ЕЛЕКТРОННО СПИСАНИЕ „ИКОНОМИКА И КОМПЮТЪРНИ НАУКИ“, БРОЙ 2, 2019,
ISSN 2367-7791, ВАРНА, БЪЛГАРИЯ
ELECTRONIC JOURNAL “ECONOMICS AND COMPUTER SCIENCE”, ISSUE 2, 2019,
ISSN 2367-7791, VARNA, BULGARIA

Техническа изпълнимост на софтуерен проект за виртуализация на задачи изпълнявани от студентите в курса на висшето образование - проблеми и решения при еволюционното развитие

Иван КУЮМДЖИЕВ[1]

[1] Икономически университет - Варна
ivan_ognyanov@ue-varna.bg

Резюме. Динамиката в изискванията на бизнеса налага модернизиране на методите на преподаване в училищата и университетите. Целта на публикацията е да бъдат изследвани технологичните особености при създаване и поддържане на сложна софтуерна система за виртуално обучение. За да се оцени техническата изпълнимост е предложен класификатор на потенциални промени в софтуера, като е взето предвид влиянието, което те оказват върху действията извършвани от администраторите на бази от данни и програмистите. Статията предлага различни подходи за управление на промените, както на ниво програмен код, така и на ниво бази от данни. Анализирани са ограниченията наложени от използваните технологии и на тази база са очертани са най-добрите методи за управление на промени в софтуерната система за виртуално обучение. Резултатите от изследването могат да бъдат полезни за проектанти на системи и администратори на бази от данни.

Ключови думи: рефакторинг на програмен код, системи за управление на бази от данни, нерелационни бази от данни, софтуерна поддръжка.

1. Въведение

Нуждите на бизнеса свързани с познанията на търсените кадри, следва в една или друга степен да бъдат зачетени и удовлетворени от висшите учебни заведения. В тази връзка се прилагат различни практики за осъвременяване на подходите за обучение, които имат за цел студентите да бъдат поставени в условия максимално близки до работна среда (Dimitrov, 2018). Като пример за такъв подход може да бъде посочено използването на интерактивна уеб среда за изграждане на приложни знания и умения. Подобно на професии изискващи работа със скъпа апаратура при която цената на грешката е висока (пилоти на самолети, капитани на кораби и т.н.) студентите ще могат да използват симулатор, който да ги доближи до предизвикателствата на реалния свят. Като потребители на такава система могат да бъдат идентифицирани студенти, работодатели, експерти, преподаватели и др., а поръчител за разработване различни образователни институции.[19]

Публикацията има за цел да изследва техническата изпълнимост на софтуерен проект насочен към изграждането на бизнес симулатор (БС) за целите на висшето образование, като акцентира на предизвикателствата пред развитие на използваната база от данни.

Практически разработки на проекти с размерите на БС би трябвало да бъдат съпроводени с редица проучвателни дейности, гарантиращи постигане на етап в който софтуерът ще отговаря на нуждите на потребителите. Нуждата от извършването на такива дейности е изследвана в редица публикации насочени както към публичния (Todoranova, 2014), (Raychev, 2018) така и в частни сектор (Nacheva, 2017)(Stoyanova, 2015)(Timofeeva, 2018). Оценката на техническа изпълнимост на проекта е една от тези дейности. Тя се фокусира върху изследване на наличните технически ресурси на организацията разработчик и тяхната приложимост към функционалните изисквания на системата (O’Brien, J., Marakas, 2011). Тази дефиниция ще бъде перифразирана за нуждите на проекта като „избор на подходящи технологии, които могат да бъдат използвани, така че да се удовлетворят изискванията към разработваната система.“

[19] Пълното описание на проекта на системата е резултат от изследванията на други автори, които ще бъдат публикувани допълнително.

В класическата представа за жизнен цикъл на една информационна система (Benington, 1983) е прието, че след като бъдат събрани и подробно описани нуждите към системата се създава проект на базата от данни. След това програмистите създават кода на приложението, като се съобразяват с вече дефинираната структура на базата от данни. Според същото това класическо разбиране, потребителите не трябва да променят изискванията си, защото това би довело до промяна в структурата на базата данни и съответно до нужда от промяна в програмния код. За разрешаване на тези проблеми, през последните десетилетия, са въведени гъвкави методологии за разработка на софтуер (Larman, 2003), при които се приема още в началния етап, че системата следва да се развива и промените в базата от данни и кода на приложението са неизбежни и дори желателни.

