ПЛОВДИВСКИ УНИВЕРСИТЕТ „ПАИСИЙ ХИЛЕНДАРСКИ
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
КАТЕДРА „КОМПЮТЪРНИ СИСТЕМИ“

АСЯ ГЕОРГИЕВА СТОЯНОВА-ДОЙЧЕВА

ДЕФИНИРАНЕ НА ПРОЦЕС И СРЕДСТВА ЗА РЕФАКТОРИНГ В ОБУЧЕНИЕТО ПО СОФТУЕРНИ ТЕХНОЛОГИИ

АВТОРЕФЕРАТ

на дисертационен труд
за присъждане на образователна и научна степен “доктор”
в област 4. Природни науки, математика и информатика,
професионално направление 4.6 Информатика и компютърни науки,
научна специалност 01.01.12 Информатика

Научен ръководител: доц. д-р Минчо Сандалски

Рецензенти: акад. проф. д.т.н. Иван Попчев
проф. д.т.н. Тодор Стоилов

Пловдив
2011

Дисертационният труд е обсъден и насрочен за защита на разширено заседание на катедра „Компютърни системи“ при Факултета по математика и информатика на ПУ „Паисий Хилендарски“.

Дисертационният труд съдържа 154 страници. Използваната литература включва 165 източника, от които 25 интернет източника.

Списъкът с авторските публикации по темата съдържа 8 заглавия, от които 6 на английски и 2 на български език.

Защитата на дисертационния труд ще се състои на 11.10.2011 г. от 14:30 в заседателната зала на Факултета по математика и информатика на ПУ „Паисий Хилендарски“, гр. Пловдив.

Материалите по защитата са на разположение на интересуващите се в секретариата на ФМИ, нова сграда на ПУ, каб. 330, всеки ден от 8:30 до 17:00 часа.

Автор: Ася Георгиева Стоянова-Дойчева

Заглавие: Дефиниране на процес и средства за рефакторинг в обучението по софтуерни технологии

Тираж: 100 бр.

Пловдив 2011 г.

Съдържание

Обща характеристика на дисертационния труд

Реинженеринг на наследен софтуер и технологията рефакторинг са две актуални теми в обучението по софтуерни технологии във висшите училища. Възниква необходимост опитът получен в реални проекти да се предаде като знание на студентите. Цели се разработване на средства, които да подпомагат обучението в тези области.

В настоящата дисертация се представя един примерен процес за рефакторинг, получен на базата на практически опит с рефакторинг върху системата XCTL. Дефинирани са стъпките, които следва процесът, дейностите и документите, които се извършват и разработват по време на всяка една от тях.

Проектирани и разработени са прототипи на софтуерни средства за обучението по софтуерни технологии и рефакторинг. rLE (Refactoring Learning Environment) е прототип на средство, подпомагащо обучението по рефакторинг. Неговият основен модул е интелигентен агент за рефакторинг, който асистира на студента когато пише код. Целта е да анализира и оценява написан от студентите програмен код на Java, основавайки се на правила за рефакторинг, при това в реално време. Целта е интелигентният асистент (агентът за рефакторинг) да показва на студентите слабите места в написания от тях код на Java, в реално време и да им предлага възможности за подобряването му.

Разработен е и прототип на средство за създаване на динамични презентации на софтуерни процеси наречено eLSEBuilder. А също така и средство за навигация и управление на процеси за рефакторинг - STREPCILS.

Направено е предложение как представените средства в дисертацията могат да се интегрират в средата за електронно обучение DeLC.

Основни цели на дисертацията

Основните цели на дисертацията са:

1. ***Дефиниране на конкретен процес на рефакторинг, базиран на опита с XCTL.***
2. ***Създаване на концепция и разработване на агент за рефакторинг (Refactoring Agent) за подпомагане обучението по рефакторинг.***
3. ***Разработване на прототип на система за управление и навигация на процеси за рефакторинг – STREPCILS.***
4. ***Разработване на прототипна система eLSEBuilder за създаване на интерактивни и динамични презентации на софтуерни процеси.***

Структура и обем на дисертацията

Предложената дисертация се състои от 7 глави и използвана литература и 6 приложения. Обемът на основната част е 134 страници, а на използваната литература – 13 страници. Използваната литература включва 165 заглавия, от които 25 интернет източника. Списъкът с авторските публикации по темата съдържа 8 заглавия, от които 6 на английски и 2 на български език.

Глава 1 запознава с темата и обхвата на дисертацията. Описва основния подход, използван в работата и основните цели в нея;

Глава 2 прави преглед на съществуващите разработки и изследвания в областта. Дава кратък преглед на състоянието, в което се намира научната област;

Глава 3 дефинира примерен процес за рефакторинг на базата на получения опит с прилагане на рефакторинг върху системата XCTL. Представя разработването на софтуерно средство за управление и навигация на процеса за рефакторинг (STREPCILS);

Глава 4 разглежда проектирането и реализацията на среда за обучение по рефакторинг – Рефакторинг Агент;

Глава 5 представя разработването на проект и реализация на прототип на софтуерно средство за визуализация и симулация на софтуерни процеси – eLSEBuilder.

Глава 6 дава кратко описание на средата за електронно обучение DeLC и създаването на специализиран клъстер по софтуерни технологии.

Глава 7 обобщава свършената работа върху поставените цели в увода на дисертацията и представя някои възможни направления, в които може да бъде продължена разработката.

Кратко съдържание на дисертационния труд

Глава 1: Въведение

Софтуерните технологии стават все по-значима дисциплина в обучението по информатика и компютърни науки. Все повече университети по света включват в учебните си планове, курсове и дисциплини от тази област. Същевременно с това в съвременното обучение се наблюдава и бързо навлизане на електронното обучение на всички образователни равнища. Голям брой образователни институции наред с традиционните форми на обучение предлагат все повече учебни материали в електронна форма. Съвременните среди за електронно обучение предлагат гъвкаво обучение, достъпно по всяко време и от всяко място.

Основната идея за разработване на дисертационния труд възниква в рамките на международен проект "Joint Course Software Engineering " (JCSE) [1], реализиран по линия на Пакта за стабилност в Югоизточна Европа и подкрепен от Германската служба за академичен обмен DAAD. В проекта участват 10 висши училища, 9 от които са източноевропейски. Координатор на проекта е проф. Клаус Боте от Института по информатика на Хумболдтовия университет в Берлин. Една от основните цели на проекта е да се направи анализ на обучението по софтуерни технологии (вкл. reverse engineering) в участващите университети и да се разработи стратегия за изграждането на програма за обучение в тази област, съобразена с изискванията на европейската образователна политика.

Глава 2: Текущо състояние в областта

Необходимост от реинженеринг и рефакторинг

Съществуването на голям брой наследени софтуерни системи и необходимостта от това, те да бъдат подобрявани за целите и нуждите на своите потребители, води до дефинирането на специфичен процес за тази цел - реинженеринг [2]. Реинженеринг се налага в различни ситуации, когато софтуерът трябва да претърпи еволюция. Някой от тях са:

- разделяне на монолитни системи на отделни модули с цел по-лесното им управление;
- подобряване на производителността на системата;
- прехвърляне на системата от една платформа на друга;
- извличане на архитектурата на софтуерната система с цел по-нататъшна реализация;
- използване на нови технологии с цел намаляване цената за поддръжка;
- промяна на изискванията на клиентите.

Изводът, който може да се направи от горе изброените фактори, налагащи реинженеринг е, че в съвременните условия софтуерът се адаптира и модифицира непрекъснато, неговият код става все по-сложен и първоначалната му архитектура и структура се размива (губи). Поради тази причина основната част от цената за разработка на софтуер е съсредоточена в неговата поддръжка [3,4,5]. При приложението на добре познати и ефективни подходи за разработка на софтуер като например итеративен, еволюционен и други подходи, не е намерено решение на проблема, свързан със сложността на кода. Това е така, защото при тези подходи усилията на софтуерните инженери се съсредоточават върху разработването на нови изисквания, а по същото време софтуерът трябва и да се поддържа [6]. За да се реши проблемът със сложността

на кода, е необходимо да се приложат техники, които да намалят сложността му и да подобрят неговата вътрешна структура. Такива техники са преструктуриране (restructuring) [7] и рефакторинг (refactoring) [8].

Рефакторинг

Рефакторинг и преструктуриране са техники за реинженеринг на кода на софтуерни системи, като преструктуриране възниква като първа и служи за подобряване на кода на софтуер, написан на процедурни езици за програмиране, а рефакторинг за подобряване на код, написан с обектно-ориентирани езици за програмиране. И двете техники имат за цел да подобрят кода и структурата на софтуера и да запазят външното поведение на системата. По-долу ще бъдат представени някои от дефинициите за преструктуриране и рефакторинг

Подобни дефиниции за реструктуриране са дадени от Griswold в [9] [10], Arnold в [11] и от Chikofsky и Cross в [7], в следния смисъл - трансформация на една форма на представяне в друга, в относително едно и също абстрактно ниво, като се запази външното поведение на системата (функционалност и семантика). Трансформациите, предизвикани от преструктуриране, много често се появяват като промяна на кода за подобряване на структурата му в традиционния смисъл на структурното проектиране. Трансформации на езикови елементи могат да се реализират, ако е възможно да се дефинира критерий за трансформируемост при съблюдаване на синтаксиса и семантиката на конкретния език за програмиране [12]. Чрез преструктуриране се създават нови версии на системата, които изпълняват или предлагат промени, но обикновено те не предизвикват модификации, породени от нови изисквания.

Терминът *Рефакторинг* се дефинира за първи път от Opdyke в неговия дисертационен труд [13] в следния смисъл - процес на промяна на обектно-ориентирани софтуерни системи, без да се променя външното поведение на кода, но се подобрява неговата вътрешна структура.

Рефакторинг също е дефиниран от Fowler в [8] в следния смисъл – това е дисциплинирана техника за промяна на вътрешната структура на съществуващ обектно-ориентиран код, без да се променя неговото външно поведение.

Opdyke дефинира термина рефакторинг в своя дисертационен труд, като дава три основни шаблона за рефакторинг за езика C++. Тези шаблони са на комплексни рефакторинг, които той нарича Creating an Abstract Superclass, Subclassing an Simplifying Conditionals и Capturing Aggregations and Components. Идеята е в написания код да се търсят ситуации, отговарящи на предложените от него шаблони, като съответно тези места се променят. Комплексните рефакторинг се състоят от множество атомарни рефакторинг, които той описва в своята работа. Целта на Opdyke е да запази външното поведение на кода след приложение на промените и да подобри структурата му. Fowler има същата цел да запази външното поведение на кода и да го подобри, но той дефинира рефакторинг като последователност от малки стъпки и всяка стъпка също я нарича рефакторинг. В своята книга „Refactoring: Improving Designe of exsisting code", Fowler дефинира процеса на приложение на рефакторинг и описва множество от шаблони за промяна на кода за езика Java. Всяко място в кода, където трябва да бъде приложен рефакторинг, той нарича „smell code", а за да гарантира, че поведението на кода се е запазило и след прилагане на рефакторинг, той предлага да се създават и провеждат тестове преди и след приложението му. Тези тестове той нарича unit тестове.

В предложения дисертационен труд е използвана дефиницията на M. Fowler. Предложената от него техника за рефакторинг на малки стъпки позволява на софтуерните инженери да гарантират запазване на поведението на софтуера, а дори и при евентуална промяна лесно може да се определи точно откъде е възникнал проблемът. Тестовете, които се провеждат преди и след прилагане на рефакторинг, са още една допълнителна гаранция, че софтуерната система е запазила поведението си и осигуряват на софтуерните инженери лесен начин за тестване на приложените промени в кода.

