Софийски университет “Св. Кл. Охридски”

Факултет по математика и информатика

Катедра Софтуерни технологии

Хабилитационен труд

на доц. д-р Силвия Христова Илиева

по конкурс за професор по
Професионално направление: 4.6 Информатика и компютърни науки,
Научна специалност: 01.01.12 Информатика
(софтуерни технологии – софтуерни процеси)

Март 2012, София

Съдържание

Списък на фигурите

Списък на таблиците

I. Увод

Софтуерната индустрия е вероятно най-големия сектор в света. Според презентацията на проф. Ескенази на 7.03.2012 г, базираща се на данни от www.gartner.com (Gartner, 2010), разходите за софтуер в световен мащаб през 2010 г. достигат 321 милиарда долара, като за сравнение през 2007 г са били 209 милиарда долара. В тези цифри не са включени неявните разходи за софтуер, който е вграден в редица устройства и системи (мобилни телефони, таблети, преносими компютри автомобилни системи, авиационни системи и др). Въпреки бурното развитие и тези успехи, софтуерните технологии (software engineering) като дял от знанието все още са изправени пред редица проблеми, подобни на тези при възникването им преди 50 години като например как да се разработи софтуер в срок, как да се разработи качествен софтуер без грешки при работата му, как да се разработи софтуер с ниска цена и др. За решаването на проблемите са съсредоточени усилията както на научната общност, така и на представителите на индустрията.

Текущият хабилитационен труд обобщава научно изследователската дейност и съществени резултати на доц. Силвия Илиева в две основни направления от софтуерните технологии – гъвкави подходи за разработване на софтуерни системи и тестване на софтуерни системи, като част от цялостния софтуерен процес. Гъвкавите подходи адресират проблема с разработването на софтуер в срок, като за целта са необходими дисциплинирани методи и практики. Тестването на софтуерните системи е един от подходите за осигуряване на качеството на софтуера. Хабилитационният труд е подготвен във връзка с участието на доц. Силвия Илиева в конкурс за професор по научно направление информатика и компютърни науки, научна специалност Информатика /софтуерни технологии – софтуерни процеси/, обявен в Държавен вестник бр. 5/17.01.2012 г. Целта на хабилитационния труд е да представи в синтезиран вид част от постиженията на автора, обединени в две тематични направления. Представеното в хабилитационния труд е в следствие от работата на автора през последните 10 години (след постъпване на работа във ФМИ) по национални и международни проекти, с колеги, докторанти и дипломанти. Софтуерните технологии като дял от знанието предполагат екипна работа за решаване на реални проблеми. Като резултат от екипната работа се предлагат работещи решения и добри практики.

Изложението на хабилитационният труд е структурирано съгласно изискванията на Правилник за условията и реда за придобиване на научни степени и за заемане на академични длъжности във ФМИ на СУ "Св. Кл. Охридски", Глава 6 Допълнителни разпоредби, алинея 5: *"Хабилитационен труд" е научно съчинение, съдържащо съществени и оригинални резултати, обединени от обща тема. Трудът следва да съдържа обзор на предметната област, мотивация за изследването, изложение на резултатите и съответните изводи.*

Хабилитационният труд съдържа 2 глави, засягащи двете основни направления - гъвкави подходи за разработване на софтуерни системи и тестване на софтуерни системи, следвани от заключение и литературна справка.

Глава 2 представя обзор на гъвкавите методологии за разработване на софтуерни системи, като са сравнени по различни критерии. Предложени са няколко нови гъвкави подходи, включително са идентифицирани метрики за оценката им и са описани резултати от прилагането им. Предложен са съответно и систематични подходи за избор и за внедряване на подходящи гъвкави методи и практики, в зависимост от

ситуационни фактори. В заключение са обобщени резултатите и са очертани насоки за бъдещо развитие.

Глава 3 представя обзор на тестването на софтуерни системи с фокус върху системите, ориентирани към услуги, следван от формулиране на проблеми в тази област, представляващи интерес за автора. Предложена е тестова среда/рамка и съответната методология за тестване. Представени со отделните софтуерни инструменти, които са интегриране в тестовата рамка. В заключение са изложени ползите и потенциалните потребители на предложените решения и са дадени насоки за бъдещо развитие.

В Заключението са обобщени съществените постижения на автора в двете основни направления - гъвкави подходи за разработване на софтуерни системи и тестване на софтуерни системи.

В секция Литература след всяка част са описани използваните литературни източници.

Секция VI предлага списък на съществените публикации на автора по темите, засегнати от хабилитационния труд.

За да се докаже влиянието на представените резултати върху развитието на областите е представена справка за известни цитирания на представения в предишната секция списък със съществени публикаци.

II. Гъвкави методологии

Настоящата глава е посветена на гъвкавите методологии за разработване на софтуерни системи. Авторът работи в това направление от 2002 г насам като през този период са публикувани 14 научни статии (9 от тях са представени за участие в конкурса – I, II, III, VI, X, XIV, XVII, XVII, XXI), на които са намерени 41 цитирания по непълна библиографска справка. Под ръководство на автора са защитени успешно 2 докторски дисертации и 12 магистърски дипломни работи по тематиката. По същата тематика авторът е ръководител на един национален и участник в три европейски научно европейски проекта, а именно:

- МУ-ФС-08/2007 "Гъвкава методология за разработване на софтуер" с Фонд "Научни изследвания", Министерство на образованието и науката, 2007-2011
- проект 288258 REuse and Migration of legacy applications to Interoperable Cloud Services - Enlarged EU REMICS с Европейската комисия, 7-ма Рамкова програма, 2011-2013
- FP7-REGPOT-2007 SISTER - Strengthening the IST Research Capacity of Sofia University - **(**Укрепване на научния капацитет на Софийски университет в областта на технологиите на информационното общество**)** с Европейската комисия, 7-ма Рамкова програма, 2008-2011
- IST-2001-34488 eXPERT: Best Practice on E-project Development Methods с Европейската комисия, 5-та Рамкова програма, 2002-2003

Софтуерните системи навлизат във всички сфери на живота ни, като същевременно стават все по-сложни. Проблемите, на които се търсят решения през последните 50 години от развитието на софтуерните технологии са свързани с това как разработването на софтуер да става „По-бързо, По-качествено, По-евтино". Един от начините са решаване на тези проблеми е прилагането на дисциплинирани процеси за разработване на софтуерни системи, които отговарят на спецификите на организаците и проектите. В тази глава 2 се представят гъвкавите методологии, като подходящ софтуерен процес водещ до по-бързо и качествено разработване на софтуер.

