СУ "Св. Климент Охридски"
Факултет по Математика и Информатика
Катедра "Информационни технологии"

Резюме
на
Дипломна работа

Тема:
"Аспектно-ориентирана система за тестване на софтуер, проектиран на базата на контракти"

Дипломант: Диана Иванова Берберова
Специалност: Информатика - Софтуерни технологии
Факултетен номер: M21571

Научен ръководител: доц. д-р Боян Бончев

София, 2007

СЪДЪРЖАНИЕ

1. Въведение

В последните двадесет години софтуерната индустрия се разраства непрестанно. Тя превзема нови и нови територии и покрива всеки аспект от живота на хората – бизнес, търговия, икономика, комуникации, медицина, наука и забавления.

Тенденцията е софтуерните продукти да стават все по-сложни и все по-критични. С това нараства и нуждата от качествен софтуер във всяка една сфера. Въпреки постоянното икономическо развитие и печалбите, които се реализират благодарение на продуктите на софтуерната индустрия, самите продукти невинаги отговарят на необходимите изисквания за качество. Основният въпрос, който стои пред софтуерната индустрия е как да осигури качеството и благонадеждността на софтуерните приложения.

Софтуерното инженерство е науката, която търси и дава отговор на този въпрос. Софтуерното инженерство предлага методи за проектиране, разработване и документиране на софтуера, чрез използване на различни технологии, които подобряват качеството на приложенията. В софтуерната индустрия се разчита почти изцяло на тестването на продуктите за отстраняване на техните дефекти, затова софтуерните инженери търсят непрестанно нови по-добри технологии за тестване.

1.1 Цел

Целта на дипломната работа е да се проектира и разработи аспектно-ориентирана система за тестване на софтуерни приложения, проектирани на базата на контракти (Design by Contract). Компонентите на системата ще бъдат създадени с аспектно-ориентирано програмиране и Java 5. Системата ще позволява описанието на различни контракти по време на

дизайна и разработката на Java приложения, и ще проверява условията на тези контракти по време на работата на приложенията.

Задачите, които произлизат от целта са:

- Дефиниране на различните условия използвани в контрактите - входни условия, изходни условия и инвариантни условия
- Създаване на Java анотации, с които тези условия да се описват
- Създаване на компонент, който оценява условията описани в анотациите
- Създаване на аспектно-ориентиран компонент, който проверява условията по време на работата на тестваното приложение

Системата се нарича CodeContract и се разработва, като проект с отворен код. Поради тази причина и за да може да бъде по-широко публикувана и използвана дипломната работа е разработена на английски език. Текущият документ представлява резюме на дипломната работа.
Системата както и цялата документация може да бъде свалена от: http://code.google.com/p/codecontract/.

1.2 Структура на дипломната работа

Дипломната работа се състои от следните глави:

- Въведение – кратко описание на целите на проекта
- Описание на проблемната област – методологии и решения
- Използвани технологии и методологии – описание на различните техники използвани при разработка на системата
- Архитектура на системата – дизайн и имплементация
- Тестване – план за тестване, тестови процедури и случаи
- Съществуващи решения – описание и сравнение с конкретната разработка
- Заключение и бъдещо развитие

- Използвана литература
- Приложения

2. Описание на проблемната област

Много често, при разработването на софтуерни продукти основното качество, за което се следи е продуктивността на продукта. В обектно-ориентираното програмиране продуктивността зависи не само от подхода за разработка и директни ползи, които реализира, но и от качеството на продукта, което той осигурява. Така въпросът, който стои пред програмистите използващи обектно–ориентиран подход е:
Как се създава качествен обектно-ориентиран софтуер?
За да се отговори на този въпрос, първо трябва да се дефинира какво означава качествен софтуер.

2.1 Фактори определящи качеството на софтуера

Основен компонент на качеството на софтуера е надеждността. Това е способността на системата да изпълнява своите задачи според спецификацията, по която е създадена и възможността да се справя успешно в неправилни и непредвидени ситуации. С други думи – надеждността е липса на дефекти в системата.

Надеждността, макар и желана във всеки подход за създаване на софтуер е изключително важна в обектно-ориентирания метод, заради специалната роля, която е отредена в него на възможността за повторно използване на софтуерните компоненти. Ако не може да се гарантира коректността на компонентите, които се създават се губи основно предимство на обектно-ориентираното програмиране. Все пак, макар и абсолютно необходима надеждността сама по себе си не е достатъчна за определяне качеството на софтуера.