Въпреки развитието на теорията по отношение на гъвкавите методологии за разработка на софтуер, считаме за наложително да бъдат разгледани специфичните проблеми пред разработката и развитието на БС, както и възможните средства за преодоляването им.

2. Анализ на възможните технически проблеми при разработката на БС, свързани с промяна на потребителските изисквания

Както беше посочено, проектирането на базата от данни е един от факторите, от които зависи програмния код на продукта. Следователно всяка промяна в изискванията на потребителите, която изисква промяна на структурата на базата от данни ще доведе до необходимост от корекции на поне две места – самата база от данни и кода на приложението. Ако приемем, че проектът на БС отразява началните нужди на посочените потребители, то е логично да предположим, че изискванията им ще се променят с развитието на реалния свят, което ще наложи и еволюция на симулатора. По тази причина считаме за наложително да бъдат изследвани кои са най-вероятните промени в БС, до какви проблеми за програмистите и проектантите на бази от данни могат да доведат и как тези проблеми, могат да бъдат смекчени или изцяло решени, чрез различни технологии.

За да бъдат категоризирани промените в базата от данни и произхождащите от тях дейности, ще разгледаме архитектурата ANSI/SPARC (фиг. 1). В нея са отделени три нива – външно, концептуално (логическо) и физическо. Представата на всички потребители за системата е обединена в концептуален (и или логически) модел на базата от данни, който след това се преработва във физически в зависимост от използваната система за управление на бази от данни (ANSI/X3/SPARC, 1975).

Фигура 1. ANSI/SPARC архитектура

Дейностите по промяна на логическия и физическите модели се извършват най-често от администраторите на бази от данни, а програмистите взаимодействат единствено с логическия. По тази причина, промяната на изискването дори на един потребител на системата, може доведе до необходимост от корекции от поне два вида специалисти.

Първоначално ще разделим възможните промени на два вида:

1) такива, които налагат промяна във физическия модел на базата от данни.

2) такива, които налагат промяна в логическия модел на базата от данни

Проблеми при реорганизацията на физическия модел на данните - физическият модел на базата от данни включва физическите структури за съхраняване на данните (файлове, страници, записи),

допълнителни структури, осигуряващи подобряване скоростта на достъп до данните (индекси), месторазположението и размера на необходимата памет за съхраняване на базата от данни и други физически характеристики, изисквани от избраната СУБД. Изпълняването на тази задача е отговорност на администраторите на бази от данни. Благодарение на т.нар. „физическа независимост“, залегнала в основата на ANSI/SPARC, промените във физическия слой не се отразяват на логическия и следователно програмистите няма да бъдат ангажирани с нанасяне на корекции в кода. По-тази причина можем да определим като по-ниско нивото на влияние на тези промени върху времето за разработка на системата.

Като се вземе предвид описанието на изискванията към БС и характеристиките на физическия модел, можем да идентифицираме следните области като източници на необходимост от промени в модела:

1) бавен достъп до данните,

2) увеличаване на обема на съхраняваните данни.

Двете области много често са свързани, защото при увеличаване на обема на данните в релационна база от данни, се забавя достъпа до тях. В такива случаи се налага администраторите на бази от данни да предприемат мерки за по-ефективно използване на наличните ресурси, чрез настройки на използваната СУБД, а ако това не може да де нужния резултат се налага закупуване на нови ресурси. Като потенциален източник на такава необходимост можем да посочим:

1) Увеличаване на броя на потребителите на системата, така че данните, които те генерират да надхвърлят изчислителните възможностите на използвания хардуер.

2) Увеличаване обема на съхраняваните данни – при запис на мултимедия и/или документи, се заема по-голямо пространство на твърдите дискове, в сравнение с запис на обикновен текст.

За справяне с първия проблем са възможни два подхода – закупуване на по-мощни ресурси или използване на СУБД, която да позволява разпределение на данните и натоварването на различни сървъри. Считаме, че изборът на решение би трябвало да бъде насочен към подхода на разпределение на натоварването, защото той предоставя възможност за почти неограничено нарастване на броя на потребителите и входно изходните операции. Въпреки, че голяма част от най-популярните СУБД предоставят такива механизми (MS SQL Server, Oracle и т.н.) трябва да се има предвид, че те са достъпни само в комерсиалните версии на продуктите. Пример за широко използвани СУБД с лиценз позволяващ безплатно използване, които позволяват разпределение на натоварването са MongoDB и CouchDB.