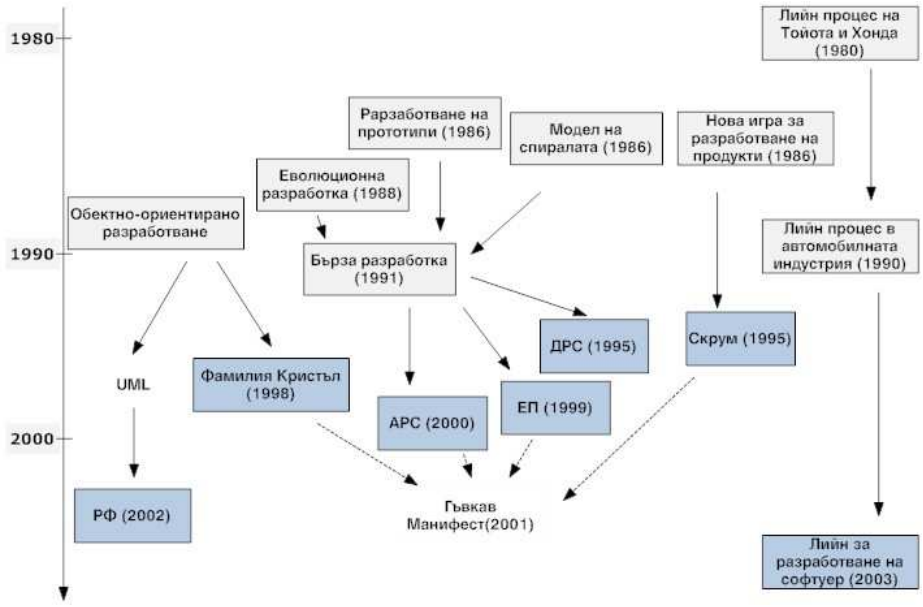
2.1. Обзор на предметната област

В настоящата секция се представя развитието на гъвкавите методологии и данни за разпространението им в различни организации. Представено е и сравнение на гъвкавите методологии кактo (Highsmith, 1999) (Highmith, 2001)о с традиционните методологии, така и помежду им. Изложението се базира на описанията в учебника (XXII), и двете докторски дисертации под ръководство на автора - (Кръстева, 2011) и (Иванов, 2006).

2.1.1. Гъвкави методологии за разработване на софтуер

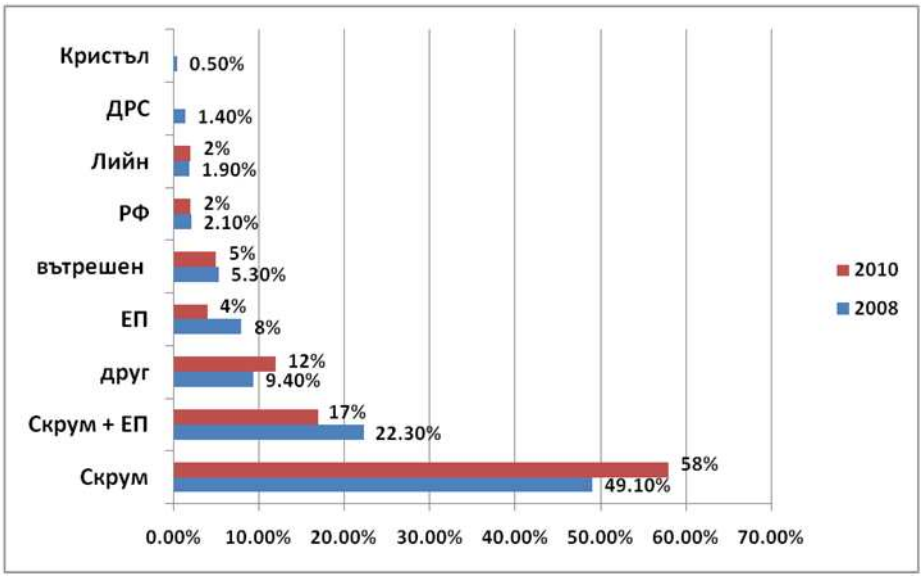
На Фигура 1 е представено развитието на гъвкавите методологии от традиционни методологии за разработване на софтуер. С по-тъмен цвят са означени гъвкавите методологии, докато стандартните са представени чрез светли правоъгълници. Моделът на еволюцията на гъвкавите методологии е представен в (Abrahamsson, Wasta, Siponnen, & Ronkainen, 2003) и е доразвит и адаптиран от (Кръстева, 2011). Разработването на прототипи (Prototyping) (Pressman, 2005), Еволюционната разработка (Evolutionary Life-cycle) (Abrahamsson, Wasta, Siponnen, & Ronkainen, 2003), Моделът на спиралата (The

Spiral Model) (Pressman, 2005), Бързата разработка (Rapid Application Development (RAD)) (Pressman, 2005) и обектно-ориентираното разработване (Pressman, 2005) поставят основата, върху която се развиват гъвкавите методологии: Динамичното Разработване на Системи – ДРС (Dynamic System Development Method (DSDM)) (Stapleton, 1997), фамилията от методологии Кристъл (Crystal family of methodologies) (Cockburn, Surviving Object-Oriented Projects: A Manager's Guide, 1998), Екстремното Програмиране - ЕП (Extreme Programming (XP)) (Beck, 1999), и Адаптивното Разработване на Софтуер – АРС (Adaptive Software Development (ADS)) (Highsmith, 1999). Гъвкавата методология Разработване на Функционалности - РФ (Feature-Driven Development (FDD)) (Palmer & Felsing, 2002) е фокусирана върху проекти, използващи обектно-ориентирани технологии, и добрият софтуерен проект е в основата ѝ. За моделиране на софтуерния проект се използва унифицираният език за моделиране UML (Unified Modeling Language) (Object Management Group). Скрум (Scrum) (Schwaber, 1995) е гъвкава методология, която доразвива и систематизира идеите на Такеши и Нонака (Takeuchi & Nonaka, 1986) в изследването, направено от тях, в областта на процесите за производство на продукти. В тяхното изследване са представени резултатите за повишена производителност на процесите за изработване на продукти, използващи малки самоорганизиращите си екипи с ниска специализация. По подобен начин принципите на Лийн (Lean) методологията за разработване на автомобили през 80-те години в Тойота и Хонда са адаптирани в софтуерната разработка в началото на 21-ви век от Мери и Том Попендик (Poppendieck & Poppendieck, 2003).