Има редица външни и вътрешни фактори, които определят качеството. Външните фактори, като скорост, производителност, леснота при употреба, са тези, които могат да бъдат открити от потребителите на дадената система. Вътрешните фактори като четимост, модуларност и възможност за разширяване са видими само за тези, които разработват и поддържат продукта. Външните фактори са тези, които имат значение, но те могат да бъдат постигнати само чрез постигането на вътрешните фактори.

Най-важните външни фактори са:

- Коректност и изпълняване на задачите според спецификацията
- Способност за справяне с грешки и неочаквани състояния
- Възможност за развитие и адаптиране на продукта към промени в спецификацията
- Възможност за използване на компонентите при конструиране на много и различни приложения
- Функционалност и възможности

Има и други фактори като ефективност, леснота при употреба, икономичност и съвместимост, които са от значение за крайния потребител. Много от външните фактори обаче си противоречат. Например, за да се направи една система максимално ефективна, тя трябва да се адаптирана напълно към конкретната хардуерна и софтуерна среда и към спецификацията. Първото изискване ограничава съвместимостта с други среди, а второто възпрепятства възможността за повторно използване на компонентите.

Несъвместимостта на различните изисквания за качество води до правенето на отстъпки и компромиси спрямо някои от тях при разработването на софтуер. Важно е, обаче тези отстъпки да се правят планирано и след оценка на всички положителни и отрицателни последствия. Все пак има един фактор, с който не трябва да се правят

компромиси независимо от цената – коректността. Ако един софтуерен продукт не изпълнява правилно функциите, за които е създаден, останалите му качества нямат значение.

2.2 Съществуващи методологии

Софтуерното инженерство предлага много и различни методологии, които осигуряват разработването на качествен софтуер. В дипломната работа са разгледани някои от основните техники заедно с техните положителни и отрицателни качества:

- Традиционно тестване (Traditional style of testing)
- Компонентно тестване (Unit testing)
- Разработване базирано на тестове (Test Driven Development)
- Защитно програмиране (Defensive programming)
- Проектиране на базата на контракти (Design by contract)

При традиционното тестване след разработването на системата се проверява дали тя отговаря на спецификацията, по която е създадена. Използват се различни методи на тестване - някои включват познания по вътрешната архитектура и имплементация (white box testing), а други са фокусирани единствено върху функционалните изисквания към системата (black box testing). Използването на традиционно тестване води до няколко съществени проблема, основният, от които е, че често се изражда в практиката “първо пиши, после поправяй”, която не позволява ранното откриване на дефектите във фазата на разработка.

Тестването на компоненти, и разработване на софтуер на базата на тестове са две практики заложени в методологията за Екстремно програмиране. Едно от предимствата на тези техники е, че резултатът от тях са множество тестове проверяващи коректността на продукта, които лесно могат да бъдат изпълнени и анализирани. Това улеснява

интеграцията, промените и развитието на продукта, тъй като предлага лесен начин за проверка на коректността след промените. Сред основните недостатъци на тези техники са невъзможността да се тестват всички възможни сценарии за работа и възможни входни данни. Освен това, те не предлагат възможност за откриване на проблеми с интеграцията и ефективността. Не на последно място, програмистите често подценяват значимостта на тестовете резултатът, от което е лошо написан тестов код и грешки.

В основата на защитното програмиране стои тестването на всички възможни, очаквани и неочаквани ситуации, които могат да възникнат по време на работата на приложението. Всички грешки се прихващат и обработват и нищо не се приема за даденост. Предимствата на тази техника са, че предпазва приложението от невалидни входни данни и обработва правилно евентуалните грешки. Недостатъците са намаляване на производителността и необходимостта от много допълнителен “защитен” код, който усложнява дизайна и сам по себе си не е защитен от грешки.

Проектирането на базата на контракти е методология основана на идеята, че връзките и взаимодействието между различните компоненти на системата, особено критичните такива трябва да се управляват от точни и прецизни спецификации наречени контракти. Тези контракти включват изпълнението на задължения – входни условия и предлагането на конкретни резултати – изходни условия и инвариантни условия, които трябва да са винаги изпълнени.