Средства за рефакторинг за Java

Средствата, които са избрани за представяне в тази точка от работата, са свързани с прилагането на рефакторинг за програмния език Java и предлагат функционалност, която е подобна на функционалността на предлагания в дисертацията прототип на агент за рефакторинг. Целта на автора е да представи разликите между съществуващите средства за рефакторинг за езика Java и предложения в дисертацията агент за рефакторинг.

Всеки метод за рефакторинг се прилага в определена ситуация, когато са налице дадени условия в първичния код. Помощта, която технически средствата могат да предоставят на един разработчик в прилагането на рефакторинг, се свежда до два типа:

- Подпомагане в извършването на самите стъпки, описани в метода за рефакторинг до достигане на желаното състояние на кода – чрез менюта, помощници и т.н., с цел да се минимизира ръчната редакция на кода на много места от самия разработчик и от там допускането на грешки.
- Търсене в програмния код на условия (места), подходящи за прилагане на различните методи за рефакторинг и привличане вниманието на разработчика върху тях. Martin Fowler нарича такива места “smell code”.

В първия от описаните типове помощ (ако се вземе предвид светът на Java, което всъщност е напълно достатъчно) монополисти са изключително развитите към момента среди за разрабока като Eclipse [14], ItelliJ [15], NetBeans [16]. Те разполагат с много богат набор от методи за рефакторинг, като потребителят трябва да реши къде и кога да започне самия процес. За целта обикновено се селектира част от първичния код в редактора на средата за разработка и се активира помощен прозорец или съветник (wizard) чрез менюта или комбинации от клавиши.

Относно втория тип помощ не може да се каже, че съществува такова изобилие от средства, които наистина са създадени с цел да намират подходящи за рефакторинг фрагменти от програми. И все пак сред средствата, чиято цел е най-близо до търсенето на smell code, най-популярни са следните три FindBugs [17], Checkstyle [18] и PMD [19]. Те водят началото си от самостоятелни приложения, които като вход приемат множество от файлове с първичен (междинен за FindBugs) код и конфигурационен файл, а като изход генерират отчети относно намерените места, считани за подходящи за намесата на разработчика. В тези отчети като минимум присъства името на активираната проверка (по което може да бъде разбрано какви именно нежелани условия са налице), името на съответния клас и номер на реда, а също така и статистическа информация. И трите приложения към момента са с дълга история и имат множество опции: отчетите могат да се генерират в няколко формата; могат да се ограничават проверките, които се правят и т.н. И трите средства предлагат множество готови проверки, които се подразделят на групи. Въпреки завидно големите възможности на тези средства, в контекста на настоящата работа е целесъобразно да се споменат най-вече два факта:

- сравнително малко от проверките, реализирани в PMD, Checkstyle и FindBugs, са специално за откриване на недостатъци в дизайна на класовете, т.е. именно това, което е обект на рефакторинг. Повечето проверки са свързани с откриването на други лоши практики или по-скоро технически грешки – празни catch/finally блокове, излишни return клаузи, сравнение на низове по адрес и т.н..
- принципът на работа на тези средства се свежда до генериране на абстрактни синтактични дървета от първичния (междинния за FindBugs) код и подаването на тези дървета като аргумент на всяка една от проверките. Проверките от своя страна представляват класове-реализации на шаблона Visitor [20]: те имат свободата да предефинират интересуващите ги методи visit(treeNode), където treeNode е възел, в който алгоритмът за обхождане току-що е навлязъл, както и endVisit(treeNode), където treeNode е възел, всички наследници на който вече са обходени, т.е. алгоритмът е на път

да излезе нагоре от възела. Алгоритмът за обхождане на дървото е търсене в дълбочина и в трите средства.

Две от трите средства – FindBugs и PMD - предлагат графичен интерфейс, чрез който може да се визуализира синтактичното дърво на клас. Това улеснява писането на нови проверки, тъй като позволява да се види структурата на под-дървото на фрагмента от кода, който представлява интерес и трябва да може да бъде идентифициран от новата проверка. И трите средства се интегрират като plug-ins в средите Eclipse и NetBeans. Местата в кода, където проверките са сработили, се маркират визуално по подходящ начин в редакторите на средите. Две от средствата (FindBugs, PMD) поддържат и списък със задействаните проверки, където те могат да се игнорират и филтрират. Също така и трите приложения се предлагат и като Ant tasks и plug-ins за Maven, което позволява да се генерират отчети при построяване на проекта (build).

Отличителна черта на средството PMD е, че то допуска проверките да бъдат написани не само като Java-класове, но и под формата на предикати на езика XPath, който принципно се използва за обръщение към части от XML документи, но реално е приложим за всяка дървовидна структура от данни. Предимството на XPath, е че записът на активиращото условие в правилото е много по-кратък.

Всяко едно от описаните по-горе средства за рефакторинг предлага дейности, подпомагащи разработчиците на софтуер. Хората, които ги използват в една или друга степен, трябва да са специалисти в областта на рефакторинг. За обучаващи се тези средства са тежки за използване. Проблемът, който възниква за тях е, че те не дават информация за конкретната промяна, какво представлява тя и как да бъде приложена върху написания от обучаващия се код. Той трябва сам да намери източници (литература), от където да се запознае с нея и да разбере смисъла и. Имено от тук дойде идеята да бъде разработен прототип на софтуерно средство, подпомагащо обучението по рефакторинг – rLE (описано в Глава 4). Прототипът е предназначен за рефакторинг на Java код и предлага интерактивен подход за обучение по време на писане на кода от обучаващия се, което повишава креативността му.

Средствата eLSEBuilder и STREPCILS, предложени в дисертацията, са също предназначени за обучение по рефакторинг. Идеята за тяхното разработване възникна след дефинирането на примерен процес за рефакторинг в Глава 3 от работата. eLSEBuilder е рамка, предназначена за създаване на динамични презентации на софтуерни процеси. Прототипът на средството позволява на обучаващия да създаде симулация на софтуерен процес, която да се ползва за онагледяването му. Обучаващият има възможност да навигира презентацията посредством включването на интерактивни обекти в нея, като по този начин подобрява ефективността от обучението.

Прототипът на средството STREPCILS е предназначен за управление и навигация на процеси за рефакторинг. Основен мотив за неговата разработка беше автоматизирането на процеса за рефакторинг, като се използват за входни данни получени резултати от съществуващи средства за рефакторинг, които автоматизират конкретни стъпки от процеса, например за тестване XUnit [21], [22], за измерване JMetric [23] или Understand C++ [24], за откриване на местата, изискващи рефакторинг jFactor [25] и др. Прототипът може да бъде полезен на хора, които прилагат процес на рефакторинг или на обучаващи се в областта на рефакторинг, като им посочва предстоящи стъпки и дейности при прилагане на един такъв процес.

Глава 3: Дефиниране на примерен процес за рефакторинг

Наследената система XCTL

Системата XCTL е 16-битово Windows приложение, което се използва от Факултета по физика в Хумболдтовия университет в Берлин за определяне на характеристиките на силициеви кристали чрез облъчване с рентгенови лъчи [26]. В системата съществуват няколко вида сканирания:

- рентгенова топография – използва се за изследването на големи кристали (1 см на 1 см площ и до 10 000 кристални мрежови нива в дълбочина). Резултатът е изображение, което показва отклонението от идеалната кристална решетка. Пробата се сканира няколко часа, без да се променя ориентацията ѝ.
- рентгенова дифрактометрия – използва се за изследване на едно от мрежовите нива в кристалната решетка. Резултатът е крива, която показва промените на атомите вътре в кристала.
- рентгенова рефлектометрия – изследват се границите на повърхностите със слоеве. Резултатът е крива, която представя точното състояние на дебелината на слоевете и граничната повърхност на различните по химичен състав слоеве.

Основно XCTL е разработена с езика за програмиране C като някои допълнителни модули са програмирани на C++. Архитектурата не е ясно разделена на слоеве, например данни, логика и интерфейс. Документацията беше остаряла и неактуална, неотразяваща множеството промени, направени във времето. Част от разработчиците на системата са пенсионирани, други напуснали. Въпреки това системата беше в употреба. Изискването на потребителя беше да бъде прехвърлена от 16-битова към 32-битова платформа. И същевременно да бъде преведена в обектно-ориентирана форма. В тази ситуация съществуват две възможности – нова реализация или реинженеринг.

Обобщавайки ситуацията – реално и коректно опериране, липса на актуална документация и реалните разработчици, изискване за прехвърляне към 32-битова платформа – беше взето решение да се приложи реинженеринг. Подходяща реинженерингова техника в случая е рефакторинг, поради това, че прехвърлянето към 32-битова платформа ще се осъществи чрез модифициране на архитектурата в обектно-ориентирана като се запазва функционалността. Това прави системата подходяща за рефакторинг в съответствие с [8]. Освен това актуализацията се извършва по време на поддръжката на системата.

Резултатите от направения рефакторинг върху XCTL са следните:

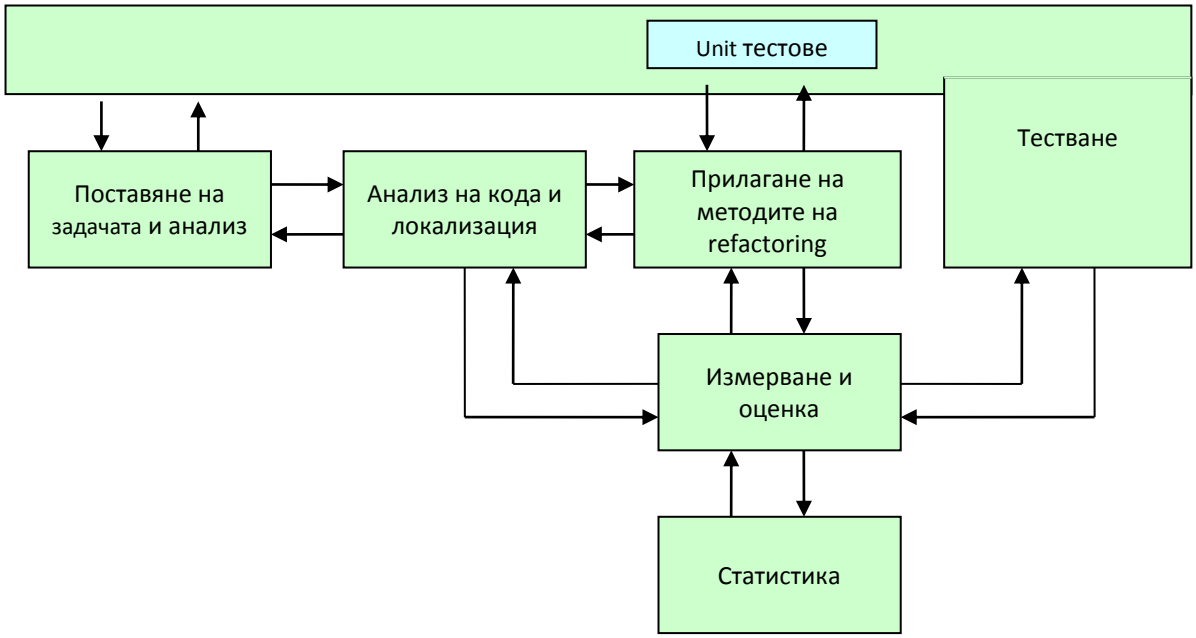
- описание на всички места в наследения код, върху които е приложен рефакторинг и описанието на конкретните методи за рефакторинг;
- описание на тестови случаи за unit тестове и резултатите от тях преди и след рефакторинг;
- описание на тестови случаи за функционални тестове и резултатите от тях преди и след рефакторинг;
- описание на измерванията, приложени върху системата преди и след рефакторинг;
- обновения код на XCTL след приключване на рефакторинг и обектноориентирана архитектура.