Фигура 1 Развитие на гъвкавите методологии (адаптирана от (Abrahamsson, Wasta, Siponnen, & Ronkainen, 2003)) (Abrahamsson, Wasta, Siponnen, & Ronkainen, 2003)

Приложението на отделните гъвкави методологии в практиката е показано на Фигура 2, която представя данни от проведени от компанията VersionOne изследвания през 2008 и 2010 година (VersionOne, 2008) (VersionOne, 2010). Както се вижда Скрум е най-широко прилаганата методология с около 50 процента приложение с тенденция към разширяване на приложението - приблизително 10 процента от 2008 до 2010 година. Хибридът между Скрум и Екстремното програмиране заема второто място по приложение. След тях с проценти между 10 и 2 се нареждат Екстремното програмиране, Разработването по функционалности и Лийн. Динамичното разработване на софтуер и фамилията Кристъл намират приложение в по-малко от 2% от проектите през 2008 година.



Фигура 2 Приложение на гъвкавите методи през 2008 и 2010 година

Според (Highmith, 2001), новото в гъвкавите методи не са практиките, които ползват, а поставянето на хората, като основен фактор за успеха на проекта и фокусиране върху ефективността и адаптивността на процеса на разработка. Това дава възможност за комбиниране на различни цели и принципи, които характеризират света на гъвкавите методологии.

2.1.2. Съпоставка на традиционни и гъвкавите методологии за разработване на софтуер

Традиционните методологии за разработване на софтуер се използват от началото на 80-те години. Те въвеждат систематизиран подход за разработване на софтуер, необходим на развиващата се софтуерна индустрия. Фокусът е поставен върху изчерпателната документация, проследимостта на софтуерната разработка от идеята до готовия продукт и постоянно наблюдение на процеса. Процедурите за контрол на качеството на продукта и подобряване на процеса на разработка са силно застъпени в традиционните методологии.

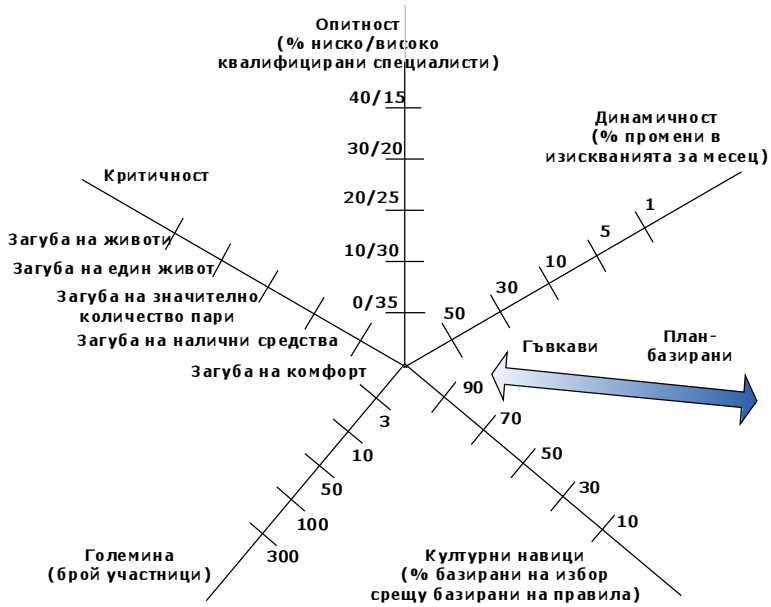
	ГЪВКАВИ МЕТОДОЛОГИИ	**ТРАДИЦИОННИ МЕТОДОЛОГИИ**
Тип комуникация	Неформална	Формална
Управленски стил	Базиран на водачество и сътрудничество	Базиран върху управление и контрол
Използвано знание	Неявно знание	Явно знание
Изисквания	Приоритизирани неформални изисквания и тестови сценарии; поддържат непредвидени промени	Изисквания във формализиран вид, определяне на възможностите на софтуера, потребителските интерфейси и качеството; предсказуема промяна на изискванията
Разработване	Софтуерен проект на високо ниво, къси итерации, лесна промяна на софтуерния проект	Изчерпателен и обхватен софтуерен проект, по-дълги итерации, повторното проектиране е нежелателен и скъп
Контрол на	Постоянно наблюдение на изискванията, проекта и	Формален контрол и късно тестване, базирано на планирана стратегия и предварително

качеството	разработваното решение; постоянно тестване	дефинирани дейности
Документация	Изчерпателна	Минимален необходим обем на документацията

Таблица 1 Основни различия между гъвкавите и традиционните методологии

За разлика от традиционните подходи за разработване на софтуер, гъвкавите методологии се фокусират върху възможностите за промяна на изискванията по време на изпълнението на проекта и графика на проекта, за да отговорят на бързо-променящата се бизнес среда. Тъй като е трудно тези промени да бъдат отразявани с изчерпателна документация и формални процедури, то се налага използването на неформални процедури и цялостно олекотяване на процеса. Гъвкавата разработка използва предимно знание в неявен вид, което се съхранява в хората като лично знание. Таблица 1 представя в синтезиран вид основните различия между традиционното и гъвкавото разработване на софтуер.

Боем и Търнер (Boehm & Turner, 2006) идентифицират пет основни характеристики на софтуерните проекти, които оказват влияние върху използването на гъвкави и традиционни методи за разработване на софтуер. Първата характеристика е *опитност на участниците в проекта*, която показва съотношението между високо и ниско квалифицираните специалисти, ангажирани в проекта. *Критичността*, определена от причинената загуба при настъпване на дефект в система, е втората характеристика. Загуба на човешки животи определя най-висока критичност на системата, следвана от загуба на само един човешки живот, загуба на значително количество пари, загуба на налични средства и загуба на комфорт определят степените на критичност. Следваща характеристика е *големина на проекта*, измерена в броя на служителите, ангажирани в проекта. *Културните навици*, възприети в организацията, са предпоследната характеристика. Организации, ориентирани към реда и следване на определени правила и процедури, предопределят по-различни културни навици от организациите, в които степените на свобода са ясно изразени. Последната характеристика е *динамичността*, определяща процента на промените в изискванията, които настъпват за месец. Фигура 3 представя диаграма, на която се вижда позиционирането на гъвкавите и план-базирани методологии спрямо петте характеристики на проектите. В центъра на графиката се поставят гъвкавите методологии, подходящи за ниско критични проекти, с преобладаващ квалифициран персонал, динамични изисквания, с големина до няколко десетки членове персонал, докато в периферията на графиката се намират по-тежките план-базирани методологии.



