())
                ));
         }
      }
      return objects;
    }

    public static ArrayList<Object> listOfMembers() {
      ArrayList<Object> objects = new ArrayList<Object>();

      for(Member m:Club.members) {
        objects.add(list(
                    list("name", m.getName()),
                    list("surname", m.getSurname()),
                    list("address", m.getAddress()),
                    list("postal", m.getPostal_code_and_city()),
                    list("school", m.getSchool_faculty()),
                    list("email", m.getEmail()),
                    list("phone", m.getPhone()),
                    list("weight", m.getWeight()),
                    list("level", m.getLevel()),
                    list("date_of_birth", m.getDate_of_birth())
                ));
      }
      return objects;
    }
}
```

6.2.5 Testiranje zgodbe *Trener lahko prijavi člana na izpit*

Če želi trener prijaviti člana na izpit, mora ta izpolnjevati naslednje pogoje: plačana članarina, vsaj 80% udeležba na seminarjih in potrjena udeležba preko elektronske pošte ali telefona.

Iz zgornjih informacij lahko sestavimo naslednjo odločitveno tabelo:

```
|exam conditions                  |
|membership|seminars|confirmed|exam|
|yes       |80      |yes      |yes |
|yes       |80      |no       |no  |
|yes       |50      |yes      |no  |
|yes       |50      |no       |no  |
|no        |80      |yes      |no  |
|no        |80      |no       |no  |
|no        |50      |yes      |no  |
|no        |50      |no       |no  |
```

In s pomočjo tega testa uspešno sestavimo naslednji pogoj:

```
exam = ((membership.compareTo("yes") == 0) &&
               (confirmed.compareTo("yes") == 0)) &&
               (seminars >= 80) ? "yes" : "no";
```

Glede na zgornjo tabelo, napišemo naslednje sprejemne teste:

```
Preveri za primer, ko član izpolnjuje vse pogoje.
Preveri za primer, ko član ne izpolnjuje enega od teh pogojev.
```

Testa opisujeta dva možna scenarija. Ko član izpolnjuje vse pogoje in ko jih ne. Eden od možnih testov, prikazan na sliki 6.8.

scenario	member _	name, surname, addr, post, fax, email, phone, weight, lvl, date, membership, seminars, confirmed			
ensure	add member to exam;	@name,@surname,@addr,@post,@fax,@email,@phone,@weight,@lvl,@date	@membership	@seminars	@confirmed

scenario	invalid member _	name, surname, addr, post, fax, email, phone, weight, lvl, date, membership, seminars, confirmed			
reject	add member to exam;	@name,@surname,@addr,@post,@fax,@email,@phone,@weight,@lvl,@date	@membership	@seminars	@confirmed

member												
name	surname	addr	post	fax	email	phone	weight	lvl	date	membership	seminars	confirmed
Žiga	Elsner	Sostrska 49	1000 Ljubljana	Fri	ziga.elsner@hotmail.com	040 476 341	65	rdeča	3.8.1986	yes	80	yes

invalid member												
name	surname	addr	post	fax	email	phone	weight	lvl	date	membership	seminars	confirmed
Žiga	Elsner	Sostrska 49	1000 Ljubljana	Fri	ziga.elsner@hotmail.com	040 476 341	65	rdeča	3.8.1986	yes	80	no

Slika 6.8: Tabele testa prijava člana na izpit.

Če poženemo test, pričakovano ne uspe. Nato v naš sistem, ki je pod testom dodamo funkcijo `addMemberToExam`:

```
public static boolean addMemberToExam(Member m, String membership,
 int seminars, String confirmed) {
        if ((membership.compareTo("yes") == 0) &&
                (confirmed.compareTo("yes") == 0) &&
                (seminars >= 80)){
                    return exams.add(m);
        }
                return false;
    }
```

Če test ponovno poženemo, se izvede uspešno.

Poglavje 7

Zaključek

Že desetletje so v uporabi agilne metodologije razvoja programske opreme. Razvite so bile z namenom izboljšati razvojni proces ter na koncu dobiti, kar se da kvaliteten končni produkt. Kot kaže, se je v zadnjih letih njihova uporaba znatno povečala. Najbolj uporabljena je trenutno metodologija Scrum. Kot vemo, da nihče ni popoln, isto velja za metodologije razvoja programske opreme. Vsaka od njih ima dobre in slabe latnosti. Metodologija Scrum ni izjema. Največja prednost metodologije Scrum je tudi njena največja slabost. In sicer proces, ki se sproti spreminja in dopolnjuje, povzroča težave pri planiranju. Uporabniku izdelka je nerazumljivo, kako mu ne moremo podati neke točne ocene o projektu ter njegovi dolžini razvoja. Ne zaveda se, da so ravno njegove zahteve tiste, ki to onemogočajo.

Z vpeljavo uporabniških zgodb v metodologijo Scrum vključimo uporabnika v sam razvoj. S tem povečamo razumljivost samega problema, poleg tega pa uporabnik sam vidi napredek razvoja ter kako lahko on vpliva na sam proces. Za metodologijo Scrum, posebej če jo uporabljamo z uporabniškimi zgodbami, je značilno pisanje sprejemnih testov pred kodiranjem, in ne na koncu razvoja, kot je to v navadi pri klasičnem razvoju. Da bi bil razvojni proces še lažji, obstaja cela vrsta orodij, med njimi tudi FitNesse, ki sem ga bolj podrobno opisal že v jedru diplomske naloge.

V treh mesecih uporabe sem spoznal, da je FitNesse zelo učinkovito orodje za pisanje sprejemnih testov in sem z njim rešil večino problemov, ki jih je bilo potrebno identificirati in rešiti. Kot programerju so mi bili sprejemni testi kot neko navodilo, kaj še moram narediti in kaj je moj cilj. Orodje je enostavno za uporabo, začetno uvajanje je kljub vsemu potrebno. Kljub temu je orodje FitNesse in proces razvoja z metodologijo Scrum hitro priučljiv. Sam vidim v agilnih metodologijah ter njihovih tehnikah in orodjih še svetlo prihodnost.

Dodatek A

Razred Date

```
package myclasscollection.sut;

import java.util.Calendar;

public final class Date {

    private int[] daysPerMonth = {31,28,31,30,31,30,31,31,30,31,30,31};
    private String[] months = {
        "Januar","Februar","Marec",
        "April","Maj","Junij",
        "Julij","Avgust","September",
        "Oktober","November","December"
    };

    protected int day;
    protected int month;
    protected int year;

    public Date(int day, int month, int year) {
        try {
            setYear(year);
            setMonth(month);
            setDay(day);
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

    public Date(){}

    public int getDay() { return day; }
    public int getMonth() { return month; }
```

```
public int getYear() { return year; }

private void setDay(int day) throws Exception {
    int daysInFebruar;
    if(isLeapYear(year))
        daysInFebruar = 29;
    else
        daysInFebruar = 28;
    if(day < 1 || day > daysInFebruar)
        throw new Exception("Dan mora biti med 1 in "+
                daysPerMonth[month - 1]+".");
    this.day = day;
}

private void setMonth(int month) throws Exception {
    if(month < 1 || month > 12)
        throw new Exception("Mesec mora biti med 1 in 12.");
    this.month = month;
}

private void setYear(int year) throws Exception {
    Calendar calendar = Calendar.getInstance();
    if(year < 1900 || year > calendar.get(Calendar.YEAR))
        throw new Exception("Leto mora biti med 1900 in "+
                calendar.get(Calendar.YEAR)+".");
    this.year = year;
}

public boolean setDate(int day, int month, int year) {
     try {
        setYear(year);
        setMonth(month);
        setDay(day);
        return true;
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
        return false;
    }
}

public boolean isLeapYear(int year) {
    boolean leapYear;
    if(year % 400 == 0) leapYear = true;
    else if(year % 100 == 0) leapYear = false;
    else if(year % 4 == 0) leapYear = true;
    else leapYear = false;
    return leapYear;
```

```
    }

    public String getDate() {
        return "" + day + ". " + months[month - 1] + " " + year + "";
    }
}
```

Dodatek B

Razred Member

```
package myclasscollection.sut;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.Scanner;

public final class Member {
    private int memberID;
    private String name;
    private String surname;
    private String address;
    private String postal_code_and_city;
    private String school_faculty;
    private String email;
    private String phone;
    private int weight;
    private String level;
    private String date_of_birth;

    private String status;

    private String[] levels = {"začetnik","modra","zelena","rdeča",
           "bela","rumena","siva"};
```