Дефиниране на примерен процес на рефакторинг

Рефакторинг процесът [27] предложен в тази секция от дисертацията е дефиниран на базата на опита получен с рефакторинг върху наследената система XCTL. Представени са основните фази с описание на дейностите, които се извършват в тях и документите, които се създават. Разработени са UML диаграми за по-добро онагледяване на стъпките във всяка фаза. Представени са и някои примери за апробация на фазите върху наследената система XCTL в приложенията.

Предложеният процес се състои от 6 фази (Фигура 1):

- поставяне на задачата и анализ;
- анализ на кода и локализация;
- приложение на методите на рефакторинг;
- тестване;
- измерване и оценка;
- статистика.



Фигура 1: *Софтуерен процес на рефакторинг*

В първата фаза *поставяне на задачата и анализ,* е необходимо да се прецени дали софтуерната система е подходяща за рефакторинг. Във фазата *анализ на кода и локализация* се определят всички места в кода, които са подходящи за рефакторинг и се определя кой точно метод за рефакторинг да бъде приложен. В следващата фаза *прилагане методите на рефакторинг* се прилагат методите на рефакторинг върху вече локализираните места. *Тестването* е фазата, която предлага различни типове тестове приложими в различните фази от процеса на рефакторинг. *Измерване и оценка* се правят с цел определяне дали даден метод на рефакторинг е подходящ в конкретна ситуация и дали приложеният рефакторинг е подобрил архитектурата на приложението. За целта в тази фаза могат да бъдат използвани различни метрики. Фазата *статистика* е предназначена за събиране на информация за даден проект, която би помогнала при следваща разработка. По-долу ще бъдат разгледани отделните фази от предложения процес и описани подробно чрез UML диаграми. Накратко е представена апробацията на фазите върху наследената система XCTL.

Поставяне на задачата и анализ - фазата на *поставяне на задачата и анализ* се описва в шест стъпки, които обхващат напълно проблема, който трябва да бъде решен от софтуерните инженери. Първата стъпка в сценария е свързана с опознаване на проблема и на наследения софтуер. Необходимо е разучаване на документацията, съществуваща за приложението и запознаване със средата, в която работи то. На базата на проучването разработчиците правят оценка на характеристиките на наследения софтуер и изискванията на потребителите и преценяват дали е възможно да бъде приложена техниката рефакторинг. За целта е необходимо да се проведат цялостни тестове върху наследения софтуер и да се разработи документ, съдържащ описанието на тестовите случаи и резултатите от тях. В ситуациите, когато софтуерът не работи коректно т.е. при тестване дава различни по произход грешки или съществуват други проблеми, свързани със средата, в която работи приложението или с възможностите за проучване на системата, то софтуерните инженери трябва да заключат, че наследената система не е подходяща за прилагане на процес на реинженеринг и следователно на техниката рефакторинг.

Фазата на поставяне на задачата и анализ е апробирана при рефакторинг на наследената система XCTL. Извличането на архитектурата на наследената система е извършено с помощта на съществуващата документация за системата XCTL [26] и различни симулации, които показват как точно работи роботът за настройка на пробите.

Анализ на кода и локализация - Анализът на кода и локализацията на местата, които се нуждаят от рефакторинг, е втората фаза от предложения по-горе процес на рефакторинг. Анализът на кода се прави на ръка в съответствие с методите за рефакторинг, дадени в [8]. Местата в кода, нуждаещи се от рефакторинг се наричат "bad smells" и трябва да бъдат локализирани с цел прехвърляне към друга платформа, добавяне на функционалност, разделяне на приложението на модули, откриване на грешки и др. Всяко такова място в кода, трябва да бъде отбелязано и описано с конкретния метод за рефакторинг. Описанието би могло да съдържа и подробности за конкретната ситуация в кода. Като неизбежна дейност е създаване на unit тестове за демонстрация на коректна работа на кода. Тестовете се създават и изпълняват върху кода, преди и след рефакторинг като целта е осигуряване запазването на поведението на кода.

При апробация на фазата анализ на кода и локализация върху модула Difractometry/Reflectometry от системата XCTL се създаде документ, в който е описано какви методи за рефакторинг и къде в кода те ще бъдат приложени. Основната стратегия, която се използва за unit тестване на XCTL, е комбинация от няколко метода за тестване – избор на контролна точка, избор на данни за проследяване, сравняване на стойности на променливи и провеждане на функционален тест.

Прилагане на методите за refactoring - След прилагане на конкретен рефакторинг върху локализиран код, трябва да се изпълнят unit тестовете, описани в предходната фаза. Ако резултатите са коректни т.е. съвпадат със записаните в документацията, и съществуват още локализирани места в кода, изискващи рефакторинг, се продължава със следващо прилагане на промените. Ако резултатите от unit тестовете не съвпадат с получените в предходната фаза, то е необходимо да се отстрани проблемът в кода, който предизвиква грешката. Тази стъпка се изпълнява, докато резултатите от unit тестовете преди и след рефакторинг съвпаднат.

Тестване - Тестването съпровожда целия предложен процес на рефакторинг . То е основна дейност, чрез която се осигурява запазване поведението на наследения софтуер. Прилагат се функционални тестове, установяващи коректната работа на приложението и unit тестове за всяко място в кода, нуждаещо се от рефакторинг. Unit тестовете се прилагат преди и след рефакторинг с цел да се осигури запазване поведението на променяния код. Резултатите от тестовете преди и след рефакторинг се сравняват като при коректни промени те съвпадат.

Тестването започва във фазата "Поставяне на задачата и анализ" с цялостни тестове на наследената система. Като резултат се получават тестови случаи за тестване на функционалността и резултати от тях. Във фазата "Анализ на кода и локализация" се продължава със създаване на unit тестове за всяко локализирано място в кода, нуждаещо се от рефакторинг. Тестовете се изпълняват и резултатите от тях се записват. Следващата фаза от процеса, в която се извършва тестване е „Прилагане на методите на рефакторинг". След прилагане на всеки рефакторинг се изпълнява unit тест, дефиниран в предходната фаза на процеса и се получават нови резултати . Възможните сценарии при сравняване на резултатите от unit тестовете преди и след рефакторинг са два:

- ако двата резултата не съвпаднат е необходимо да се отстрани грешката и тестовете да се проведат отново.
- ако резултатите от unit тестовете преди и след рефакторинг съвпаднат се смята, че променения код е запазил поведението си.

След успешно провеждане на всеки unit тест се прилага отново цялостен тест върху системата, за да се осигури, че тя е запазила функционалността си. Тази дейност е част от фазата "Тестване" от процеса за рефакторинг и получените резултати се сравняват с резултатите от тези тестове получени във фазата "Поставяне на задачата и анализ". При сравнение на резултатите са възможни следните сценарии:

- ако резултатите не съвпадат – грешката се търси в последно променяния код във фазата "Прилагане методите на рефакторинг" и след отстраняването и отново се изпълняват същите тестове;
- ако резултатите съвпадат и има още места в кода за промяна - продължава се с прилагане на следващ рефакторинг;
- ако резултатите съвпадат и няма други места в кода, нуждаещи се от рефакторинг, получаваме коректно променения код на приложението с документацията.

Измерване и оценка - Фазата *измерване и оценка* от предложения процес за рефакторинг е предназначена да подпомага софтуерните инженери в два аспекта:

- избор на метод за рефакторинг – това са ситуации, в които на дадено място в кода може да се приложи един или друг метод на рефакторинг;
- осигуряване на качеството на архитектурата – това е ситуация, в която софтуерните инженери искат да проверят дали са подобрили архитектурата на приложението.

От гледна точка на процеса, *измерване и оценка* могат да се правят по време на фазата *анализ на кода и локализация* и след прилагане на даден рефакторинг.

Статистика - Събирането на статистически данни е полезна дейност при завършване на един процес. На базата на такава статистика разработчиците могат да черпят информация за бъдещи разработки. Могат да предотвратят много от грешките, допуснати в минали разработки.

На базата на такава статистика може да се оцени оригиналната програма. Оценяването включва различни аспекти - как е била структурирана програмата, кои видове рефакторинг са били използвани и колко често са прилагани, могат да се опишат всички примери в кода на наследената програма като софтуерни структури, които не се препоръчителни, да се направи статистика за редовете код преди и след рефакторинг и да се определи обектната-ориентираност на програмата преди и след промените. За целта могат да се използват различни хранилища, в които да се събират данните за всеки един проект. Това би улеснило ръководителите на проекти при вземане на важни решения за текущ проект.

Средство за управление на процеси на рефакторинг – STREPCILS

Процесът на рефакторинг дефиниран в горните секции на тази глава, доведе до идеята да се създаде средство за управлението му и подпомагащо обучението в рефакторинг. Реализацията на това средство, наречено STREPCILS (Software Refactoring Process Tool for Code Improving of Legacy Systems), ще бъде представена по-долу.

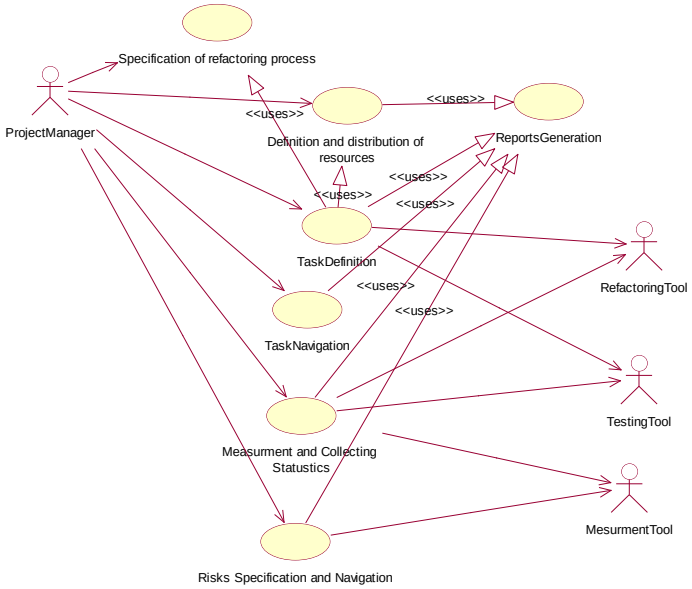
Ръководителят на проекта е отговорен за управлението на процеса. За подпомагане дейността му се предоставят автоматични средства, които улесняват работата. Едно софтуерно средство, чрез което се управлява и навигира процеса за рефакторинг е STREPCILS. То може да използва, като входни данни, резултати от други автоматизирани средства, подпомагащи процеса на рефакторинг. Такива средства могат да бъдат:

- средства за автоматичен рефакторинг;
- средства за измервания;
- средства за тестване.

STREPCILS реализира следните функции (Фигура 2):

- *Specification of refactoring process* – спецификация на процеса на рефакторинг за конкретен проект. Определят се фазите на процеса, през които ще премине разработката на текущия проект. В много от случаите може да се наложи някои от фазите да се прескочат;
- *TaskDefinition* – дефиниране на задачите. Необходимо е да се дефинират всички задачи (основни и подзадачи), да се определи времето, за което трябва да се свърши дадена задача и кой ще се заеме с нея;

- *Definition and distribution of resources* – дефиниране и разпределяне на ресурсите, нужни за проекта. Дефинирането на задачите е необходимо условие за разпределяне на ресурсите. Под ресурси трябва да се разбират човешки, софтуерни, хардуерни и времето;
- *TaskNavigation* – вече дефинираните задачи с горните две функционалности трябва да могат да бъдат навигирани, т.е. приложението трябва да дава възможност да се види развитието на проекта, да представя приключилите задачи, тези, които са в процес на работа и задачите, които предстоят;
- *Measurement and Collecting Statistics* – събиране на различни данни за статистика и за измерване на развитието на проекта. За тази цел могат да се използват други средства за измерване. Статистиката, която се събира, трябва да служи като основа за измерване на други величини, определящи самия проект – качество, сложност, големина и др.;
- *Risks Specification and Navigation* – описват се възможните рискове за проекта, като се категоризират в една от няколко категории. Ръководителят на проекта описва всеки възможен риск за проекта, разработва план за избягване на риска и управлението му, ако това стане необходимо;
- *ReportsGeneration* – генериране на различни справки.