Фигура 3 Позициониране на гъвкавите и план-базирани методологии спрямо пет основни характеристики на проектите (адаптирана от (Boehm & Turner, 2006))

Оценката на една методология често означава съпоставянето ѝ с друга. В Глава 3 озаглавена "Гъвкави методологии за разработване на софтуер" на част 1 на учебника "Походи и методи за реализация на софтуерни системи" най-разпространените гъвкави методологии са разгледани систематизирано според *същността на софтуерния процес*, *ролите на различните членове на екипа*, тяхната дейност и отговорности, *практиките* препоръчвани и използвани в дадена методология, както и *препоръчителните условия, в които трябва да бъде прилагана*. На базата на структурираното описание гъвкавите методологии се сравняват по ключови характеристики (Таблица 2) и по покритие на жизнения цикъл на разработване на софтуер (фигура 3).

Съпоставка по ключови характеристики

Почти всички от гореизброените методологии споделят сходни характеристики и принципи. Въпреки това обаче всяка от тях подхожда към проблемите на софтуерната разработка от различен ъгъл. В Таблица 2 всяка една от методологиите е обобщена като акцентът е върху три основни аспекта: основни черти, особености, идентифицирани недостатъци. Под основни чери се има предвид ключови особености и основни принципи, а особенсоти са специфични отличителни черти и един или няколко аспекта на метода, с които той се отличава от останалите. Като недостатъци се смятат или идентифицирани недостатъци документирани в литературата или ако някои важни аспекти в софтуерната разработка, въобще не са разгледани в съответния метод.

Име	**Основни черти**	**Специални отличителни черти**	**Недостатъци**
APC	Адаптивна култура, сътрудничество, компонентно базирана, инкрементална архитектура	Организациите се разглеждат като адаптивни системи. Екип от близко свързани професионалисти Комуникация „лице в лице"	Отнася се по скоро към концепция и култура, а не толкова към практиката.
Кристъл	Фамилия методи. Методите във фамилията имат общи ценности и принципи.	Позволява избирането на най-подходящия метод според	Все още рано за оценка. Едва два от предложените четири

	Техниките, инструментите и ролите варират.	големината на проекта и критичността му.	метода реално се използват
ДРС	При управлението използва времеви блокове, упълномощени DSDM екипи, активен консорциум, следящ метода на разработка	Първият изцяло документиран гъвкав софтуерен метод, използване на прототипи. Няколко потребителски роли: посланик, визионер, съветник	За разлика от другите методи при този, единствено членовете на консорциума имат достъп по всички документи засягащи използването на този метод
ЕП	Софтуерна разработка повлияна от клиента, малки екипи, ежедневни builds	Refactoring – редизайн на системата с цел подобряване на нейната работа и възможността и за бъдещи промени	Малко внимание е отделено на практиките за ръководене на проекта
РФ	Пет стъпков процес, разработка ориентирана към отделни компоненти (парчета функционалност), Кратки итерации от часове до две седмици	Простота на нововъведенията, дизайн и имплементация на системата компоненти, обектно моделиране	FDD се фокусира единствено на дизайна и имплементацията. Трябва да се използва в комбинация с други методологии.
Скрум	Независим, кратки итерации, малки, самоорганизиращи се екипи, 30-дневен цикъл на разработка	Лесно се комбинира с други методи, олекотява организационните дейности Използва основните черти на гъвкавите ме-тоди – кратки итерации и планиране за самата итерация	Scrum детайлно описва как да бъде управляван 30-дневния цикъл по реализация, не включва практики по разработка, не се дефинират прак-тики за интеграция и приемни тестове от клиента.

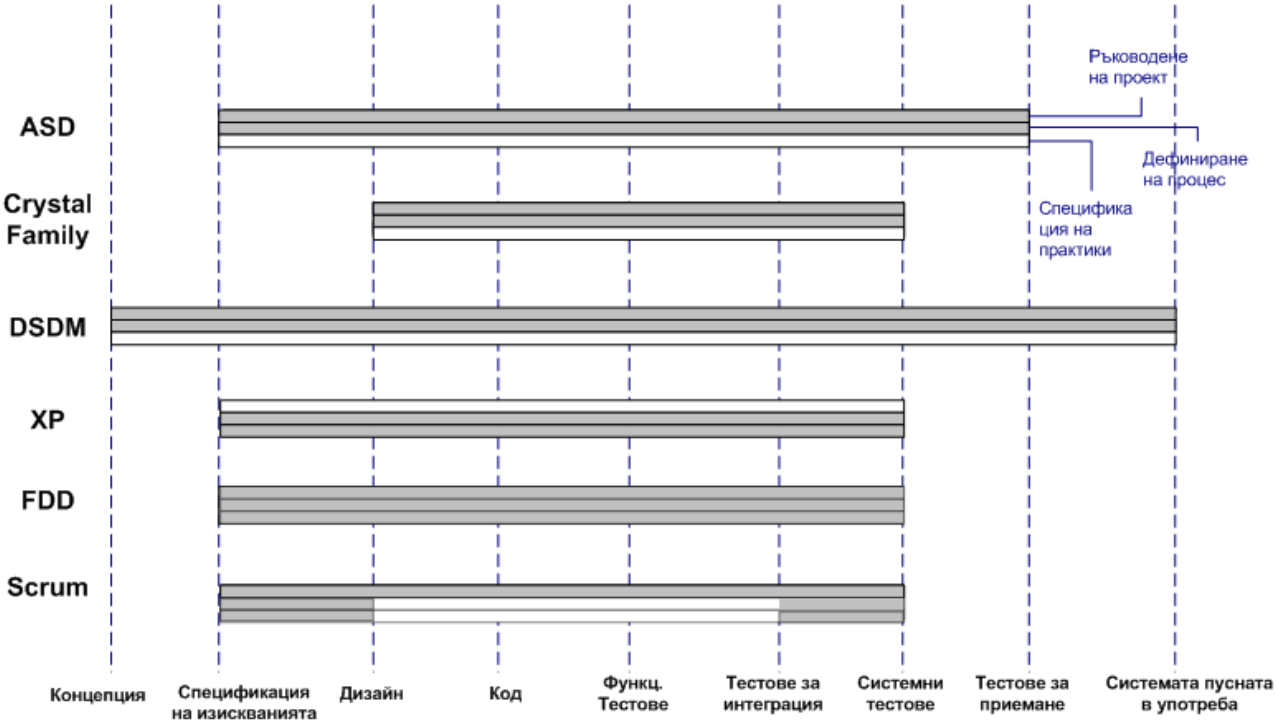
Таблица 2 Сравнение на гъвкавите методологии по ключови особености

Таблица 2 показва разликите в ключовите елементи, на които разглежданите гъвкави методологии акцентират.