За да се избегне проблем със съхраняването на големи обеми от данни е възможно да се приложи предложения по-горе подход. Въпреки това, с навлизането на техника с възможности за запис на мултимедия с високо качество, е възможно дори единствен потребител да генерира гигабайти данни за кратък период от време. По тази причина би следвало системата да съхранява мултимедийни файлове само в случай, че очакваният им обем не надхвърля възможностите на използвания софтуер и хардуер.

Проблеми при реорганизацията на логическия модел на данните - ако приемем, че в даден момент от време БС изпълнява посочените в проекта функции, то последващи промени в логическия модел на данните, биха произтекли от промяна на потребителските изисквания към системата и по-конкретно към данните, които трябва да се съхраняват и извличат от нея. Според Macek и Richta (Macek and Richta, 2014) трудността на процесите на еволюция на информационната система са свързани, както с броя на промените, така и с броя на засегнатите софтуерни компоненти. Вече беше посочено, че тези промени засягат както администраторите на бази от данни, така и програмистите (към засегнатите от тези промени могат да бъдат включени и всички останали участници в процеса на разработка – тестери, бизнес анализатори, специалисти по внедряване и документация и т.н.). Това води увеличаване обема на дейностите, които да се извършат и съответно времето за разработка и стойността на продукта. По тази причина са правени опити да бъде решен този проблем от различни автори.

Рефакторингът (преработка на код) е много популярна практика за обектноориентираната концепция, насочена към еволюционно развитие на кода и софтуерната архитектура. Той се дефинира от (Beck *et al.*, 1999) като промяна на структурата на програмния код, която не се отразява на неговото поведение от гледна точка на потребителя. Пречупването на рефакторингът през призмата на базите от данни обаче изисква по-различен подход, тъй като те отразяват представите на потребителите, промените в които, както беше посочено налагат необходимостта от преструктуриране.

Може да се твърди, че сложността на промяна на логическия модел на данните зависи до голяма степен от набора от правила на които той се подчинява. Според някои автори (Eckstein and Schultz, 2017) релационните бази от данни са една от най-устойчивите и използвани концепции в информационните системи за последните десетилетия. Те осигуряват усъвършенствани функции за запис и извличане на данните, които се съхраняват в релационна схема. При релационните бази от данни се прилага процес наречен нормализация, който се дефинира като разпределяне на данните в релационните схеми, при спазване на набор от правила (Purba, 1999). Именно тези правила гарантират, че данните ще бъдат

непротиворечиви и ще се съхраняват, така че да удовлетворят нуждите на потребителите. Ето защо може да се твърди, че поддържането на базата от данни в състояние, което правилно да отразява процесите от реалния свят се решава, чрез правилата за изграждане на схема на релационни бази от данни.

Според някои автори (Ambler and Sadalage, 2006), рефакторинг е малка промяна на схемата (структурата) на базата от данни, която я подобрява, като същевременно запазва информационната същност и поведението й. В това определение остава неясно какво се разбира под „малка промяна“ и по тази причина предпочитаме да използваме дефиниция, която описва по-точно целия процес – техниката за рефакторинг на бази от данни се свежда до приложение на контролирани операции за промяна на съществуващата схема (D’Sousa and Bhatia, 2009).

Промените в работещи софтуерни проекти се извършват на три етапа (Martin Lippert, 2006), като първите два от тях можем да причислим към процеса на рефакторинг на бази от данни:

1. Промяна на модела на данните (схемата).
2. Миграция на данните между стария и новия модел.
3. Рефакторинг на кода, който достъпва засегнатите части на базата от данни.

В зависимост от промените се определя вида на рефакторинг на базите от данни и произтичащите от него действия. За да оценим по-точно възможните промени в изискванията на потребителите на БС и произтичащите от тях усложнения, ще разгледаме и класификация на видовете рефакторинг на бази от данни. Според някои автори (Ambler and Sadalage, 2006) те са:

1) Структурен – промяна на дефиницията на един или повече обекти от схемата.

2) Насочен към подобряване на качеството на данните – налагане на ограничения на приеманите стойности.