Фигура 2: *Базови изисквания към системата STREPCILS – Use case диаграма.*

Средството е реализирано със C++Builder и е подходящ за използване от обучаващи се в рефакторинг – позволява им да следват стъпките от процеса, подсказва им какви документи и дейности следва да изпълнят във всяка една от тях.

Дефинираният процес на рефакторинг в тази глава от дисертацията е предназначен основно за обучението по софтуерни технологии. (С определени уговорки би могъл да подпомогне реални проекти). Отделните фази и описанието на всяка стъпка в тях дава ясна представа какво трябва да се прави във всеки един момент от проекта.

Дефинираният процес на рефакторинг има и някои недостатъци:

- процесът не дефинира явни условия за приключване на рефакторинг. Може да се изпадне в ситуация на прилагане на рефакторинг, произлизащ от друг. В такива ситуации понякога е трудно да се прецени кога да се спре;
- определянето на пълния списък от места в кода, нуждаещи се от рефакторинг може да затрудни по неопитните. Това се извършва във фазата “анализ на кода и локализация“ от дефинирания примерен процес за рефакторинг;

- лесно може да се стигне до неспазване на срокове за приключване на проект – една причина е произтичащите един от друг рефакторинг.

Предимство на процеса е добре структурираната последователност от стъпки със строго дефинирани дейности. Формализацията на стъпките с UML дава възможност да се създадат средства, подпомагащи изпълнението му като предложения прототип на STREPCILS за управление и навигация на процеса.

Глава 4: Интелигентна среда за електронно обучение

В тази глава от дисертацията е представена интелигентна среда за обучение по рефакторинг (rLE). Основна част от архитектурата е интелигентен агент за рефакторинг (RA), който изиграва ролята на асистент, подпомагащ студентите при обучение по рефакторинг. Околната среда на агента е създадена посредством интеграция на две среди – мултиагентна и среда за разработка.

RA предлага взаимодействие със студента, като при необходимост задава въпроси, извежда информация за методите на рефакторинг в диалогови прозорци, което характеризира средата rLE като интерактивна.

Средата реагира адекватно на действията на студента и предлага решение на конкретния проблем. RA дава възможност на студента да приложи определен метод на рефакторинг, подходящ за необходимите промени в кода.

Интелигентният асистент „следи“ кода, написан от студента в средата за разработка, и при необходимост се задейства, като маркира тези части в кода, които се нуждаят от промяна (рефакторинг). Това е един пример за проактивно опериране на средата.

Предложеният прототип rLE има следните различия от разгледаните в Глава 2 програми написани на Java :

- агентът е създаден за обучение на студенти, а не за професионално приложение;
- анализът на кода се извършва в реално време, докато студентът пише код;
- архитектурата на средата е агентно ориентирана.

Рефакторинг агент

Рефакторинг агентът, наречен RA (Refactoring Agent) е предназначен за подпомагане на обучението на студенти в курса по Софтуерни технологии. Основната му задача се състои в „следене” в реално време на кода, създаван от студентите, в средата за разработване и извеждането по подходящ начин на указания за подобряването му.

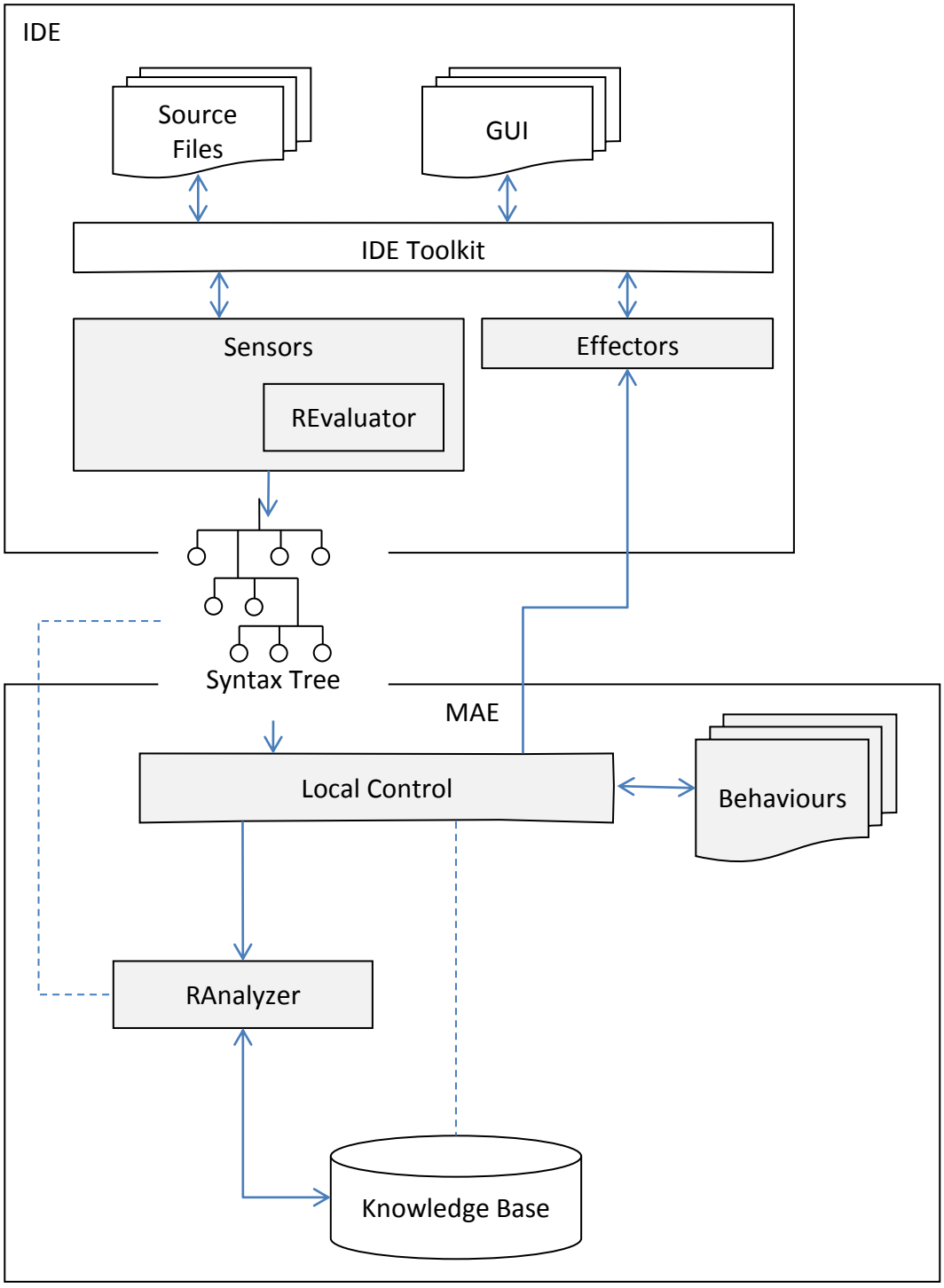
В зависимост от рефакторинг метода, който трябва да се приложи, агентът може да активира три базови поведения:

- автоматично прилагане на метода за рефакторинг (автоматичен рефакторинг) - в случаите, когато рефакторинг методът е сравнително прост и критериите за прилагането му са ясни, агентът може да предложи на потребителя автоматични да трансформации на кода;
- извеждане на подробно обяснение за мястото и начина за приложение на метода за рефакторинг (предложение за рефакторинг) - в случаите, когато критериите за рефакторинг са ясни, но прилагането на метода е свързано със значителни промени в кода, агентът информира потребителя за наличието на такава ситуация и му дава детайлни препоръки за подобряване на кода;
- анкетиране на потребителя за доизясняване на условията и последващо определяне на възможен метод за рефакторинг (анкета) - в определени случаи прилагането на един или друг метод за рефакторинг се определя от почти еднотипен набор от критерии и само някой от тях се различават. В случаите, когато част от условията за прилагане на рефакторинг методите са изпълнени, но не са достатъчни за еднозначно определяне на точния рефакторинг, агентът може да “попита” потребителя за доизясняване на

конкретната ситуация.След доизясняване на условията, агентът отново определя типа на ситуацията: автоматичен рефакторинг или предложение за рефакторинг.

Среда за обучение по рефакторинг

Рефакторинг агентът трябва да оперира в някаква среда, която в случая е средата за обучение, подпомагаща студентите при подобряване на написания от тях код посредством прилагане методи за рефакторинг [28]. Архитектурата на средата за обучение по рефакторинг е представена на Фигура 3.



Фигура 3: *Архитектура на rLE*

Околна среда

Един агент е автономен, реактивен и проактивен софтуерен модул, който оперира в някаква околна среда. За RA околната среда е интеграция на две среди – мултиагентна и развойна. От една страна, студентът работи в някаква развойна среда (IDE), използвана в обучението. Тази среда доставя необходимите средства за разработване на първичен код. Като минимум развойната среда трябва да предоставя:

- редактор на код;
- компилатор и/или интерпретатор;
- средства за създаване на изпълними приложения;
- дебъгер.

От друга страна, агентът оперира в своя оперативна среда (MAE – Multiagent Environment), която притежава следните основни характеристики:

- предоставя инфраструктура с комуникационни протоколи и протоколи за взаимодействие между агентите;
- отворена архитектура;
- поддържа децентрализирано управление;
- предназначена за автономни и разпределени софтуерни компоненти, които могат да оперират самостоятелно или да взаимодействат с други.

Освен това агентът трябва да „знае“ какво прави студентът, т.е. необходим му е достъп до развойната среда. За целта се предлага комплексна среда, получена като интеграция между IDE и MAE. За да се осъществи интеграцията, сензорите и ефекторите на агента са разположени изцяло в средата за развой, а останалите компоненти – в MAE. По този начин става възможно взаимодействието между потребителя, който работи само с развойната среда, и агента, който функционира в комплексната (интегрираната) среда.

Сензори

Агентът за рефакторинг (RA) комуникира с околната си среда посредством своите сензори и ефектори. Сензорите изпълняват следните задачи:

- осигуряват пълен достъп до първичния код и другите информационни ресурси, създадени и използвани от студентите;
- предоставят общ „поглед” върху работата и действията на студентите;
- трансформират първичния код в удобно за изследване от агента представяне;
- събират и предават необходимата оценъчна информация.

Сензорите включват два основни модула – RParser и REvaluator. RParser трансформира създадения от студента първичен код в представяне, удобно за анализ и прилагане на методите за рефакторинг. Избраното представяне е вид синтактично дърво, съдържащо основните синтактични елементи, връзките между тях и структурата на първичния код. Синтактичното дърво съдържа необходимата информация за локализиране на подходящите за рефакторинг места. То обаче, не доставя информация за избор на възможни методи за рефакторинг. За целта е необходима допълнителна оценъчна информация, която се осигурява от REvaluator. Примерна оценъчна информация могат да бъдат брой редове на първичен код на даден клас/метод, броя методи/атрибути на даден клас и др.

Ефектори

Ролята на ефекторите е да предизвикват различни събития в околната среда, които да подпомагат изпълнението на задачата на студента. Такива събития могат да бъдат:

- маркиране на места в кода, подходящи за рефакторинг;
- оцветяване на текст;
- предизвикване диалог с потребителите;

- генериране на съобщения и звукови сигнали;
- анимиране на агента за подсилване ефекта от помощта.