Проектирането на базата на контракти е методология, която макар и много полезна не е широко застъпена. Поради тази причина тя е избрана за реализацията на системата разработена в дипломната работа и ще бъде разгледана по-подробно в следващата глава.

3. Използвани методологии и технологии

В тази глава са представени подробно методологиите и технологиите използвани за разработка на системата CodeContract - проектиране на базата на контракти, аспектно програмиране, AspectJ и Java 5 анотации.

3.1 Проектиране на базата на контракти

Методологията за проектиране на базата на контракти е разработена и приложена за първи път в езика за програмиране Eiffel, но вече е спечелила популярност и в други езици като Java и C++. Това е техника за проектиране на приложения, въз основата на точни, прецизни и проверими спецификации.

Обикновено софтуерните системи представляват набор от компоненти, които си взаимодействат. Архитектурата и работата на системата зависят изцяло от комуникацията между тези компоненти. В обектно-ориентираната архитектура тези компоненти се наричат класове. За да се гарантира коректността и способността за справяне с грешки на системата, е необходимо да се уточнят прецизно условията, при които компонентите си комуникират и да се осигури възможност за проверка на тези условия.

В този контекст се прилага метафората за бизнес контракти - концепцията за договаряне на условия между производители и потребители на даден продукт. В термините на обектно-ориентираното програмиране, производителят е отговорен за функционалността предлагана от даден клас, потребителят е отговорен за правилното използване на тази функционалност, а контрактите са условията, който дефинират тези взаимоотношения.

Проектирането на базата на контракти дава отговор на следните въпроси:

- Какви са отговорностите на даден клас
- Какви са отговорностите на потребителите на даден клас
- Как се обвързват отговорностите на потребителите с тези на производителите

Дефинирането на контракти гарантира, че ако потребителите и създателите на даден клас спазват определените за него условия, софтуерът ще бъде по-разбираем, по-надежден и с по-малко грешки.

В този контекст се дефинират различните видове условия на контрактите:

- Входно условие – задължение на потребителя на класа и полза за производителя, тъй като го освобождава от отговорност за случаи, които не са дефинирани.
- Изходно условие - задължение на създателя на класа и полза за потребителя, тъй като дефинира точно резултатът, който ще бъде получен.
- Инвариантно условие – дефинира определено състояние на обекта, което трябва да е изпълнено преди и след всяко извикване на метод на класа. Това е условие дефинирано за целият клас.

Условията на контракта имат следните особености:

- Условията винаги обвързват две страни – потребител и производител
- Условията са ясно описани и формулирани и нямат скрити клаузи
- Условията дефинират взаимните задължения, ползи и връзката между тях

Условията на контрактите се описват с булеви изрази, които се оценяват на истина или лъжа. Ако някое условие не бъде изпълнено, то приложението трябва да преустанови работа. В случай, че не е изпълнено входно условие – това означава дефект в клиента, който използва класа. Ако не е изпълнено изходно или инвариантно условие – дефектът е в самия клас.

Проектирането на базата на контракти има много предимства спрямо стандартните assertion техники:

- Контрактите позволяват създаването на смислови конструкции
- Контрактите определят дали дефинираното в тях условие е входно, изходно или инвариантно
- Контрактите могат да бъдат наследявани и са част от интерфейсите
- Осигуряват винаги актуална, достъпна и четима документация, която се генерира автоматично
- Грешките се прихващат близо до мястото, където са възникнали
- Контрактите допълват и подпомагат други техники, като разработването основано на тестване

3.2 Аспектно програмиране

Аспектното програмиране е нов подход в проектирането на софтуера, предназначен да решава проблеми с модулярността на системите. Това са проблеми свързани със сигурността, логването и транзакционното поведение, за които няма достатъчно добри решения предложени от други подходи, като структурното и обектно-ориентираното програмиране. Аспектното програмиране не замества, а допълва тези подходи.