3) Насочен към налагане на ограничението „цялост на връзките между таблиците“.

4) Архитектурен – подобряване на начина по който външните програми взаимодействат с базата данни.

5) Насочен към методите – промяна на съхранени процедури, тригери и функции.

6) Трансформации – промяна на схемата, която води до промяна на нейната логика.

Като положителна черта на тази класификация можем да посочим, разделянето на ясни и управляеми проблеми от гледна точка на администраторите на бази от данни. Въпреки това, тя не отразява връзката им с промяна изискванията на потребителите, както и обема на нужните за извършване промени от гледна точка на програмистите.

По тази причина предлагаме пет сценария, които да бъдат използвани за класификация на промените в изискванията на потребителите, като се взема предвид влиянието им върху администраторите на релационни бази от данни и програмистите:

1) Промяна на стратегията на бизнеса (цялостна промяна на концепцията на разработваната система) – в т.ч. функции, входни и изходни данни, потребители. Подобна кардинална промяна налага връщане в начален етап на разработка на приложението и по тази причина е крайно нежелателна от ангажираните с разработката на системата.

2) Промяна в начина на съхранение на съществуващи данни. Този казус може да бъде разделен на два отделни в зависимост дали се налага промяна в кода на приложението или не. Съответно това ще са случаите на промяна начина на съхранение без необходимост от корекция на методите на извличане на данни и вариант в който се налагат промени и в кода на приложението. Сложността на извършваните корекции ще зависи пряко от степента на съответствие между старата схема на данните и новата, която удовлетворява промяната в изискванията.

3) Добавяне на нови данни със специфични правила за тях. Тази операция изисква както промяна в схемата на базата от данни, така и промяна в кода на приложението. И на двете места би следвало да се отразят изискванията към стойностите, които могат да приемат данните. Разликата между този и предходния казус, е че тук данните за първи път постъпват в системата, а при горния вече са налични под някаква форма. По тази причина операциите тук са по-малко, защото изключват нуждата от миграция на данни от едни структури в други.

4) Промяна на изискванията за изхода от системата, при използване на съществуващите данни. Дори да налага програмиране на по-сложни изчисления, този проблем е по-лесен за разрешаване от предходните, защото не изисква промяна на базата от данни и корекциите ще се сведат само до пренаписване на програмен код за създаване на отчети/справки.

5) Добавяне на нови данни без специфични правила за тях. Следва да се тълкува като добавяне на характеристика към обект, който вече е описан в базата от данни, но тази характеристика не съдържа и ограничения към стойностите, които може да приема. Този вид промени изискват само добавяне на едно или няколко полета в съществуващи обекти от базата данни и програмен код за записването им там, но без необходимост от допълнителни обработки.

3. Анализ на възможните източници на промяна на изискванията в БС и оценка на сложността на разрешаването им

За разпределение на потенциалните области с необходимост от промяна ще бъде използвана посочената по-горе класификация. Считаме, че не е възможно разработката да попадне в сценарий 1, защото изискванията на системата са ясно дефинирани и частични нейни функционалности са прилагани от авторите на проекта, което е довело да пълно изясняване на изпълняваните от нея функции.

Като много висока ще определим вероятността да се стигне до сценарии 4 и 5. Причината се корени във факта, че системата ще работи с все по-голям кръг потребители. Въпреки, че основните видове потребители не би следвало да се променят, то с увеличаването на броя на потребителите и промяна на условията в реалния свят, е все по-вероятно да настъпи момент в който ще има потребител с необходимост от добавяне на нов вход или изход от системата.

Като две области с вероятност за навлизане в сценарии 2 и 3, можем да посочим функции свързани със съхраняване на автобиография в системата, както и изпълняване на мисии от обучаващите се. Съхраняването на автобиографията на студентите е типичен пример за възникване на необходимост от промяна на начина на съхранение на данните. Ако автобиографията се съхранява под формата на файл, това би улеснило до голяма степен разработчиците при запис на данните, но при необходимост от търсене на кандидати с точно определени характеристики би възникнал труден за решаване проблем. По правило съхраняването на данните в обобщен и неструктуриран формат, затруднява извличането на информация от тях. По тази причина е възможно в определен момент да се наложи промяна на структурата на базата от данни, така че да бъде възможно съхраняване на данните от автобиографията в отделни обекти от схемата на данните.