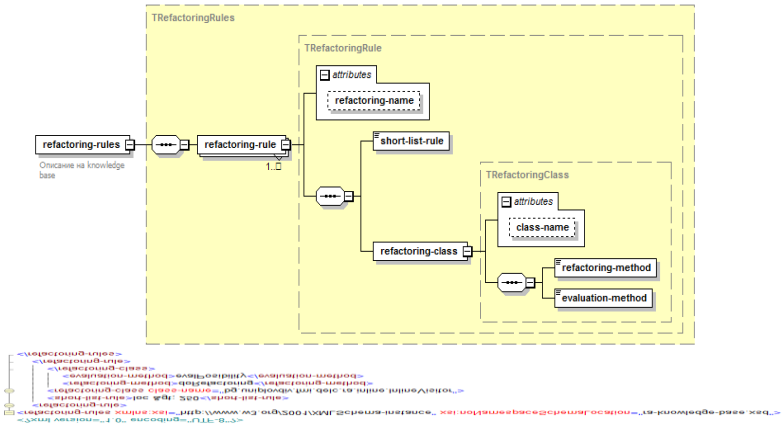
За визуализиране на събитията и осъществяване комуникация със студентите ефекторите използват услугите на графичния потребителски интерфейс на средата.

Друга основна задача на ефекторите е реализация на обратната трансформация - на синтактичното дърво в първичен код.

База знания

Базата знания на агента е множество от правила, наречени Refactoring Rules (RRs). Всяко правило RR се състои от две части (Фигура 4):

- Short-list-rule – мета-правило, което се използва за генериране на списък с потенциално приложими правила за рефакторинг. Правилата подпомагат провеждането на предварителна селекция на рефакторинг методите (наречена „груб анализ"). Те представят условия, удовлетворяването на които предизвиква включване на анализирания метод в „късата листа". Активирането на правило предизвиква pattern-matching операция, използваща доставяната от сензорите оценъчната информация. Така напр., short-list-rule за рефакторинг метода "Extract class" (описан в **Error! Reference source not found.**) може да бъде „LOC_by_class > predefined_value". Активиране на правилото присвоява стойност (оценъчна информация) на променливата LOC_by_class, която впоследствие се проверява дали е по-голяма от предварително дефинираната константа predefined_value. При удовлетворяване на условието "Extract class" се включва в „късата листа". Включването на „груб анализ" е от съображения за ефективен run-time на агента;
- Refactoring-class – класовете се състоят от две части – evaluation-method и refactoring-method. Evaluation-methods се използват за оценка на потенциално приложимите методи от „късата листа" за реална приложимост (наречен „детайлен анализ"). Refactoring-methods реализират кореспондиращите към метода трансформации на синтактичното дърво.



Фигура 4: *Структура на Refactoring Rule*

Съществен момент за изграждане на базата знания е намиране подходяща формализация на методите за рефакторинг. Представената по-горе структура е предложена след анализ на голям брой методи, описани в специализираната литература (основно в [8]). Трябва да се отбележи също и факта, че някои от известните методи не се поддават на предложената формализация.

RAnalyzer

Изборът на приложими методи за рефакторинг минава през два етапа – предварително филтриране (груб анализ) на потенциално приложимите методи и детайлен анализ за намиране на реално приложимите методи за конкретната структура на програмния код. RAnalyzer осъществява първия етап, като:

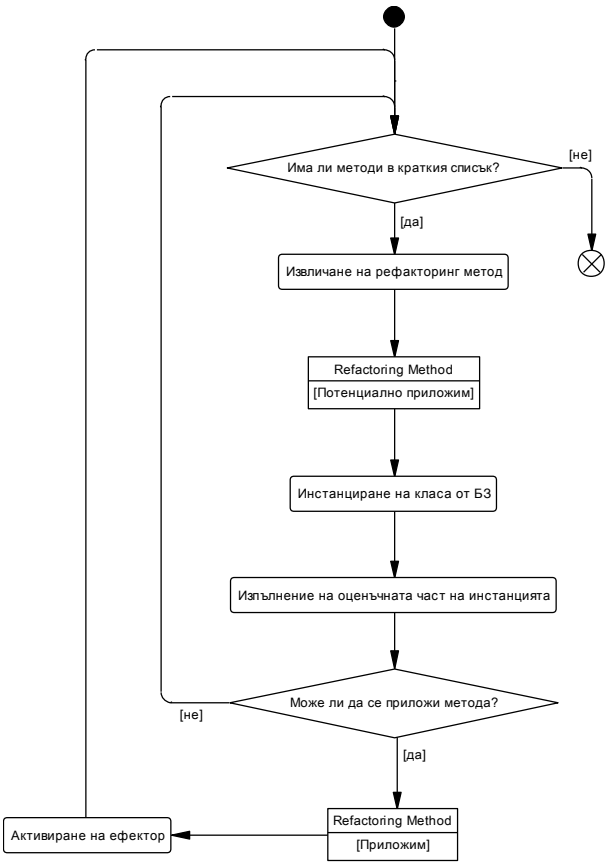
- прави „груб" анализ на синтактичното дърво – за целта използва short-list-rules от базата знания;
- генерира „къса листа" на потенциално използваеми методи за рефакторинг.

Детайлният анализ се извършва от локалното управление.

Локално управление

Локалното управление реализира жизнения цикъл на RA, като изпълнява следните задачи:

- синхронизация работата на отделните компоненти на агента;
- инстанциране и изпълнение на предоставените от refactoring-classes методи;
- активиране ефекторите на агента;
- управление поведенията на агента.



Фигура 5: *Алгоритъм на локалното управление*

Алгоритмът на локалното управление е следният (Фигура 5):

1. за всеки метод от късата листа се извлича класа на потенциално приложимия рефакторинг метод от базата знания;
2. създава се инстанция на този клас;
3. изпълнява се оценъчната част на инстанцията;
4. ако извършеният детайлен анализ покаже, че методът е приложим:

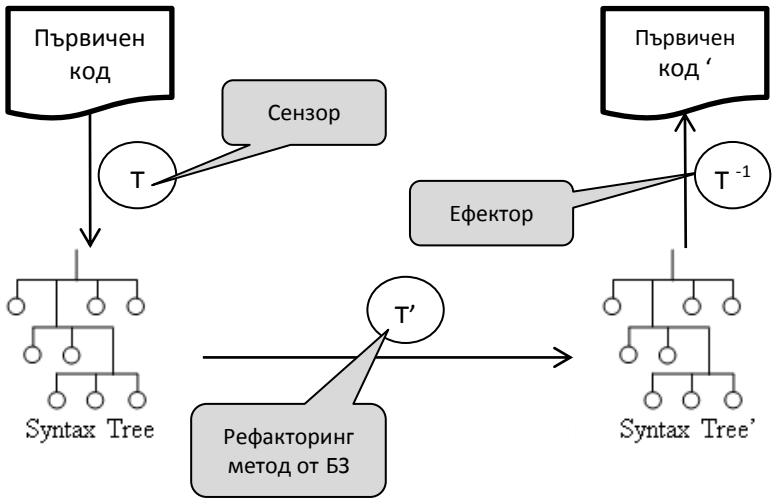
   а) сменя се статуса на метода в „реално приложим";

b) активира се ефекторът за да се маркират местата в кода, които имат отношение към този рефакторинг метод;
5. ако методът не е приложим той се игнорира.

Схема на трансформациите на RA

На

Фигура 6 е даден общ преглед на трансформациите, извършвани от агента. Разработеният от студента първичен код се трансформира от сензорите (трансформация T) в синтактично дърво. Локалното управление, активирайки refactoring-metod частта на избрания метод, предприема трансформации (T') върху актуалното синтактично дърво (Syntax Tree), в резултат от което се генерира модифицирано синтактично дърво (Syntax Tree'). Накрая, ефекторите трансформират обратно (трансформация T^{-1}) резултатното синтактично дърво в модифициран първичен код, който се предлага на студента.



Фигура 6: *Схема на трансформациите в RA*

Потребителски интерфейс

Потребителският интерфейс дава възможност на студентите да взаимодействат със средата. Съществуват различни възможности за включване на интерфейса:

- като интегрирана част на околната среда на агента;
- като надстройка на околната среда на агента;
- като надстройка на част от околната среда на агента.

В работата е реализирана третата възможност.

Реализация на средата

Реализацията на горе описаната архитектура и функционалност на средата за обучение по рефакторинг [29] беше изпълнена в два етапа:

- създаване на околната среда на RA;
- реализация на RA.

Характерното за околната среда на RA е, че е интеграция между две развойни среди – Eclipse, като IDE, и JADE като среда за разработка на мултиагентни приложения. Сензорите и ефекторите на агента са разположени в Eclipse, а локалното управление, RAnalyzer, поведенията и базата знания оперират в JADE.

Eclipse [14] е избран като IDE на предложената в предходната точка архитектура на агента за рефакторинг е избран Eclipse, поради следните причини:

- широко разпространена среда за развой;
- предназначена основно за разработване на Java приложения;

- с отворен код;
- позволява взаимодействие с външни компоненти под формата на т.нар. plugins (набор от софтуерни компоненти, които добавят специфична функционалност към по-голяма софтуерна система); Това основно предимство на Eclipse дава възможност рефакторинг агентът да бъде интегриран в средата за разработка;

За реализация на мултиагентната среда (MAE) е избран JADE (Java Agent Development Framework) [30]. Основните причини са:

- широко разпространена и добре описана рамка за агентно-ориентиран софтуер;
- среда с отворен код, изцяло реализиран на Java;
- съответства на FIPA спецификациите (IEEE Foundation for Intelligent Physical Agents) [31];
- в контекста на рефакторинг агента, избора на JADE като технология подпомага успешното интегриране на околната среда, синхронизиране работата на отделните компоненти на агента, прозрачна агентна комуникация и др.

Съществен критерий за избора и на двете среди е, че са Java базирани. Това прави възможно интеграцията между тях да стане на сравнително ниско ниво – единна Java виртуална машина. Интерфейсът между двете среди се осъществява чрез plugin-модела (RA Plugin), който реализира функциите по стартиране на JADE контейнера и осъществяване на връзката между JADE и Eclipse. По този начин взаимодействието между агента и средата за развой става прозрачно за потребителя. Потребителят вижда само резултата от анализа извършен от агента под формата на съответни маркировки в кода и/или активиране на диалог с агента.

Реализацията на описаната архитектура на RA е представена подробно в дисертационния труд.

Обобщавайки основните резултати, представени в тази глава, искаме да обърнем внимание на някои проблеми. За разработване на RA, предназначен за целите на дисертацията, в специализираната литература не са намерени готови модели. Предложената архитектура е резултат на задълбочен анализ на естеството и предназначението на описаните в някои от цитираните източници (основно [8]) методи за рефакторинг. Разгледани са 32 метода. Може да се отбележи, че повечето от проучените методи могат да бъдат реализирани като автоматични методи, по-малка част като предложения за рефакторинг и най-малка като анкета.

Друг проблем е, че методите са описани само вербално. Беше необходимо да се намери подходяща формализация. В базата знания на агента методите (RRs) се представят като правила. За всяко отделно правило е предложена обектно-ориентирана интерпретация. В актуалната версия са имплементирани 11 правила; останалите 21 са апробирани на ниво спецификации.

При анализа на литературните източници са идентифицирани около 100 правила за рефакторинг. За останалите (около 70) е проведен само предварителен анализ за формализируемост и реализируемост в предложената архитектура. Най-общо може да се обобщи, че част от правилата с голяма вероятност ще могат да се формализират и реализират, друга част – евентуално, а за трета група (засега) остава неформализуема.