Проблеми с модулярността възникват, когато определени услуги трябва да се вмъкнат и осигурят в различни части от дадено система. Аспектно-ориентираното програмиране възстановява модулярността чрез т.нар. аспекти или пресечни точки (cross-cutting concerns). Аспектите се разработват отделно от другите компоненти и след това се комбинират с тях. За целта се използват декларативни или програмни механизми, които не нарушават модулярността. Пресечните точки се дефинират веднъж и на едно място, което ги прави лесни за разбиране и за поддръжка. Процеса на “пресичане” или вплитане (weaving) на аспектите в останалия код може да

се извърши по време на компилацията или по време на работа на приложението. Този процес позволява да се добави код преди и след извикването на методи, да се подмени имплементацията им и дори да се промени състоянието и държанието на съществуващи класове.

3.3 AspectJ

AspectJ представлява аспектно-ориентирано разширение на езика за програмиране Java, създадено от Xerox PARC и достъпно като проект с отворен код в Eclipse Foundation. Достъпен е като самостоятелен проект и интегриран в средата за програмиране Eclipse. AspectJ вече е широко използван за реализация на аспектното програмиране. В него се използва синтаксис подобен на езика Java. AspectJ позволява различни начини за имплементация на аспектите: промяна на кода на приложението (sourcecode weaving), промяна на .class файловете (bytecode weaving) и промяна на класа директно във виртуалната машина по време на зареждането му. Всички класове променени от аспектите, са напълно съвместими с останалите.

3.4 Java 5 анотации

Java анотациите предоставят възможност за добавяне на мета данни към различни елементи на програмата като класове, методи, атрибути и параметри. Анотациите се дефинират като интерфейси, изпозлвайки знака @ пред името. Анотациите могат да бъдат записани в кода, в клас файла по време на компилация и дори да са достъпни по време на работа през т.нар. reflection. Така мета данните са достъпни за програмиста и по време на работата на приложението.

4. Архитектура на системата

При създаването на едно софтуерно приложение се следва стандартен процес за разработка, който включва:

- Събиране на изискванията към системата и създаване на спецификация
- Анализ на изискванията
- Проектиране на архитектурата
- Реализация на системата
- Тестване и валидация на системата

4.1 Изисквания към CodeContract

Първата основна стъпка включва събирането на всички изисквания към системата. Изискванията могат да бъдат разделени на:

- Функционални - описващи функционалността и услугите, които дадена система трябва да осигури
- Нефункционални - определят системни атрибути и ограничения към системата и процеса на разработване - платформа, база данни, език за програмиране, скалируемост
- Потребителски – неформално описание на функционалните и нефункционалните изисквания от потребителска гледна точка
- Системни - точно и детайлно описание на потребителските изисквания

4.1.1 Функционални изисквания

- Системата трябва да предостави поддръжка за методологията за проектиране на базата на контракти за Java 5 приложения

- Системата осигурява проверка на описаните входни, изходни и инвариантни условия по време на работата на тестваните приложения
- Ако проверката на някое условие пропадне, работата на приложението трябва да бъде прекратена с подходящо съобщение за грешка
- Системата трябва да поддържа следните механизми за наследяване в Java – наследяване на клас, наследяване на интерфейс и имплементиране на интерфейс.
- Поддръжка за наследяване на контрактите на методите с пълен достъп (public). Правилата за наследяване са в съответствие с принципа за заместване на Liskov (Liskov Substitution principle) и принципите за наследяване на контракти
- Допълнителна поддръжка за наследяване на контракти при
    - Методите с ограничен достъп (private, protected, package)
    - Статични методи
    - Конструктори
- Не изисква промени по сорс кода на приложенията
- Не изисква промени по процеса на създаване на приложенията

4.1.2 Нефункционални изисквания

- Системата да бъде създадена на Java 5.0
- Аспектно-ориентираният компонент да бъде разработен с AspectJ 1.5.2a или по късна версия

След като се дефинират изискванията към системата се пристъпва към определяне на актьорите и случаите на употреба на системата. Актьорите представят различните видове потребителски роли, а случаите на употреба работата на актьорите със системата.