APC е най-абстрактният от всички методи от гледна точка на софтуерната разработка. ЕП съдържа множество емпирично проверени практики, за които е преценено, че са полезни в софтуерната разработка. Фамилията методи Crystal единствена предлага различни вариации на методологията в зависимост от големината и критичността на проекта. Ключовият момент в ДРС е допускането, че проектите са с фиксиран срок и бюджет. Единствената променлива величина остава обемът на работата. Разработката се провежда, така че да се максимализира текущата полза за клиента. Различава се от другите методи по използването на прототипи. РФ няма за цел да предостави цялостно решение за софтуерна разработка, а се фокусира на прост пет стъпков подход базиран на идентифициране, проектиране и реализация на отделните задачи. РФ предполага, че част от работата по проекта е била вече направена, поради това не покрива някои от ранните фази на разработка. Скрум е методология за ръководене на проекти, разчитаща на самоорганизиращи се независими екипи. Имплементацията протича в цикли от по тридесет дни наречени спринтове. Интеграционните и тестове за приемане на системата не са обхванати от тази методология.

Съпоставка на гъвкавите методологии по жизнен цикъл

Фигура 4 показва кои фази на разработка са застъпени в различните гъвкави методологии. Всеки метод е разгледан в три основни области: Първата област показва дали методът дефинира практики и дейности за ръководене на проект. Втората показва дали методът осигурява поддръжка за дейностите на съответната фаза от процеса на разработка. Третата показва дали методът описва практиките, действията и средствата, които трябва да се прилагат и използват. В сиво са оцветени блоковете, които покриват съответната фаза, а в бяло – които не покриват.



Фигура 4 Съпоставка между гъвкавите методологии по жизнен цикъл. Фигурата е адаптирана от (Abrahamsson, Salo, Ronkainen, & Warsta, 2002), страница 95

Фигура 4 показва, че гъвкавите методологии са фокусирани върху различни аспекти от целия жизнен цикъл на софтуерна разработка. Ясно се вижда, че някои са фокусирани предимно на практическата страна на процеса за разработка (Екстремно програмиране), докато други са фокусирани предимно върху управлението на софтуерните проекти (Скрум). DPC покрива почти всички аспекти при разработката на един софтуерен продукт, без да дефинира конкретни практики за всяка една фаза.

Гъвкавият подход е практически приложим в ситуации, в които изискванията не са напълно ясни. В случаите, когато изискванията (напр. свързани с производителност, нова функционалност и др.) са ясни и могат да бъдат предварително специфицирани, гъвкавия подход не допринася с добавена стойност към разработвания проект.

Познаването на съществуващите методологии е особено важно за избора на конкретна методология в отделна организация, пред която предстои процес на преминаване към гъвкав модел за разработка на софтуер.

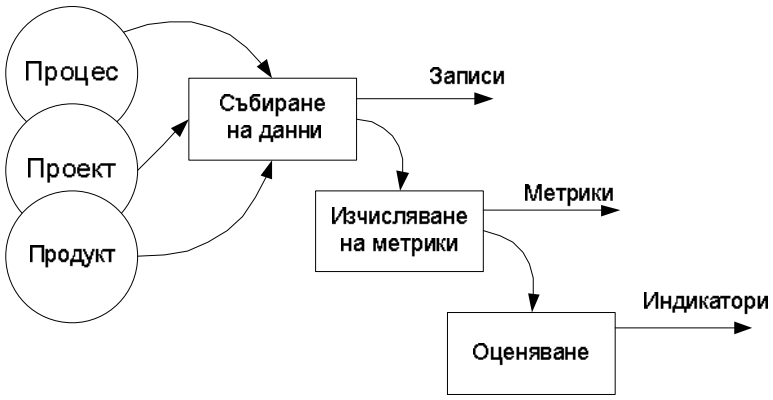
2.1.3. Метрики

За ефективното следене на софтуерните проекти данните, събирани за процеса за разработване, трябва да са количествени и обективни. За да се постигне това се използват софтуерни метрики. Софтуерните метрики са количествени "измерители", които измерват различни характеристики на софтуерната система. Всички инженерни дисциплини имат своите метрики – за тегло, за гъстота, дължина на вълната и др. Тъй

като софтуерът няма физически характеристики, стандартните метрики са неприложими. Затова са разработени специални метрики, които оценяват големина, сложност и надеждност на даден софтуерен продукт. Стойностите на някои метрики могат да се получат директно чрез измерване, докато стойностите на други се извеждат по косвен начин (наричат се индиректни метрики). Например някои от параметрите на качеството не могат да се измерят непосредствено. За оценяването са нужни модели. Ако стойността на някои метрика не може да се получи директно, тогава се построява модел за нейната оценка като се използуват стойностите на други метрики, които могат да се измерят. Моделът може да използва емпирични данни или може да бъде аналитичен.

Метриките, измерванията и моделите представляват едно цяло. Метриката дава количествена оценка на някое свойство, измерването дава действителната стойност на метриката за даден проект, а моделът осигурява стойностите на онези метрики, които не могат да се получат директно. Всяка метрика трябва да има цел и обикновено тази цел се свързва с основната цел на проекта - ниска цена и високо качество. Следователно всяка метрика, която не допринася директно за подобряване на качеството или намаляване на цената е интересна единствено от академична гледна точка. Например за лингвистите или специалистите по кодиране е може би важно в една програма какви са честотите на отделните символи, но от гледна точка на софтуерното производство тя няма практическо приложение.