С голяма вероятност за реализиране на сценарий 3 е модулът, който ще се грижи за управление на мисиите, изпълнявани от студентите. Първоначално изпълняваните мисии ще бъдат надлежно описани, което позволява създаване на модел на данните и код за обработката им. При добавяне на мисии, които се подчиняват на текущо описаните правила разработваната система би могла да бъда настроена сравнително лесно. За сметка на това при добавяне на нови мисии с нереализирана до момента логика, ще бъде необходимо да бъде променена схемата на базата от данни, да се добавят нови данни, нови правила за работа с тях, както и нови начини за извличането им.

4. Възможни решения на посочените проблеми

Както беше посочено, еволюцията на информационните системи е очакван процес и за управлението й следва да бъдат приложени мерки още в самото начало на проекта. Според някои автори (Aboulsamh and Davies, 2010) проблемите на еволюцията на системата могат да бъдат решени, чрез използване на UML диаграми. Езикът е утвърдено средство за моделиране на системи, и съществуват софтуерни продукти[20], които предлагат възможност за генериране на код на база създадената диаграма. Въпреки това, не считаме за възможно, чрез наличния към момента софтуер и хардуер, промяна в UML диаграмата описваща една система да доведе автоматично до пресъздаване на целия софтуерен продукт.

В тази връзка подкрепяме тезата, че работата по промяна на структурата на релационна база от данни и съответните промени в кода не може да бъде напълно автоматизирана (Schink, 2013), но считаме, че може да бъде значително улеснена при употреба на определени технологии.

ORM работни рамки – съществуват множество инструменти за осъществяване на връзка между релационни бази от данни и обекти в програмния код, като някои от тях предлагат и инструменти за промяна на структурата на данните. При използване на език за разработка Java може да се вземе предвид продукта Hibernate, който има възможности за създаване на таблици и добавяне на колони в базата от данни при промяна в приложението.

Active Record е ORM рамка използвана в среди с език за програмиране Ruby on Rails, която дава възможност не само за промяна на структурата на базата от данни (наричани миграции[21]), а и добавяне на потребителски SQL команди.

Подобни инструменти са налични при използване на платформата .NET в лицето на Entity Framework, както и на разпространения за разработване на уеб приложения език PHP (Propel, Doctrine). Въпреки, че използването на ORM работни рамки подпомага процеса по рефакторинг, той не може да бъде изцяло автоматизиран от тях.

Инструменти за еволюция на бази от данни – освен на ниво програмен код, са налични инструменти за рефакторинг насочени само към базата от данни. Проектът MeDEA (Metamodel-based

[20] https://www-03.ibm.com/software/products/bg/enterprise
[21] Hansson, D.H., Kemper, J.: ActiveRecord::Migration (2009), http://api.rubyonrails.org/classes/ActiveRecord/Migration.html

Database Evolution Architecture) е стъпка към разрешаване на проблемите на еволюцията на базата от данни, като дефинира метамодел, чрез който по-лесно да се извършват и проследяват необходимите промени по базата от данни (Domínguez *et al.,* 2008). Въпреки това не е добил широко приложение и популярност, а е известен по-скоро в научните, отколкото в средите на разработчиците.

Друг проект добил по-широка популярност под формата на софтуерен инструмент е DB-MAIN (Hick and Hainaut, 2006). Той предоставя възможности за управление модела на данните и промяна на структурата му, както и експортиране на данните към различни формати и използване на UML диаграми[22]. Въпреки посочените предимства, трябва да се вземе предвид факта, че продуктът не е безплатен.

Продукт, който предоставя същите възможности за рефакторинг на базата от данни, има безплатна версия и добавя и автоматизирано създаване на документация за промените е Liquibase[23]. Подобно на посочените по-горе инструменти той е насочен само към релационни бази от данни, което го прави най-подходящ избор за целите на разработката при използване на този вид организация на данните.

Прилагането на посочените по-горе инструменти има потенциала да смекчи последиците от необходимост от промени в системата и да направи процеса на рефакторинг по-лесен. Въпреки това, не считаме, че тези инструменти могат да елиминират изцяло някой от посочените проблеми.

За да се избегне нуждата от промяна на структурата на базата от данни е възможно да се използва такава технология, в която няма предварително дефинирана схема.