4.1.3 Актьори

Изискванията към CodeContract определят два вида актьори:

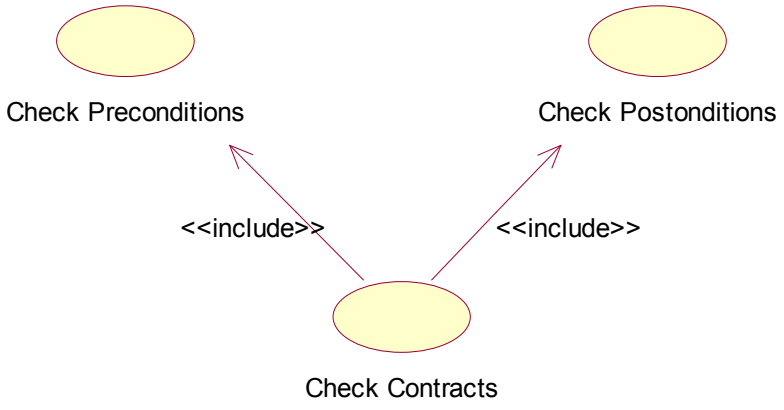
- програмисти, които разработват Java 5 приложения, базирани на контракти
- тестери, които валидират Java 5 приложения, базирани на контракти

4.1.4 Случаи на употреба

Случаите на употреба описват начините, по които актьорите използват системата. Всеки случай на употреба описва различен сценарий или поредица от действия, които системата извършва, при взаимодействие с актьорите за постигане на определен резултат или цел.

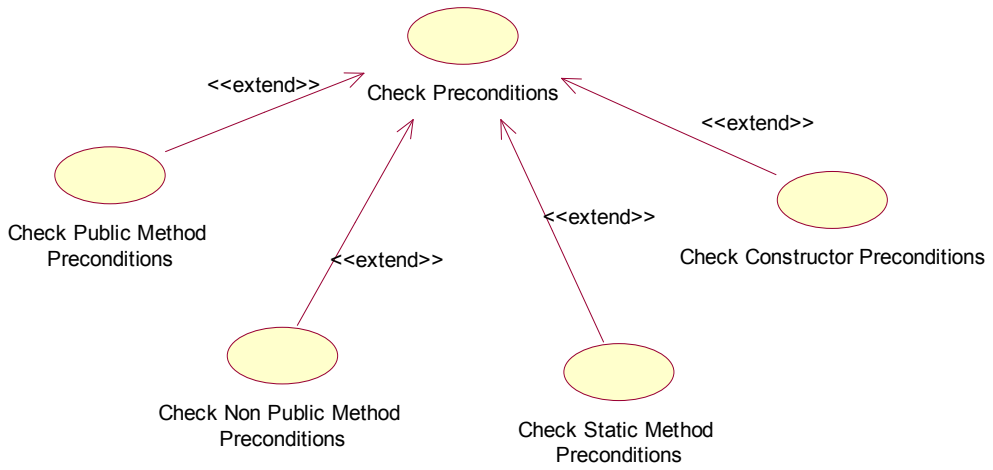
В системата CodeContract има един основен случай на употреба: *Проверка на контракти.* Този случай на употреба покрива проверките на контрактите описани в тестваното приложение. Той включва два под случая на употреба:

- *Проверка на всички входни условия*
- *Проверка на всички изходни условия*



фигура 1 Диаграма на случая на употреба Проверка на контракти

И двата под случая се наследяват от случаите на употреба представящи проверка на входни и изходни условия за методи с пълен достъп, методи с ограничен достъп, статични методи и конструктори.



фигура 2 Диаграма на случая на употреба Проверка на всички входни условия

Всеки от тези случаи на употреба включва няколко или всички от следните стъпки:

- Проверка дали класът, чийто метод е извикан дефинира контракт (анотиран с @Contract)
- Проверка на входните условия, декларирани за даден метод в текущия клас
- Проверка на всички изходни условия, за даден метод в йерархията от класове
- Проверка на изходните условия, декларирани за даден метод в текущия клас
- Проверка на всички инвариантни условия за даден клас в йерархията от класове

След като изискванията към системата са събрани и анализирани се преминава към проектиране на дизайна и архитектурата на системата. По

време на анализа и дизайна се създават по-детайлни UML диаграми, които описват подробно случаите на употреба.