Метриките за софтуер се анализират и оценяват от управляващите проектите и се консолидират от мениджърите на компаниите. Самото събиране на данни за метриките се осъществява от разработчиците.



Фигура 5 Процес за събиране и обобщаване на софтуерни метрики

Стъпките за събиране и оценка на метриките започват от дефинирането на събираните метрики за отделен проект (фигура 5). Обикновено се избират метрики, които се събират лесно и/или автоматизирано, за да не се допусне загуба на време и средства в процеса на събиране на данните. Резултатите от записите за метриките се анализират и се сравняват със средно приети стойности и/или подобни проекти вече реализирани в компанията.

Съществуват множество класификации на метриките (Манева, 2001) (Рендал, 1987), (Pressman, 2005) (McConnell, 1994). Тук е важна класификацията по следната категория (Pressman, 2005).

- Метрики за процес – събират се за всички проекти в рамките на организацията. Целта им е да представят набор от индикатори, които биха спомогнали за постигането на подобрение в процеса на компанията в по-дългосрочен аспект. Съществуват както “цялостни” метрики – за целия процес и резултат от

определена дейност, така и "лични" метрики, които са свързани с индивидуалното представяне на член от екипа (дефекти на хиляда ред код). По този начин ефикасността на проекта се измерва индиректно – чрез обобщаване на определени индикатори. Използват се, когато една организация би искала да повиши своето ниво на зрялост [CMMI].

- Метрики за проект – за разлика от метриките за процес, които се използват за стратегически за компанията решения и цели, метриките за проект се използват за вземането на тактически решения по време на самият проект. Консолидирането на метриките за проектите в една организация дават обобщените метрики за процеса. Целите в метриките за проект са две:
    - Използват се за намаляване на усилията, изразходвани по време на проекта и намаляване на продължителността. Да бъдат предотвратени евентуални забавяния от крайния срок за предаване.
    - Използват се за оценка на качеството на продукта в течение на процеса на реализацията му. С подобряването на качество намалява броят на дефектите, намалява времето за корекция и от там намалява и цената за реализация.

Друга класификация, която може да бъде направена относно софтуерните метрики е следната:

- Директни – метрики, които могат да бъдат директно извлечени от артефактите на проекта – линии код (LOC), скорост/производителност, брой дефекти и други.
- Индиректни – метрики, като сложност, качество, надеждност.

Според ДеМарко (DeMarco) „не може да контролираш това, което не можеш да измериш". Или казано по друг начин без софтуерни метрики еволюцията и подобренията в софтуерния процес са невъзможни.

2.2. Мотивация за изследванията

В сравнителен анализ на разпространените гъвкави методологии Абрахамсон (Abrahamsson, Oza, & Siponen, Agile Software Development Methods: A Comparative Review, 2010) поставя въпроса за нуждата от адаптиране на метода за разработка за всеки отделен проект и извежда необходимостта от дефинирането на ясни насоки за извършването на тази адаптация. Настройването на методите за разработка към конкретния проект се извършва на база на предходен опит от внедряването, който най-често се съхранява под формата на лично знание. Поради тази причина внедряването е локализирано в рамките на конкретна компания или проект, а повторното използване на натрупан опит е почти невъзможно.

Освен при адаптирането на методите за разработка, съществуват предизвикателства пред разширяване на областта на приложение на гъвкавите методологии. Ограничения пред прилагането на гъвкавата разработка са идентифицирани и в изследванията на Турк (Turk, France, & Rumpe, 2005) и Кручтен (Kruchten, 2008). Разпространените гъвкави методологии са базирани на предпоставки, ограничаващи тяхното прилагане - клиент на разположение по време на разработването на проекта, разработка в малки екипи, разположени на едно място и създаване на приложения с ниска степен на критичност. Въпреки идентифицираните ограничения са публикувани множество доклади, които свидетелстват за адаптирането и прилагането на гъвкави методи за разработване на софтуер в проекти и приложни области, считани за неподходящи (Conboy K. , 2009) (Hildenbrand, Geisser, Kude, Bruch, & Acker, 2008) (Paige, 2008) (Ge, Paige, & McDermid, 2010). Два големи европейски проекта – AGILE (Project A. , 2011) и

FLEXI (Project F.), предоставят резултати за успешното използване на гъвкави методи при разработването на вградени системи с високо ниво на критичност и надеждност, и при внедряването им в големи проекти с разпределени екипи.

Като общо заключение и глобална насока за бъдещо развитие на изследванията в областта на ГСР в (Abrahamsson, Oza, & Siponen, Agile Software Development Methods: A Comparative Review, 2010) е *определена необходимостта от създаването на малко и качествени методологии, които удовлетворяват множество ситуации на приложение*, а не много методологии с ограничено приложение.

В тази посока са насочени и усилията на членовете на инициативата SEMAT – Software Engineering Method and Theory. SEMAT [www.semat.org] инициативата е основана през 2009 година от Бертран Майер, Ричард Соли и Ивар Якобсон. Тя се занимава с предизвикателствата пред софтуерните технологии (software engineering), като обръща внимание на човешкия аспект, заедно с технологичния. Задачите, които си поставя са да преформулира дефинициите за софтуерните технологии и да подпомогне успешното разработване на съвременните сложни софтуерни системи.

SEMAT формулира основната си цел в едно изречение по следния начин: “Целта е да се създаде множество от елементи на ядро и език, които са разширяеми и лесни за използване, и които позволяват хората да описват същносттите на техните съществуващи и бъдещи методи и практики, така че те да могат да се композират, сравняват, оценяват, настройват, използват, адаптират, симулират и измерват от практиците, а също и да се преподават и изследват от научните среди.” [ ] Необходимо е да се отбележи, че *според терминологията на SEMAT методология, метод и софтуерен процес се използват взаимозаменяемо. Важно е да се разграничат метод от практики за разработване на софтуер. В настоящия хабилитационен труд авторът се придържа към това разбиране.*

В следващата секция ще се представят хронологично резултатите от изследванията, които следват очертаните тук насоки – от отделни гъвкави методи за определени организации и проекти, до метод за създаване на обобщена методология, която да се адаптира за различни ситуации.