Системата е апробирана с избрани студенти в практикума по рефакторинг, включен в магистърската програма „Софтуерни технологии” на ФМИ на Пловдивския университет.

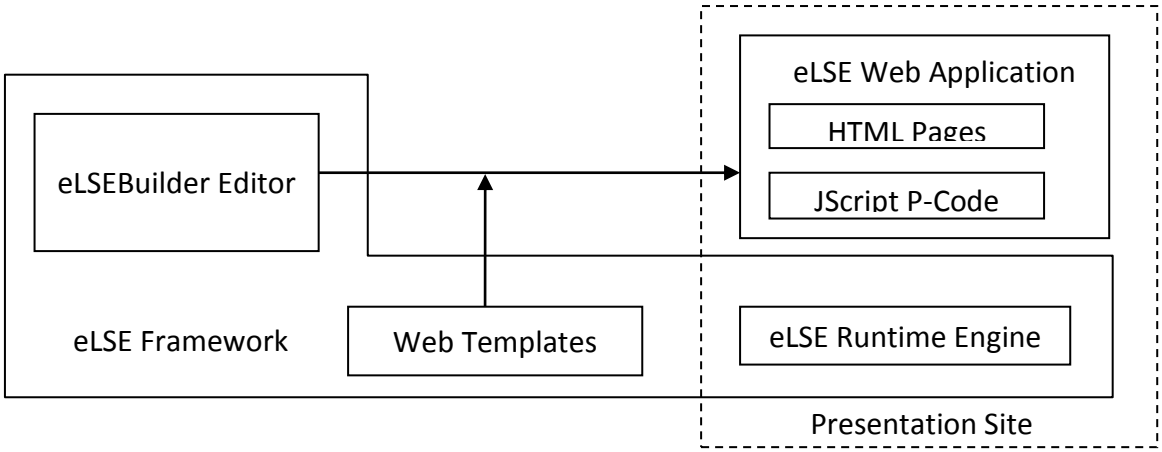
Глава 5: Визуализация и симулация на софтуерни процеси

Основен дял в обучението по софтуерни технологии заема изучаването на софтуерни процеси. С цел успешното им усвояване от студентите е добре да бъдат визуализирани. Симулирането на софтуерни процеси позволява студентите да проиграят различни сценарии, през които преминават тези процеси. За целта сме проектирали и разработили средството eLSEBuilder.

eLSEBuilder

eLSEBuilder е софтуерно приложение, което позволява създаването на динамични презентации и симулации на софтуерни процеси. Средството може да се използва от преподавателя за онагледяване на материята по софтуерни технологии. eLSEBuilder може да се използва както в обучението по софтуерни технологии, така и в други области (например в [32] е представено използването му за симулиране на телекомуникационни процеси в железопътния транспорт).

eLSEBuilder се състои от три основни модула– eLSEBuilder Editor, eLSE Runtime Engine и eLSE Web Application. Фигура 7 показва основните модули и връзките между тях.



Фигура 7: *Модулите на eLSEBuilder*

eLSEBuilder Editor е клиент/сървър приложение, което се използва, за да се дефинират и създават ***презентации.*** Това средство използва СУБД (MS SQL Server 2000) за организиране и съхраняване на пълната информация за симулациите и презентациите. eLSEBuilder имплементира две основни групи от функции:

- Създаване на презентации (модели, обекти, скриптове, симулации и т.н.)
- Генериране на ***Презентационно уеб приложение (Presentation Site)*** - тази процедура генерира уеб сайт, който включва пакет от предварително създадени симулации, необходими за специфична презентация. Този уеб сайт се състои от html страници и специално създаден код (p-code) в JavaScript файлове.Този сайт не изисква връзка към базата данни, нито пък специален приложен сървър, за да бъде стартиран и поради тази причина е лесно преносим – на лаптоп, флаш памет и др.

eLSE Runtime Engine е runtime машина, базирана изцяло на JavaScript, която се използва, за да интерпретира генерирания презентационен сайт в html браузър. Това на практика не е средство, а библиотека от функции, написани на JavaScript, които се използват за създаване на динамиката в презентациите. Те използват DHTML и кода (p-code), генериран в презентационния сайт от eLSEBuilder Editor за всяка от симулациите, за да изпълнят презентацията в ***eLSE Web Application***.

eLSE Web Application е средството, което използва крайният потребител (преподавател), за да демонстрира презентацията. Това е уеб приложение, което се изпълнява в html браузър и използва подготвения от eLSEBuilder Editor презентационен сайт. eLSE Web Application показва презентацията в няколко рамки, предоставяйки на потребителя набор от бутони (ControlToolbox) за контролиране на изпълнението на презентацията. За всяка от симулациите, включени в презентационния сайт, може да се изпълняват действията: старт, стоп или изпълнение стъпка по стъпка.

eLSEBuilder разполага с възможност за експорт (и последващ импорт) на симулационни модели, описани на език, базиран на XML. Форматът е създаден специално за eLSEBuilder и позволява в описанието да се включи не само сценарият на симулацията, но и всички обекти (изображения напр.), които се използват в симулацията.

Прототипът на програмната рамка на eLSEBuilder е предназначена за подпомагане процеса на обучение по софтуерни технологии. Но той би могъл да бъде използван за моделиране и на други процеси, които не са свързани директно със софтуерните технологии, въпреки че първоначалната идея възникна именно във връзка със симулирането на софтуерни процеси.

Глава 6: Специализиран клъстер към DeLC

Описаните в предишните глави средства за електронно обучение могат да се използват изолирано за подпомагане на студентите за овладяване на учебен материал по специфична тема на технологията рефакторинг. В последните години във Факултета по математика и информатика на Пловдивския университет „Паисий Хилендарски" се изгражда инфраструктура за доставка на електронни образователни услуги, наречена DeLC (Distributed eLearning Center) [33,34]. В тази глава се предлага една първоначална представа за интегриране на средствата в архитектурата на Центъра.

Разпределен център за електронно обучение

От концептуална гледна точка DeLC е отворена мрежова инфраструктура, където отделните възли (наречени „образователни") са интелигентни хранилища за доставка на електронни образователни услуги и електронно учебно съдържание [35,36]. Възлите моделират реални образователни единици (лаборатории, катедри, факултети, колежи или университети), предлагащи пълен или частичен цикъл на обучение. Възлите могат да се интегрират, създавайки виртуални структури, наречени клъстери. Клъстерите са удобни когато при изпълнение на една заявка е необходимо използването на информационни ресурси, разположени върху различни възли.

Възлите се характеризират посредством атрибути. Някои от основните атрибути са следните:

- Тип на възела – в актуалната версия на центъра са дефинирани следните типове: Базови(оперират като самостоятелни доставчици на услуги и съдържание), Помощни (оперират съвместно с определен базов възел, като доставят различни помощни средства, разширяващи възможностите на базовия възел. Обикновено са прозрачни за потребителите), Специализирани (разширяват възможностите на даден базов възел със специализирани средства);
- Достъп – достъпът до електронните услуги и електронното съдържание на един възел може да бъде фиксиран и мобилен.

Ядрото на DeLC са два възела, които се използват за организация и провеждане на електронно обучение във ФМИ. Първият възел, DelC Portal, е с фиксиран достъп до услугите и електронното съдържание. В следващия раздел образователният портал е представен подробно. Вторият възел предоставя трислойна архитектура, осигуряваща мобилен достъп до услуги и информационни ресурси посредством интелигентни безжични точки на достъп (наричани Information Stations - ISs), разположени около сградата на университета. IS възелът е представен детайлно в различни публикации (напр. [37,38,39,40]).

DeLC е създаден като интерактивна и проактивна среда за персонализирана доставка на образователни услуги и учебно съдържание [41]. Такава среда може да бъде изградена основно посредством подходящи архитектурни решения, като: използване агентно- и сървисно-ориентирана архитектура и интелигентни компоненти, представяне и обработка на електронно съдържание в съответствие със съществуващите стандарти (напр. SCORM 2004), ясно и явно

разграничаване на съдържание (данни) и описание (метаданни), използване на богата профилна информация [42,43].

Клъстер „Софтуерни технологии"

Интеграцията на представените в дисертацията средства за електронно обучение в архитектурата на DeLC има съществени предимства. Специализираното използване на средствата не се губи – само че, те трябва да се активират през потребителския интерфейс на портала. Посредством интеграцията, обаче, е възможно изпълнението на комплексни образователни сценарии за постигане на определени педагогически цели, реализирани като комбинации между услугите, доставяни от образователния портал (усвояване на нови знания чрез електронни лекции, проверка на знания посредством системата за електронно тестване) и специализацията на средствата за рефакторинг.

Съществуват две възможности за интегриране на средствата за рефакторинг в архитектурата на DeLC. Първата възможност е директно интегриране в образователния портал, т.е. съществуващият възел се разширява с нови функционални компоненти. Втората възможност е изграждане на специализиран клъстер, подпомагащ електронно обучение в дисциплината „Софтуерни технологии" [44]. От технологична и програмно-техническа гледна точка, първият подход е може би по-лесен за реализиране – в един реинженерингов процес съществуващите средствата за рефакторинг трябва да бъдат преструктурирани като електронни услуги. Вторият подход е по-близък до философията за разширение на DeLC – посредством изграждане на клъстери. В този случай основният пакет образователни услуги (лекции, упражнения, тестове) ще се предоставя от портала и заедно със специализирания възел ще образуват клъстер.

За интеграцията на представените в дисертацията средства (SREPTICS, rLE, eLSEBuilder) се използва възможността за изграждане на клъстери в инфраструктурата на DeLC. Специализираният клъстер „Софтуерни технологии" ще включва два образователни възела – базов и специализиран:

- базовият възел - образователният портал DeLC - доставя електронно съдържание за областта „Софтуерни технологии" и електронни услуги за неговото използване;
- специализиран възел eLSE (e-Learning in Software Engineering) – във възела ще бъдат интегрирани SREPTICS, rLE и eLSEBuilder. Освен това, тук е инсталирана адаптация на средата S-Bahn-Tool, с която е възможно автоматизиран превод на учебен материал.

Чрез изграждането на клъстера се дава възможност за създаване на по-гъвкави сценарии за обучение по софтуерни технологии. Основните знания в тази област студентите ще получава от базовия възел, а специализирани и допълнителни знания от специализирания възел.

За да бъде реализиран специализирания възел архитектурата на предложените средства трябва да бъде променена. В момента те са реализирани като stand alone приложения. Необходимо е те да бъдат реализирани като услуги. За целта средствата ще бъдат разширени с web services.

Глава 7: Заключение

Дисертацията представя изследване в областта на софтуерните технологии и по- специално рефакторинг.

Използвайки резултатите и практическия опит, получени от реинженеринга на наследената система XCTL, е предложен примерен процес за рефакторинг, предназначен основно за обучение на студенти.

В дисертацията са представени три специализирани средства, които могат да се използват за подпомагане на обучението по дисциплината „Софтуерни технологии" – STREPCILS, rLE и eLSEBuilder.

В дисертацията е дадена идея и първоначална представа за изграждане на специализиран клъстер, като разширение на DeLC. Освен интегриране на трите средства

(STREPCILS, rLE, eLSEBuilder), в клъстера е адаптирана софтуерната среда S-Bahn-Tool. Адаптацията е използвана за автоматизирано изграждане на терминологичен българо-английски речник по софтуерни технологии.

Някои от представените средства са тествани в упражненията по дисциплината „Софтуерни технологии” (бакалаври) и практикума „Реинженеринг” (магистри).

Таблица 1 показва връзката между приносите, структурата на дисертацията и направените публикации.