4.2 Анализ на изискванията

При разработването на системата CodeContract първо се създават диаграми на взаимодействията. Идентифицират се основните класове, които участват в случая на употреба *Проверка на контракти*, и се описва взаимодействието между обектите на тези класове. Определят се основните видове класове:

- Boundary (гранични) – моделират взаимодействието между системата и нейните актьори. В CodeContract има един граничен клас – аспекта Interceptor.
- Control (контролни) – използват се за моделиране бизнес логиката и контрола на другите обекти. Това са ObjectHandler, ContractHandler и ExpressionHandler. В тях се съдържа логиката за обработка на обекти, извличане стойностите на контракти и оценка на изразите, представляващи контракти.
- Entity (обектни) – използват се за моделиране на дълготрайна информация, за нещата, които представят. Това са класовете Expression и четирите анотации Pre, Post, Invar и Contract. Expression съдържа информация за изразите, с които се описват контрактите

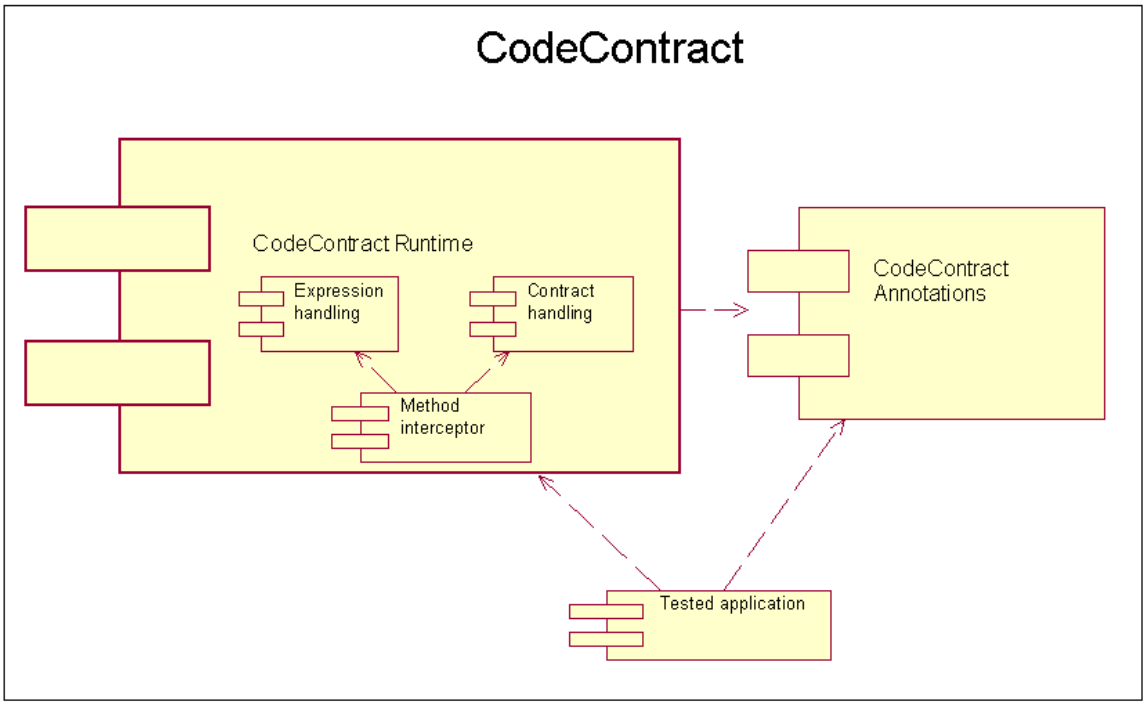
4.3 Дизайн и архитектура

Системният дизайн е третата фаза при разработката на софтуер. При нея се прави детайлно описание на системата спрямо избраните технологии и методологии. Описват се отделните компоненти и пакети, класовете и интерфейсите, които я изграждат.

Системата CodeContract е разделена на две основни части – компонент с анотации (CodeContract Annotations) за описание на контракти по време на създаване на тестваното приложение и компонент за проверка на тези контракти по време на работа на тестваното приложение (CodeContract Runtime). Компонентът за проверка на контракти съдържа три подкомпонента:

- Компонент за прихващане на извикванията на методите (Method Interceptor)
- Компонент за извличане и управление на контракти (Contract Handling)
- Компонент за оценяване на изразите, представляващи условия на контракти (Expression Handling)

Диаграмата на системната архитектура на CodeContract представя връзките между отделните компоненти на системата и тестваното приложение.



фигура 3 Диаграма на системната архитектура на CodeContract

След като основните компоненти и класове на системата са идентифицирани се преминава към създаването на клас диаграми. В клас диаграмата се представя връзката между граничните, контролните и обектните класове.