2.3. Основни Резултати

През периода на изследване в областта с водещо участие на автора са предложени няколко нови гъвкави методологии. Извършени са и редица емпирични изследвания от тип казус за експериментално прилагане на предложените методологии. В следващите подточки ще се представят 4 нови гъвкави методологии. Представянето на методологиите ще следва структурата – описание същността на подхода, идентифициране метрики за оценка, резултати от внедряването.

2.3.1. Нови гъвкави методологии

2.3.1.1. eXPERT

Авторът е активен участник в европейския проект IST-2001-34488, в рамките на който през 2002 г е предложен гъвкавия подход eXPERT, който комбинира практики от Екстремното програмиране (ЕП) и Персоналния софтуерен процес (ПСП). Причината да се предложи тази комбинация е направеното проучване на литературните източници, което показва, че въпреки успешното прилагане на Екстремното програмиране (повишава се производителността на екипа, качеството на софтуера) организациите изпитват трудности с управлението на проектите, по специално с оценяване и

планиране на проекта в термините на цена и време. Това е логично, тъй като както е видно от Фигура 4, Екстремното програмиране не предлага практики за управление на проект. За преодоляване на тези трудности идеята е да се добавят принципи от ПСП, с които да се осигурат на екипите средства за измерване на тяхната ефективност, да се използват за по-добро планиране и управления на проектите и като краен ефект да се подобри удовлетвореността на потребителите на софтуера.

Практиките на методологията са обобщени в Таблица :

Практики от XP	Практики PSP
Планиране (Planning Game) Прост дизайн (Simple Design) Метафора (Metaphor) Стандарти за кодиране (Coding Standards) Тестване (Test before code – Unit and Acceptance Testing) Колективна собственост (Collective Code Ownership) Програмиране по двойки (Pair Programming) Постоянна интеграция (Continuous Integration) Рефакторинг (Refactoring) Отворено работно пространство (Open Workspace) Клиент при екипа (Customer on-site) 40 часова работна седмица (40 hours / week)	Следене на дефектите (Defect Logs) Записване на работата (Time Logs) Предложения за подобрение на процеса PROBE

Таблица 3 EXPERT Практики

В допълнение, за правилното прилагане на практиките от ПСП, подходът eXPERT предлага шаблони за:

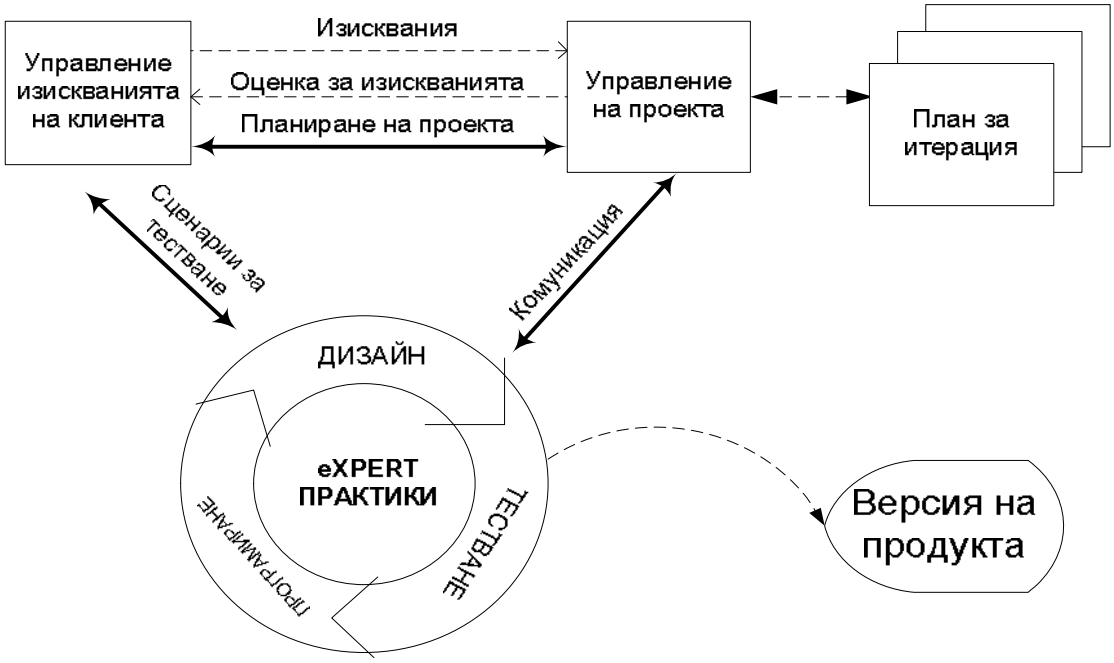
- Предложение за подобрение на процеса – всеки участник в проекта с eXPERT методология може да предложи начини за подобряване на някои елементи от процеса с цел подобрение на работата на екипа.
- Шаблон за следене на дефектите – за всеки един дефект се записва към кое клиентско изискване се отнася, с цел правилното следене на дейностите спрямо клиентските изисквания и правилното отчитане на работа според тях.
- Шаблон за отчитане на работата. Детайлите по този шаблон са разработени спрямо изискванията от PROBE методологията и по-точното отчитане на изразходваното време за разработка спрямо определена дейност според процеса и спрямо дадено клиентско изискване.