Впечатлението на автора е, че рефакторингът става една все по-актуална тема в областта на софтуерните технологии. Тенденцията е той да става част от процесите за разработка на софтуер, поради налагащия се бърз развой и често променящите се изисквания на клиентите. Това налага разработка на средства, подпомагащи процесите за рефакторинг, а също така и все по-голямо внимание на тази тематика в обучението във висшите училища.

Таблица 1: *Връзка на приносите с основните цели и публикации*

Приноси	Цели	Задачи	Секция в дисертацията	Публикации от приложение 1
1. ***Предложение за процес за рефакторинг .***	1.2: Дефиниране на примерен процес за рефакторинг, базиран на опита получен при рефакторинг на системата XCTL.	1.2.1:Дефиниране на примерен процес за рефакторинг, базиран на опита получен при рефакторинг на системата XCTL	Глава III	5, 6
2. Разработване на прототипни средства за електронно обучение.	2.1: Създаване на концепция и разработване на агент за рефакторинг (Refactoring Agent)	2.1.1: Създаване на архитектура на агента за рефакторинг . 2.1.2: Разработване на агент за рефакторинг (Refactoring Agent)	Глава IV, параграф 4.1	1, 2, 3, 4
	2.2: Разработване на прототип на система за управление и навигация на процеси за рефакторинг	2.2.1: Проектиране на прототипа 2.2.2: Разработване на прототип на система за управление и навигация на процеси за рефакторинг	Глава III параграф 3.4	3, 5, 6
	2.3: Разработване на прототипна система eLSEBuilder за създаване на интерактивни и динамични презентации на софтуерни процеси.	2.3.1:Разработване на проект на прототипа на eLSEBuilder. 2.3.2:Разработване на прототипна система eLSEBuilder за създаване на интерактивни и динамични презентации на софтуерни процеси	Глава IV, параграф 4.2	3

Перспективи

В бъдеще се планира да се реализира идеята за специализиран възел към DeLC по Софтуерни технологии (eLSE) като:

- интегрираните средства в него се обогатяват с нови функционалности:
    - базата знания на RA ще бъде разширявана с нови методи за рефакторинг. За сега са изследвани 32 метода, които могат да бъдат имплементирани в настоящата архитектура на rLE. Продължава изследването на останалите около 70 метода описани в [8], за които се търси начин за реализация;
    - ще използваме средата S-Bahn-Tool за създаване на терминологични речници по софтуерни технологии и на други езици. За сега сме създали българо-английски, но може да се създаде и българо-немски;
- Създаване на нови прототипи, които да подпомагат обучението по софтуерни технологии.

Апробация

Резултати получени в изследването са използвани в следните международни и национални проекти:

- "Software Engineering Education And Reverse Engineering" Проектът е реализиран по линия на Пакта за стабилност в Югоизточна Европа и подкрепен от Германската служба за академичен обмен – DAAD, 2001- досега;
- Изследователски проект към фонд "Научни изследвания" – Министерство на науката и образованието. Тема на проекта: "Инфраструктура и електронно съдържание за обучението по софтуерно инженерство.", 2005 г.;
- Изследователски проект към фонд "Научни изследвания" – Министерство на науката и образованието. Тема на проекта: "Дигитална библиотека – Софтуерно инженерство (SEDILA)", 2008 г.

Междинни резултати и отделни части от работата са представени в редица публикации на международни и национални конференции:

- Четвърта Национална конференция „Образованието в информационното общество", 26-27 Май, 2011, Пловдив;
- International Scientific Conference "Informatics in scientific knowledge", Varna Free University "Chernorizets Hrabar", 23-26 June, Varna, 2010;
- International Scientific Conference "Informatics in scientific knowledge", Varna Free University "Chernorizets Hrabar", 26-28 June, Varna, 2008;
- International Conference on Computer Systems and Technologies – CompSysTech, Technical University - Gabrovo, Bulgaria, 12-13 June 2008;
- Национална научна конференция "Информатиката в научното познание", Варненски Свободен Университет "Черноризец Храбър", 14-16 юни, 2004;
- Fourth International Conference for Informatics and Information Technology, Bitola, Macedonia, 11-14 Dec 2003.

Авторска справка за приносите в дисертационния труд

Основните приноси на дисертационния труд са два:

1. **Предложение за процес за рефакторинг** – предложеният процес е дефиниран на базата на опита получен с рефакторинг върху наследената система XCTL;

2. **Разработване на прототипни средства за електронно обучение –** разработени са три прототипа на софтуерни средства, подпомагащи обучението по софтуерни технологии и е направено предложение за интегрирането им в специализиран възел към DeLC:
    - Софтуерното приложение **STREPCILS** е разработено като помощно средство за изучаване и овладяване на предложения в дисертацията процес на рефакторинг;
    - За изграждането на средата **rLE** е разработена агентно-ориентирана архитектура. Предложена е формализация на методите за рефакторинг като правила. Реализирана е обектно-ориентирана интерпретация на тези правила. Предложено е подходящо междинно представяне на анализирания първичен код и единна трансформационна схема, реализирана от агента;
    - **eLSEBuilder** подпомага създаването на симулации и визуализации на динамични процеси. Разработен като средство за подпомагане на обучението по софтуерни технологии той би могъл да се адаптира за приложение в други области. Използваните за представяне и визуализация графични символи се групират в „символни набори" и се съхраняват в специализирана база данни. При създаване на желаната симулация или визуализация потребителя може да избере желан набор.

Благодарности

Изказвам сърдечна благодарност на проф. д-р Станимир Стоянов – ръководител на катедра „Компютърни системи", на научния си ръководител доц. д-р Минчо Сандалски, на ръководството на ФМИ на ПУ в лицето на проф. д-р Асен Рахнев и на всички колеги за съветите, препоръките и съдействието.

Публикации по дисертационния труд

[1] Stoyanov S., A. Stoyanova-Doycheva, I. Popchev, M. Sandalski, *ReLE - A Refactoring Supporting Tool*, Compt. rend. Acad. bulg. Sci., 2011, Vol. 64, No. 7, pp. 1017-1026, ISSN 1310-1331, IF 2010=0.219

[2] Sandalski M., A. Stoyanova-Doycheva, I. Popchev, S. Stoyanov, *Development of Refactoring Learning Environment*, Cybernetics and Information Technologies, Vol. 11, No. 2., 2011, Bulgarian Academy of Sciences, ISSN: 1311-9702 (приета за печат)

[3] Сандалски М., А. Стоянова-Дойчева, *Среда за обучение по софтуерни технологии*, Четвърта Национална конференция „Образованието в информационното общество“, 26-27 Май, 2011, Пловдив, стр. 64-72, ISSN 1314-0752

[4] Glushkova T., A. Stoyanova, *Interaction and Adaptation to the specificity of the subject domains in the systems for e-learning and distance training DeLC*, International Scientific Conference “Informatics in scientific knowledge”, Varna Free University “Chernorizets Hrabar”, 26-28 June, 2008, pp. 295-306, ISSN 1313-4345, ISBN 10:954-715-303-X, ISBN 13:978-954-715-303-5

[5] Стоянова-Дойчева А., *Автоматизирано средство за навигация на процеси за рефакторинг*, Национална научна конференция “Информатиката в научното познание”, Варненски Свободен Университет “Черноризец Храбър”, 14-16 юни, 2004, стр. 138-145, ISBN 954-715-265-3.

[6] Stoyanova-Doycheva A., *A Software Refactoring Process and the Supported Tools*, Fourth International Conference for Informatics and Information Technology, Bitola, Macedonia, 11-14 Dec 2003, pp. 70-77, ISBN 9989-668-45-0

Доклади на работни срещи

[1] Сандалски М., А. Стоянова-Дойчева, *Среда за обучение по софтуерни технологии*, Четвърта Национална конференция „Образованието в информационното общество“, 26-27 Май, 2011, Пловдив

[2] Stoyanova-Doycheva A., *Development of Refactoring Learning Environment*, Research and Education in Mathematics, Informatics and their Applications, 10-12 December, 2010, Plovdiv, Bulgaria

[3] Stoyanova-Doycheva A., *eLearning Environment for Software Engineering Education (Refactoring Agent)*, 10th International Workshop “Software Engineering Education and Reverse Engineering”September 6 - 11, 2010,Ivanica, Republic of Serbia, http://www2.informatik.hu-berlin.de/swt/intkoop/daad/

[4] Stoyanova-Doycheva A., *Software Engineering eDictionary - our expirience with S-Bhan Tool*, 10th International Workshop “Software Engineering Education and Reverse Engineering” September 6 - 11, 2010, Ivanica, Republic of Serbia, http://www2.informatik.hu-berlin.de/swt/intkoop/daad/

[5] Stoyanova A., T. Glushkova, *Model for implementation of interactive distance learning*”, International Scientific Conference “Informatics in scientific knowledge”, Varna Free University “Chernorizets Hrabar”, 23-26 June, 2010

[6] Stoyanov S., A. Stoyanova-Doycheva, Michael Chigrichenko, *Improving the creativity in software engineering education with Refactoring Learning Environment*, Second International Conference Creativity and Innovation in Software engineering, Ravda, Bulgaria 10-12 June 2009

[7] Stoyanova-Doycheva A., *RFAgent – an eLearning Supporting Tool*, 8th International Workshop "Software Engineering Education and Reverse Engineering", Durres, Albania, 8-13 September, 2008, http://www2.informatik.hu-berlin.de/swt/intkoop/daad/

[8] Stoyanova-Doycheva A., *eLSEBuilder*, 7th International Workshop "Software Engineering Education and Reverse Engineering", Rissan, Monte Negro, 9-14 September, 2007, http://www2.informatik.hu-berlin.de/swt/intkoop/daad/

[9] Stoyanova-Doycheva A., *eLSEBuilder Framework*, 6th International Workshop "Software Engineering Education and Reverse Engineering", Ravda, Bulgaria, 18-23 September, 2006, http://www2.informatik.hu-berlin.de/swt/intkoop/daad/

[10] Stoyanova-Doycheva A., *A Refactoring Process based on the experience with XCTL*, 4th International Workshop "Software Engineering Education and Reverse Engineering", Zagreb, Croatia, 6-12 September, 2004, http://www2.informatik.hu-berlin.de/swt/intkoop/daad/

[11] Stojanov S., A. Stoyanova-Doycheva, M. Trendafilova, *From course materials to a textbook*, 4th International Workshop "Software Engineering Education and Reverse Engineering", Zagreb, Croatia, 6-12 September, 2004, http://www2.informatik.hu-berlin.de/swt/intkoop/daad/

[12] Стоянова-Дойчева А., *Автоматизирано средство за навигация на процеси за рефакторинг*, Национална научна конференция "Информатиката в научното познание", Варненски Свободен Университет "Черноризец Храбър", 14-16 юни, 2004

[13] Stoyanova-Doycheva A., *A Software Refactoring Process and the Supported Tools*, Fourth International Conference for Informatics and Information Technology, Bitola, Macedonia, 11-14 Dec 2003

[14] Stoyanova-Doycheva A., B. Botev, R. Gospodinov, *Description of changes – Deffractometrie/Reflectometrie – use case AreaScan*, 3rd International Workshop "Software Engineering Education and Reverse Engineering", Ohrid, Macedonia, 25-30 August, 2003, http://www2.informatik.hu-berlin.de/swt/intkoop/daad/

[15] Stoyanova-Doycheva A., R. Gospodinov, B. Botev, *Description of refactorings made on use case AreaScan, metrics, research about automated refactoring tools and future plans*, 3rd International Workshop "Software Engineering Education and Reverse Engineering", Ohrid, Macedonia, 25-30 August, 2003, http://www2.informatik.hu-berlin.de/swt/intkoop/daad/

[16] Stoyanova-Doycheva A., *Basic concepts for the scenario-based view*, 3rd International Workshop "Software Engineering Education and Reverse Engineering", Ohrid, Macedonia, 25-30 August, 2003, http://www2.informatik.hu-berlin.de/swt/intkoop/daad/

[17] Stoyanova-Doycheva A., *Basic concepts for the state-oriented view*, 2nd International Workshop "Software Engineering Education and Reverse Engineering", University of Plovdiv, Bulgaria, 15-21 September, 2002, http://www2.informatik.hu-berlin.de/swt/intkoop/daad/

[18] Стоянов С., М. Трендафилова, А. Стоянова-Дойчева, *Функционален модел на услугите предлагани от виртуалния университет*, Национална научна конференция "Информатиката в научното познание", Варненски Свободен Университет "Черноризец Храбър", 13-15 юни, 2002

[19] Stoyanova A., *Distance software development - an experience report*, 1st International Workshop "Software Engineering Education And Reverse Engineering", University of Novi Sad, Yugoslavia, 24-29 September, 2001, http://www2.informatik.hu-berlin.de/swt/intkoop/daad/

Библиография

[1] Course in Software Engineering. [Online]. http://www2.informatik.hu-berlin.de/swt/intkoop/jcse/

[2] Tom Mens and Serge Demeyer, *Software evolution*. Berlin Heidelberg: Springer-Verlag, 2008.