След това се пристъпва към дефиниране на точните алгоритми, по които ще бъдат извличани контрактите за даден метод, валидни в момента на извикването му. За целта се анализират детайлно изискванията за наследяване на контракти - принципа за заместване на Liskov (LSP) и принципа за наследяване на контракти. Изискванията определят реализацията на различни алгоритми за извличане на контракти при различни видове методи – с пълен достъп, с непълен достъп, статични и конструктори.

За двата най-сложни алгоритъма за извличане на контракти за методи с пълен достъп се създават диаграми на активностите. Алгоритмите се реализират в статичния метод getContracts () на контролния клас ContractHandler, който се използва при всяка проверка на входни и изходни условия. С него се определя пълният набор от контракти, които трябва да се оценят в случай на методи, които не са статични или конструктори. За извличането на контракти за статични методи се използва метода getStaticContracts (), а за извличане на контракти на конструктори - getConstructorContracts ().

Проверката на входни условия за метод с пълен достъп включва:

1. Проверка дали класът дефинира контракти (анотиран с @Contract)
2. В случай, че стъпка 1 не е удовлетворена се преминава към стъпка 4
3. Извличане на входните условия дефинирани в текущия клас
4. Извличане на всички инвариантни условия за класа

Проверката на изходни условия за даден метод с пълен достъп включва:

1. Проверка дали класът дефинира контракти (анотиран с @Contract)
2. В случай, че стъпка 1 не е удовлетворена се преминава към стъпка 4
3. Извличане на инвариантните условия дефинирани в текущия клас
4. Извличане на изходните условия дефинирани в текущия клас
5. Извличане на изходните условия на родителските класове и интерфейси

След диаграмата на активностите се създава диаграма на пакетите на системата. В нея се представят двата основни компонента:

- Пакет Annotations – съдържа класовете на анотациите, с които се описват контрактите
- Пакет CodeContract – съдържа аспектите и класовете, които реализират проверките на контрактите по време на работата на тестваното приложение.

При разработването на приложение, базирано на контракти е необходимо то да има референция към пакета с анотации, за да може да използва класовете Pre, Post, Invar и Contract за описание на контрактите и условията. Класовете в пакета CodeContract, също изискват референция към анотациите, за да се реализира логиката за извличане на контракти и проверката на условията, описани в тях.

За представяне на системната архитектура се използва диграма на пакетите. В нея са представени връзките между тестваното приложение (TestApplication) с системата CodeContract по време на тестването.

4.4 Имплементация

По време на имплементацията се създава спецификация на класовете и за всеки клас се изброяват значимите функции и атрибути. Методите на класа се описват с име, брой и тип на аргументите, тип на връщаната стойност. Атрибутите на класа се описват с тип, име и стойност по подразбиране.

В спецификацията на класовете на системата CodeContract са описани подробно следните класове:

- Contract – използва за маркиране на клас, съдържащ контракти
- Pre – използва се за описание на входни условия на метод
- Post – използва се за описание на изходни условия на метод
- Invar – използва се за описание на инвариантни условия на клас
- Interceptor – аспект, който прихваща извикванията на методи на класове, анотирани с @Contract
- ObjectHandler – представя обекта, за който се правят проверки на контрактите
- ContractHandler – използва се за извличане на контрактите за даден метод на клас
- ExpressionHandler – представлява контекста, в който се оценяват изразите, зададени в условията на контрактите
- Expression – представя отделно входно, изходно или инвариантно условие
- ContractException – изключение, което се хвърля при грешка възникнала по време на проверката на контрактите
- LogUtillity – помощен клас, който обвива функционалността предоставена от log4j класовете за логване на грешки и информация

В текущата имплементация се използват и два third party компонента, разработени от Apache Software Foundation и разпространявани с Apache Software License. Това са Log4j – широко разпространен пакет

предоставящ методи за логване на информация и commons-jexl, предоставящ методи за интерпретация на JEXL изрази.

Проектът commons-jexl е надграден в текущата имплементация на CodeContract за да предостави възможност за обработване на запазените ключови думи $return и $argsN. Те се използват при дефиниране на условия за аргументите или връщаната стойност на даден метод.