Използване на СУБД в които няма предварително дефиниран модел на данните -развитието на изискванията за мащабируемост към информационните технологии през последното десетилетие, наложи появата на бази от данни с концепция различна от релационната. Те най-често се наричат NoSQL бази от данни и предоставят възможности за използване на по-гъвкава структура на данните, в сравнение с релационния модел. Като най-широко разпространени представители могат да бъдат посочени MongoDB, CouchDB, Bigtable и Dynamo.

В MongoDB например данните се съхраняват в структури наречени колекции, състоящи се от документи, към които няма изискване за предварително дефиниране на схема. Това означава, че ако до даден момент от време определен обект се описва с точно определени атрибути, то няма никакви пречки в следващия да бъде описан със съвсем различни характеристики, които да се запишат в базата от данни.

Като произтичащи ползи от използването на такъв вид бази от данни могат да бъдат посочени смекчаване на проблемите с мащабируемостта и промяна на логическата структура на данните. Промяната в структурата се отразява минимално на самата СУБД, защото тя няма схема и следователно не се налага администраторите да я променят и да реализират новите изисквания за ограничения към данните.

Посочените предимства биха улеснили нанасяне на промените в случаи при които се налага добавяне на нови мисии, нови характеристики на автобиография и т.н. Освен това повечето нерелационни СУБД предоставя механизми, чрез които да се разреши евентуален проблем с експоненциално увеличаване на потребителите и генерираната от тях информация, като се използват множество компютри.

Въпреки, че използването на нерелационни СУБД има неоспорими предимства, трябва да бъдат разгледани и недостатъците на тази концепция.

1) Липсата на предварително дефинирана структура усложнява извличане и обобщаване на данни в различни по вид отчети. Това означава, че сценарий 4 по задаване на нови изисквания към изхода на системата, може да се превърне в задача с повишена сложност.

2) Наложените от потребителите логически изисквания към данните трябва да бъдат реализирани в кода на приложението – докато при релационните схеми тези изисквания се налагат от системата за управление на бази от данни, то тук отговорността за това е изцяло в ръцете на програмистите. По тази причина според някои автори (Rocha *et al.,* 2015), NoSQL базите от данни нямат изцяло неопределена структура – структурата всъщност се прилага от самото приложение. Ето защо, дори когато се използва СУБД, която не задължава предварително дефиниране на структурата на базата от данни, се налага извършване на промени в приложението при рефакторинг на базата.

По тези причини считаме, че използването на нерелационни бази от данни може да реши някои от проблемите на еволюцията на системата, но не би следвало да бъде приеман като единствена възможност.

[22] http://www.rever.eu/en/technical-features
[23] http://www.liquibase.org/

5. Заключение

Направеното изследване показва, че създаването и развитието на софтуерен проект за виртуализация на задачи за изпълнение от студенти в курса на висшето образование е възможно. За тази цел обаче е нужно да се вземат предвид особеностите на софтуерното производство в комбинация със специфичните нужди на потребителите на системата.

Създаденият класификатор на промените в системата е приложен към възможните им източници. По този начин са очертани най-критичните области и са изведени препоръки за смекчаване негативните последици от промените. Някои от тях включват използване на СУБД, които позволяват разпределяне на натоварването и имат по-свободни изисквания към структурата на базата данни; при използване на релационна БД да се вземе предвид инструмента Liquibase; за улесняване на промените в частите от програмния код, които правят връзка с базата данни да се използва ORM работна рамка.

Literature

Aboulsamh, M. A. and Davies, J. (2010) ‘A metamodel-based approach to information systems evolution and data migration’, in *Proceedings - 5th International Conference on Software Engineering Advances, ICSEA 2010*. doi: 10.1109/ICSEA.2010.31.

Ambler, S. and Sadalage, P. (2006) ‘Refactoring databases: Evolutionary database design’, *Queue*.

ANSI/X3/SPARC (1975) ‘Interim Report: ANSI/X3/SPARC Study Group on Data Base Management Systems’, *FDT - Bulletin of ACM SIGMOD*.

Beck, K. *et al.* (1999) ‘Bad Smells in Code’, in *... Improving the design of existing code*.

Benington, H. D. (1983) ‘Production of Large Computer Programs’, *Annals of the History of Computing*. doi: 10.1109/MAHC.1983.10102.

D’Sousa, A. and Bhatia, S. (2009) ‘Refactoring of a database’, *International Journal of Computer Science and Information Security*.