[3] D. Coleman, D. Ash, B. Lowther, and P. Oman, "Using metrics to evaluate software system maintainability," *IEEE Computer*, no. 27 (8), pp. 44-49, 1994.

[4] T. Guimaraes, "Managing application program maintenance expenditure," *Comm. ACM*, no. 26 (10), pp. 739-746, 1983.

[5] B. P. Lientz and E. B. Swanson, *Software maintenance management: a study of the maintenance of computer application software in 487 data processing organizations*.: Addison-Wesley, 1980.

[6] L. Tahvildari and K. Kontogiannis, "A methodology for developing transformations using the maintainability soft-goal graph," in *Working Conf. Reverse Engineering*, 2002, pp. 77–86, IEEE Computer Society.

[7] E. J. Chikofsky and J. H. Cross, "Reverse engineering and design recovery: A taxonomy," *IEEE Software*, vol. 7, no. 1, pp. 13-17, 1990.

[8] M. Fowler, *Refactoring: Improving the Design of Existing Programs*.: Addison-Wesley, 1999.

[9] J. Grundy, J. Hosking, and W. Mugridge, "Inconsistency management for multiple-view software development environments," *Trans. Software Engineering*, vol. 24, no. 11, pp. 960-981, 1998.

[10] W. G. Griswold and D. Notkin, "Automated assistance for program restructuring," *Trans. Software Engineering and Methodology*, vol. 2, no. 3, pp. 228–269, July 1993.

[11] T. Mens and A. Van Deursen, *Refactoring: Emerging Trends and Open Problems*., 2003.

[12] M. Sandalski, "Untersuchungen zur Nutzbarmachung des MS-ALGOL-Dialogcompilers für die Dialog-BESM-ALGOL-Programmierung," Humboldt-Universität zu Berlin, Dissertation 1979.

[13] W. F. Opdyke, "Refactoring: A Program Restructuring Aid in Designing Object-Oriented Application Frameworks," University of Illinois at Urbana-Champaign, Ph.D. 1992.

[14] (2011) Eclipse. [Online]. www.eclipse.org

[15] (2011) IntelliJ IDEA. [Online]. http://www.jetbrains.com/idea/

[16] NetBeans. [Online]. http://netbeans.org/

[17] FindBugs™ - Find Bugs in Java Programs. [Online]. http://findbugs.sourceforge.net/

[18] Checkstyle. [Online]. http://checkstyle.sourceforge.net/

[19] PMD. [Online]. http://pmd.sourceforge.net/

[20] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable ObjectOriented Software*. Reading, Massachusetts: Addison-Wesley, 1995.

[21] JUnit. [Online]. http://www.junit.org

[22] CUnit. [Online]. http://cunit.sourceforge.net/

[23] JMetric. [Online]. http://www.jmetric.com

[24] Understand C++. [Online]. www.scitools.com

[25] (2002) jFactor. [Online]. www.instantiations.com/jfactor/

[26] Документация на системата XCTL. [Online]. https://www2.informatik.hu-berlin.de/swt/projekt98/index.html

[27] Stoyanova-Doycheva, A., "A Software Refactoring Process and the Supported Tools," in *Fourth International Conference for Informatics and Information Technology*, Bitola, Macedonia, 11-14 Dec 2003, pp. 70-77, ISBN 9989-668-45-0.

[28] S. Stoyanov, A. Stoyanova-Doycheva, I. Popchev, and M. Sandalski, "ReLE - A Refactoring

Supporting Tool," *Comptes rendus de l'Acade'mie bulgare des Sciences*, vol. 64, no. 7, pp. 1017-1026.

[29] M. Sandalski, A. Stoyanova-Doycheva, I. Popchev, and S. Stoyanov, "Development of Refactoring Learning Environment," *Cybernetics and Information Technologies (CIT)*, vol. 11, no. 2, 2011, (to print).

[30] JADE (Java Agent DEvelopment Framework). [Online]. http://jade.tilab.com/

[31] IEEE Foundation for Intelligent Physical Agents. [Online]. www.fipa.org

[32] M. Hristova, S. Stoyanov, and A. Stoyanova-Doycheva, "Application of platform for electronic education on securing systems in railroad transport," in *International Conference on Computer Systems and Technologies – CompSysTech*, Gabrovo, Bulgaria, June 208, Technical University Gabrovo.

[33] S. Stoyanov, I. Ganchev, I. Popchev, and M. O'Droma, "From CBT to e-Learning," *J. Information Technologies and Control*, vol. III, no. 4, pp. 2-10, 2005.

[34] S. Stoyanov and I. Popchev, "Evolutionary Development of an Infrastructure Supporting the Transition from CBT to e-Learning," *Cyberbetics and Information Technologies (CIT)*, vol. 2, pp. 101-114, 2006, Bulgarian Academy of Sciences.

[35] Stoyanov, S., "Context-Aware and Adaptable Architecture CA3," in *International Conference REMIA 2010*, 2010, pp. 97-104.

[36] С. Стоянов, М. Сандалски, В. Вълканова, А. Стоянова-Дойчева, and Е. Дойчев, "Среда за доставка на електронни образователни услуги," in *Международна конференция „Електронно, дистанционно. или обучението на 21-ви век"*, София, 2011, pp. 207-214.

[37] Stoyanov, S.; Ganchev, I.; O'Droma, M.; Zedan, H.; Meere, D.; Valkanova, V., "Semantic Multi-Agent mLearning System," in *Semantic Agent Systems: Foundations and Applications*, A. Elci, M. T. Kone, and M. A. Orgun, Eds.: Springer Verlag, 2011, vol. 344.

[38] S. Stoyanov, I. Ganchev, D. Mitev, V. Valkanov, and M. O'Droma, "Service-oriented and Agent-based Architecture Supporting Adaptable, Scenario-based and Context-aware Provision of Mobile e-Learning Services," *International Journal of Computer Information Systems and Industrial Management Applications*, vol. 3, pp. 771-779, 2011.

[39] S. Stoyanov, I. Ganchev, I.Popchev, M.O'Droma, and V. Valkanova, "Agent-Oriented Middleware for InfoStation-based mLearning Intelligent Systems," in *5th IEEE International Conference on Intelligent Systems IS'10*, London, 07.07 – 09.07.2010, pp. 91-95.

[40] I.Ganchev, S. Stoyanov, V. Valkanova, and M. O'Droma, "Service-oriented and Agent-based Architecture Supporting Adaptable Context-Aware Provision of Mobile e-Learning Services," in *IADIS e-Learning 2010 Conference*, Freiburg, Germany, 26 - 29 July 2010, pp. 97-104.

[41] Stoyanov, S.; Popchev, I.; Doychev, E.; Mitev, D.; Valkanov, V.; Stoyanova-Doycheva, A.; Valkanova, V.; Minov, I., "DeLC Educational Portal," *Cybernetics and Information Technologies (CIT)*, vol. 10, no. 3, pp. 49-69, 2010.

[42] С. Стоянов, И. Попчев, Е. Дойчев, Д. Митев, and Г. Чолаков, "Архитектура на образователния портал DeLC," in *4-та Национална конференция „Образованието в информационното общество"*, Пловдив, 26-27 май, pp. 129-138.

[43] С. Стоянов, М. Сандалски, И. Попчев, Г. Чолаков, and Е. Дойчев, "Персонализирана и проактивна доставка на електронни услуги в образователния портал на DeLC," in *4-та Национална конференция „Образованието в информационното общество"*, Пловдив, 26-27 май, pp. 119-128.

[44] М. Сандалски and А. Стоянова-Дойчева, "Среда за обучение по софтуерни технологии," in *Четвърта Национална конференция „Образованието в информационното общество"*,

Пловдив, 26-27 Май, 2011, pp. 64-72.

[45] Software Engineering Education and Reverse Engineering (SEERE). [Online]. http://www2.informatik.hu-berlin.de/swt/intkoop/daad

[46] Стоянов, С.; Стоянова-Дойчева, А.; Трендафилова, М.; Дойчев, Е., *Софтуерни технологии*. Пловдив: Пловдивско университетско издателство, 2006.

[47] S. Demeyer, S. Ducasse, and O. Nierstrasz, *Object-Oriented Reengineering Patterns*.: Morgan Kaufmann and DPunkt, 2002.

[48] C. Atkinson and T. Kühne, "The role of meta-modeling in MDA," in *UML 2002 Workshop on Software Model Engineering*, Dresden, Germany, October 2002, pp. 67–70.

[49] K. Beck, *Extreme Programming Explained: Embrace Change*.: Addison Wesley, 2000.

[50] J. Pipka, "Refactoring in a "Test First"-World," in *Proc. 3rd Int'l Conf. eXtreme Programming and Flexible Processes in Software Engineering*, 2002.

[51] A. van Deursen and L. Moonen, "The video store revisited – thoughts on refactoring and testing," in *rd Int'l Conf. eXtreme Programming and Flexible Processes in Software Engineering*, 2002, pp. 71-76.

[52] A. Van Deursen, L. Moonen, A. Van den Bergh, and G. Kok, "Refactoring test code," in *Extreme Programming Perspectives*, 2001st ed.: Addison-Wesley, 2002, ch. 14, pp. 92-95.

[53] T. Mens, "A Formal Foundation for Object-Oriented Software Evolution," Department of Computer Science, Vrije Universiteit Brussel, Belgium, Ph.D. thesis 1999.

[54] T. Mens and T. Tourwé, "A declarative evolution framework for object-oriented design patterns," in *Int'l Conf. Software Maintenance*, 2001, pp. 570–579, IEEE Computer Society.

[55] "UML Specification 1.4.2," ISO/IEC 19501:2005,.

[56] T. Glushkova and A. Stoyanova, "Interaction and adaptation to the specificity of the subject domains in the system for e-Learning and distance training DeLC," in *International Scientific Conference "Informatics in scientific knowledge"*, June 2008, Varna Free University "Chernorizets Hrabar".

[57] A. Stoyanova-Doycheva, "A Software Refactoring Process and the Supported Tools," in *Fourth International Conference for Informatics and Information Technology*, Bitola, Macedonia, December 2003.

[58] А. Стоянова-Дойчева, "Автоматизирано средство за навигация на процеси за refactoring," in *Национална научна конференция "Информатиката в научното познание"*, юни 2004, Варненски Свободен Университет "Черноризец Храбър".

[59] Roger Pressman, *Software Engineering. A practitioner's Approach European Adaptation*.: McGraw-Hill Publishing Company, 2000.

[60] Fowler, M. Refactoring Web Site. [Online]. www.refactoring.com