```
public Member(String name, String surname, String address,
                    String postal_code_and_city,
                    String school_faculty, String email,
                    String phone, int weight,
                    String level, String date_of_birth) {
    try {
        setName(name);
        setSurname(surname);
        setAddress(address);
        setPostal_code_and_city(postal_code_and_city);
        setSchool_faculty(school_faculty);
        setEmail(email);
        setPhone(phone);
        setWeight(weight);
        setLevel(level);
        setDate_of_birth(date_of_birth);
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
    }
}

public Member(){}

public int getMemberID() {
    return memberID;
}
public String getName() {
    return name;
}
public String getSurname() {
    return surname;
}
public String getAddress() {
    return address;
}
public String getDate_of_birth() {
    return date_of_birth;
}
```

```
public String getEmail() {
    return email;
}
public String getLevel() {
    return level;
}
public String getPhone() {
    return phone;
}
public String getPostal_code_and_city() {
    return postal_code_and_city;
}
public String getSchool_faculty() {
    return school_faculty;
}
public int getWeight() {
    return weight;
}

public void setName(String name) throws Exception {
    if(!name.matches("[a-zA-ZščžŠČŽ]+") ||
name.length() <= 2)
        throw new Exception("Neveljavno ime.");
    this.name = name;
}

public void setSurname(String surname) throws Exception {
    if(!surname.matches("[a-zA-ZščžŠČŽ]+") ||
surname.length() <= 2)
        throw new Exception("Neveljaven priimek.");
    this.surname = surname;
}

public void setAddress(String address) throws Exception {
    String[] addr = address.split(" ");
    if(addr.length < 2 || addr[addr.length-1].matches("\\D"))
        throw new Exception("Neveljaven naslov.");
    this.address = address;
}

public void setDate_of_birth(String date_of_birth)
                  throws Exception {
    if(!date_of_birth.
                  matches("[0-9]{1,2}[.][0-9]{1,2}[.][0-9]{4}"))
        throw new Exception("Neveljaven datum.");
    this.date_of_birth = date_of_birth;
}
```

```
public void setEmail(String email) throws Exception {
    if(!email.matches(".+@(hotmail|gmail)\\.[a-z]+"))
        throw new Exception("Neveljaven poštni naslov.");
    this.email = email;
}

public void setLevel(String level) throws Exception {
    if(level.isEmpty()){ this.level = "začetnik"; }
    if(!level.isEmpty() && !isLevelValid(level))
        throw new Exception("Neveljavna stopnja.");
    this.level = level;
}

public void setMemberID(int memberID) {
    this.memberID = memberID;
}

public void setPhone(String phone) throws Exception {
    if(!phone.matches("04[01]\\s+[0-9]{3}\\s+[0-9]{3}"))
        throw new Exception("Neveljaven telefon.");
    this.phone = phone;
}

public void setWeight(int weight) throws Exception {
    if(weight < 0 || weight > 200)
        throw new Exception("Neveljavna teža.");
    this.weight = weight;
}

public void setPostal_code_and_city(String postal_code_and_city)
    throws FileNotFoundException, Exception {
    if(!isPostalCodeAndCityValid(postal_code_and_city))
    throw new Exception("Neveljavna poštna številka in kraj.");
    this.postal_code_and_city = postal_code_and_city;
}

public void setSchool_faculty(String school_faculty)
    throws Exception {
    if(school_faculty.matches("[a-z|ščž]*"))
    throw new Exception("Neveljaven naziv fakultete.");
    this.school_faculty = school_faculty;
}

public void checkMember(String name, String surname,
      String address, String postal_code_and_city,
        String school_faculty, String email, String phone,
```

```
    int weight, String level, String date_of_birth) {
    status = "";
    setMemberID(memberID);

try {
    setName(name);
} catch (Exception ex) {
    status += ex.getMessage();
}

try {
    setSurname(surname);
} catch (Exception ex) {
    status += ex.getMessage();
}

try {
    setAddress(address);
} catch (Exception ex) {
    status += ex.getMessage();
}

try {
    setPostal_code_and_city(postal_code_and_city);
} catch (Exception ex) {
    status += ex.getMessage();
}

try {
    setSchool_faculty(school_faculty);
} catch (Exception ex) {
    status += ex.getMessage();
}

try {
    setEmail(email);
} catch (Exception ex) {
    status += ex.getMessage();
}

try {
    setPhone(phone);
} catch (Exception ex) {
    status += ex.getMessage();
}