Dimitrov, I. (2018) ‘A STATE OF ERP LEARNING IN ACCOUNTING EDUCATION IN BULGARIAN UNIVERSITIES’, *Списание „Диалог“*, 3, pp. 37–71.

Domínguez, E. *et al.* (2008) ‘MeDEA: A database evolution architecture with traceability’, *Data and Knowledge Engineering*. doi: 10.1016/j.datak.2007.12.001.

Eckstein, J. and Schultz, B. R. (2017) *Introductory Relational Database Design for Business, with Microsoft Access, Introductory Relational Database Design for Business, with Microsoft Access*. doi: 10.1002/9781119430087.

Hick, J. M. and Hainaut, J. L. (2006) ‘Database application evolution: A transformational approach’, *Data and Knowledge Engineering*. doi: 10.1016/j.datak.2005.10.003.

Larman, C. (2003) *Agile & Iterative Development, Addison-Wesley*.

Macek, O. and Richta, K. (2014) ‘Application and relational database co-refactoring’, *Computer Science and Information Systems*. doi: 10.2298/CSIS130610033M.

Martin Lippert, S. R. (2006) *Refactoring in Large Software Projects: Performing Complex Restructurings Successfully, Wiley*.

Nacheva, R. (2017) ‘Architecture of Web-Based System for Usability Evaluation of Mobile Applications’, *Izvestiya Journal of Varna University of Economics*, 61(2), pp. 187–201.

O’Brien, J., Marakas, G. (2011) *Developing Business/IT Solutions In Management Information Systems*. New York: McGraw-Hill/Irwin.

Purba, S. (1999) *Handbook of Data Management*. CRC Press.

Raychev, T. (2018) ‘Assessment of canceled and problematic concession projects in the water supply and sewerage sector worldwide (in Bulgarian)’, *Izvestia Journal of the Union of Scientists -Varna. Economic Sciences Series*, 7(3), pp. 184–195.

Rocha, Á. *et al.* (2015) ‘New contributions in information systems and technologies’, in *Advances in Intelligent Systems and Computing*. doi: 10.1007/978-3-319-16528-8.

Schink, H. (2013) ‘Sql-schema-comparer: Support of multi-language refactoring with relational databases’, in *IEEE 13th International Working Conference on Source Code Analysis and Manipulation, SCAM 2013*. doi: 10.1109/SCAM.2013.6648199.

Stoyanova, M. (2015) ‘Gamification process in information systems’, *Special Issue Proceedings of ETAEc 2015 Conference, Scientific Bulletin – Economic Sciences*, 14, pp. 174–180.

Timofeeva, A. (2018) ‘Information systems and databases in malls and opportunities for their improvement (in Bulgarian)’, *Izvestia Journal of the Union of Scientists - Varna*, 7(3), pp. 124–132.

Todoranova, L. (2014) ‘Implementation of Business Intelligent Systems in the Public Sector in Bulgaria (in Bulgarian)’, *Izvestia Journal of the Union of Scientists - Varna*, pp. 115–123.

ЕЛЕКТРОННО СПИСАНИЕ „ИКОНОМИКА И КОМПЮТЪРНИ НАУКИ“, БРОЙ 2, 2019,
ISSN 2367-7791, ВАРНА, БЪЛГАРИЯ
ELECTRONIC JOURNAL “ECONOMICS AND COMPUTER SCIENCE”, ISSUE 2, 2019,
ISSN 2367-7791, VARNA, BULGARIA

Technical feasibility of a software project for virtualization of tasks performed by students in the higher education course - problems and solutions in evolutionary development

Ivan KUYUMDZHIEV[1]

[1] University of Economics, Varna, Bulgaria
ivan_ognyanov@ue-varna.bg

Abstract. The dynamics of business requirements necessitates the modernization of teaching methods in schools and universities. The purpose of the publication is to explore the technological aspects of creating and maintaining a complex software system for virtual learning. In order to assess technical feasibility, a classifier of potential software changes is proposed, taking into account the impact they have on the database administrators and developers. The article offers different approaches to managing changes, both at the programming code level and at the database level. The limitations imposed by the technologies used are analyzed and, on this basis, the best methods for managing changes in the virtual learning software system are outlined. The results of the study can be useful for software system designers and database administrators.

Key words: code refactoring, database, database management systems, NoSQL, software changes.