4.5 Ръководство за употреба

В ръководството за употреба на CodeContract версия 1.0 са включени следните точки:

- Концепции на програмирането на базата на контракти (DbC)
- Поддръжка на CodeContract за DbC
- Инсталация
- Как да използваме CodeContract
    - Описание на контракти в Java 5 приложения
    - Тестване на приложения с описани контракти
- Съществуващи проблеми

5. Тестване на системата

Тестването на системата е последната фаза от разработката. Целта е да се провери коректността на имплементацията на системата и съответствието и със спецификацията, по която е създадена. За да се постигне това се създава тест план, описват се тестовите процедури и необходимите тестови случаи. След това се проектират и реализират тестовите случаи. Накрая тестовете се изпълняват, резултатите се събират и анализират. При евентуални проблеми, грешките се отстраняват и тестовете се пускат отново.

За тестване на системата CodeContract се използват два различни подхода – unit тестване с използване на JUnit и ръчно тестване на цели сценарии. С junit тестовете се проверява правилната работа на основните класове от реализацията на системата: ContractHandler и ObjectHandler. С тестването на пълните сценарии се проверява правилната работа на CodeContract системата в четирите основни случая:

- условията на всички контракти са изпълнение (няма дефекти)
- не е изпълнено входно условие (дефект в клиента)
- не е изпълнено изходно условие (дефект в класа)
- не е изпълнено инвариантно условие (дефект в класа)

6. Съществуващи решения

В тази глава са представени три подобни софтуерни решения, предлагащи поддръжка за проектиране на базата на контракти за Java приложения - IContract, JContractor и Barter. За всеки от продуктите има кратко описание на възможностите, които предлага.

След това се прави сравнение със системата CodeContract, като основните компоненти, които се сравняват и оценяват са:

- Поддръжка на концепцията за проектиране на базата на контракти
  Всички продукти предлагат поддръжка за дефиниране на входни, изходни условия за методи и инвариантни условия. Всички предлагат възможност за описване на методи с пълен и ограничен достъп, статични методи и конструктори.
- Наследяване на контракти при класовете
  Приложенията IContract, JContractor и CodeContract поддържат наследяване на контрактите за методите с пълен достъп според принципа за наследяване LSP. Barter поддържа наследяване, но не според LSP. За останалите видове методи, всяко приложение има различна политика за пропагандиране на контрактите.
- Начин за описване на контрактите и използван език за описание
  Приложенията IContract, JContractor и Barter използват Javadoc формат за описване на условията на контрактите. В CodeContract се използват Java анотации.
- Инструментиране на кода на тестваните приложения
  IContract и Barter инструментират кода на приложението, който след това трябва да бъде допълнително компилиран. Jcontractor позволява инструментиране по време на компилацията и по време на работа. CodeContract инструментира кода по време на зареждането на класовете във виртуалната машина.

7. Заключение и насоки за бъдещо развитие

Софтуерната индустрия се развива с изключително бързи темпове и софтуерните приложения стават все по-сложни и все по-критични. Качеството, макар и относително понятие е най-значимата характеристика на софтуерния продукт. Качеството е съвкупност от много фактори, но двата най-важни са - способността на софтуера да изпълнява правилно своите функции при нормална работа и способността да се справя с непредвидени ситуации. Софтуерното инженерство непрестанно търси по-добри средства и техники за осигуряването им.

В конкретната дипломна работа се разработва аспектно–ориентирана система за тестване на софтуер наречена CodeContract. В основата е залегнала методологията за проектиране на софтуер на базата на контракти. Системата е лесна за научаване и употреба и предлага възможност за описание и проверка на контракти в Java 5 приложения.

Системата CodeContract е създадена, следвайки стандартен процес за разработка на софтуер. Първо се събират и анализират изискванията към системата. След това се проектира архитектурата с помощта на UML диаграми. Следва реализация, тестване и оценка на самата система.

Първата версия на CodeContract има следните възможности за развитие:

- Оптимизации на обработването на изразите, които описват условия
- Възможност за добавяне на информация към описанието на условие на контракт, която да се показва, ако условието не е изпълнено.
- Възможност за определяне на видовете условия, които да се проверяват.
- Възможност за оценяване на нестатични атрибути на класа в условията дефинирани за конструктори.