try {
```

```
            setWeight(weight);
        } catch (Exception ex) {
            status += ex.getMessage();
        }
        try {
            setLevel(level);
        } catch (Exception ex) {
            status += ex.getMessage();
        }
        try {
            setDate_of_birth(date_of_birth);
        } catch (Exception ex) {
            status += ex.getMessage();
        }
    }

    private boolean isLevelValid(String level) {
        boolean validLevel = false;
        for(int i = 0;i < levels.length;i++){
            if(levels[i].compareTo(level) == 0) validLevel = true;
        }
        return validLevel;
    }

private boolean isPostalCodeAndCityValid(String postal_code_and_city)
      throws FileNotFoundException {

   File file = new File("C:/Development/MyClassCollection/ps.txt");

   Scanner sc = new Scanner(new FileInputStream(file),"UTF-8");

   try {
       while(sc.hasNextLine()){
           String[] postalCodeAndLocation = sc.nextLine().split(" ");
           String location = "";
           for(int i = 0;i < postalCodeAndLocation.length-1; i++) {
               location += " " + postalCodeAndLocation[i];
           }
           String niz =
      postalCodeAndLocation[postalCodeAndLocation.length-1]+""+location;
           if(niz.compareTo(postal_code_and_city) == 0) {
               return true;
           }
       }
   } finally {
       sc.close();
   }
```

```
   return false;
}

public String errors() {
    return status;
}

@Override
public String toString() {
    return "(" + this.getMemberID() +","+ this.getName() +","+
           this.getSurname() +","+ this.getAddress() +","+
           this.getPostal_code_and_city() +","+
           this.getSchool_faculty() +","+ this.getEmail() +","+
           this.getPhone() +","+ this.getWeight() +","+
           this.getLevel() +","+ this.getDate_of_birth() + ")";
 }
}
```

Dodatek C

Razred Club

```
package myclasscollection.sut;

import java.util.ArrayList;

public class Club {
    public static int countMembers = 0;

    public String status;

    public static ArrayList<Member> members;

    public static boolean addMember(Member m) {
        m.setMemberID(countMembers);
        countMembers++;
        m.checkMember(m.getName(), m.getSurname(), m.getAddress(),
        m.getPostal_code_and_city(), m.getSchool_faculty(),
        m.getEmail(), m.getPhone(), m.getWeight(), m.getLevel(),
        m.getDate_of_birth());
        if(!"".equals(m.errors()))
            return false;
        if(memberExists(m))
            return false;
        return members.add(m);
    }

    public static boolean memberExists(Member m) {
  for(int i=0; i < members.size(); i++) {
      if(members.get(i).getName().compareTo(m.getName()) == 0 &&
         members.get(i).getSurname().compareTo(m.getSurname()) == 0 &&
         members.get(i).getAddress().compareTo(m.getAddress()) == 0) {
              return true;
      }
```

```
     }
     return false;
    }

    public static int countMembers() {
        return members.size();
    }

    public static String showMembers() {
        String mem = "";
        for(int i=0; i < members.size(); i++) {
            mem += members.get(i).toString() + "\n";
        }
        return mem;
    }

    public static String findMember(String name, String surname) {
        String s = "";
        for(int i=0; i < members.size(); i++) {
            if(members.get(i).getName().compareTo(name) == 0 &&
               members.get(i).getSurname().compareTo(surname) == 0) {
               s += members.get(i).toString() + "\n";
            }
        }
           return s;
    }

    public static boolean removeMember(int id) {
        return members.remove(members.get(id));
    }
}
```

Slike

Tabele

Literatura

[1] (2011) Manifesto for Agile Software Development.
Dostopno na: http://agilemanifesto.org/.

[2] (2010) Scrum. Dostopno na:
http://www.scrum.org/storage/scrumguides/Scrum%20Guide.pdf.

[3] M. Cohn, "User Stories Applied: For Agile Software Development,"
Addison-Wesley Professional, 2004.

[4] (2011) FitNesse. Dostopno na: http://fitnesse.org/.

[5] (2011) Software Testing. Dostopno na:
http://en.wikipedia.org/wiki/Software_testing

[6] (2002) Planning Poker or How to avoid analysis paralysis while release planning. Dostopno na:
http://www.renaissancesoftware.net/files/articles/PlanningPoker-v1.1.pdf.