Архитектура на подхода eXPERT

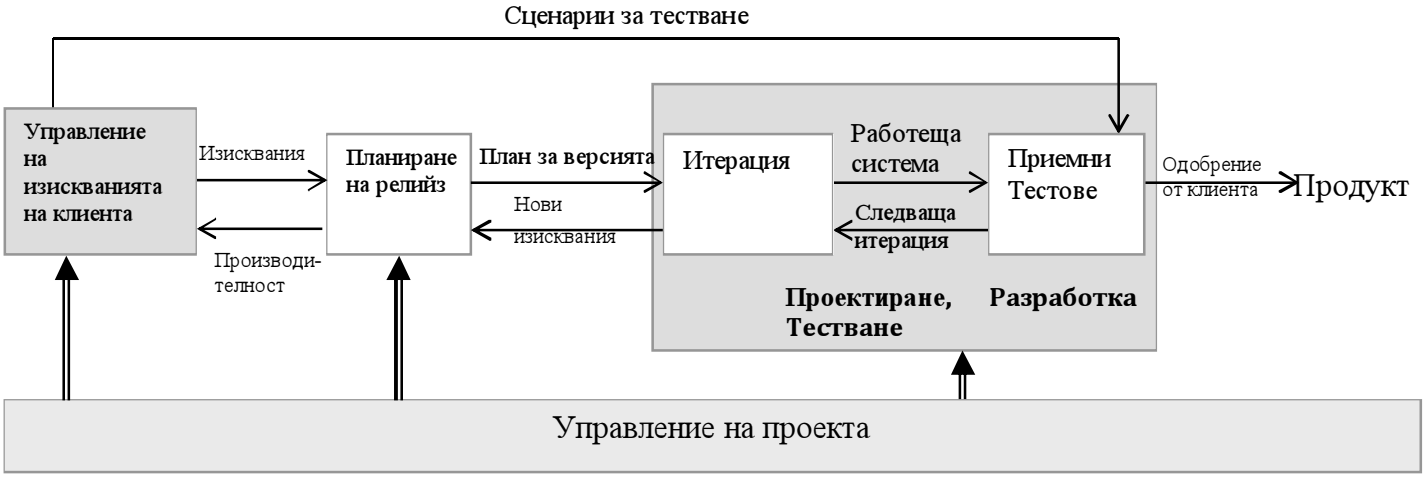
Подходът *e*XPERT се описва в термините на процеси поради две основни причини – повечето софтуерни фирми са повлияни от добре известните модели на софтуерни процеси като CMM, ISO and SPICE и ще им е по-лесно да разберат и приложат нов подход, който също е ориентиран към процеси и второ, този начин на описание улеснява сравнението между текущите и новите активности при разработването на софтуер.

Подходът *e*XPERT се състои от следните процеси: Управление на изискванията на клиента (CRM), Управление на проект (PM), Проектиране (DS), Кодиране (CD) и Тестване (TS). От своя страна процесите се описват в термините на: активности; задачи, които трябва да се свършат, за да се завърши активността; роля, отговорна за извършване на отделна задача.

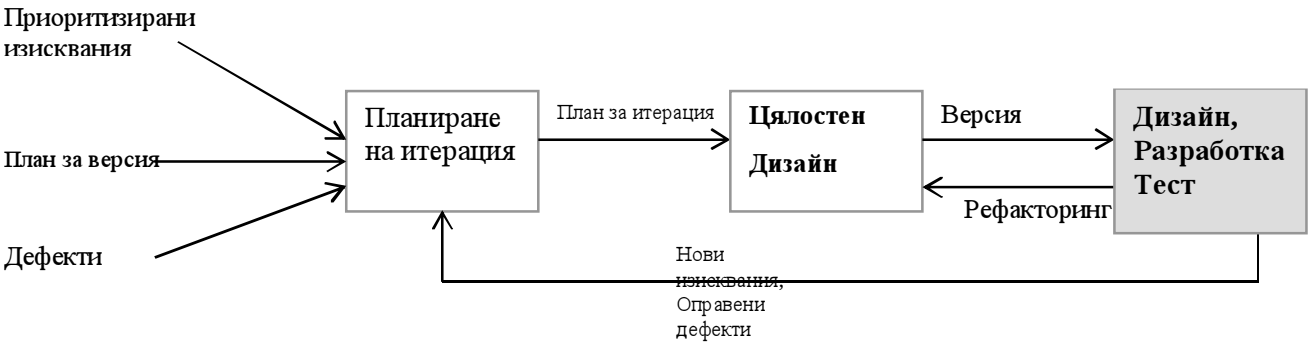
Фигура 6 дава най-обща представа за архитектурата на подхода. Фигура 7 показва динамиката на подхода чрез последователността от дейности и резултати дефинирани в процеса. Фигура 8 показва само дейностите изпълнявани в рамките на една итерация. Итерацията е дефинирана в термините на Екстремното програмиране. Целта на тези три фигури е да представят подхода от различна гледна точка и така, да се улесни разбирането му.



Фигура 6 – Архитектура на eXPERT процес



Фигура 7- eXPERT дейности



Фигура 8 – Дейности в една итерация

Метрики

За методологията eXPERT са дефинирани набор от метрики, които е необходимо да бъдат събирани по време на приложението му в пилотен проект (IST-2001-33488, 2005). Тези метрики помагат за да бъде направено заключение относно полезността на приложение на методологията и приемането й. Дефинираните и използвани метрики са следните:

- *Производителност* - Производителността се дефинира в два аспекта :

  - Размер на реализирания софтуерен код – хиляди линии код (KLOC) разделен на усилието/работата вложена за реализирането му;
  - Скоростта разделена на усилието от целия екип, като **скоростта** е сумата на оценките за времето за реализация на определени изисквания за дадена итерация.

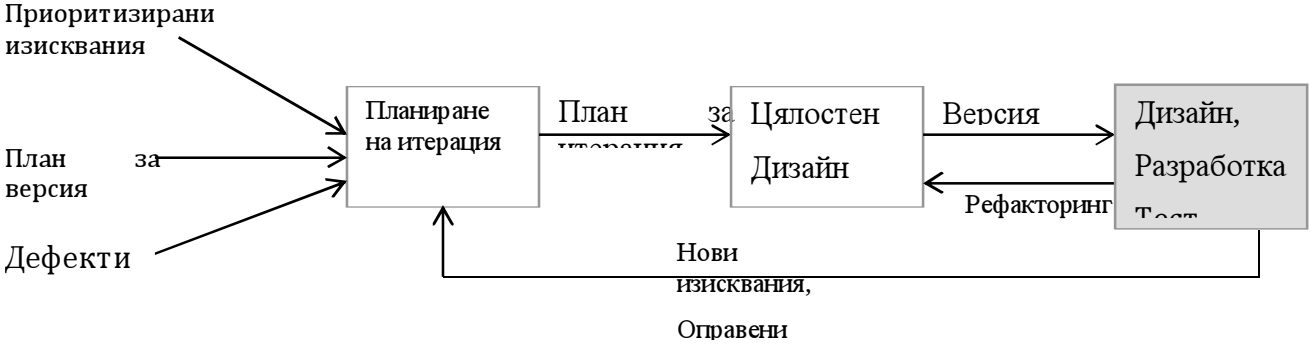
- *Дефекти* - Дефинирани са две метрики за измерване на дефектите:

  - Броя на дефектите допуснати от екипа или отделен член на екипа за определен период (фаза) на проекта или целият проект;
  - Процентно отношение на усилието изразходване за корекция на дефекти спрямо общото усилие за определен период (фаза) на проекта или целият проект

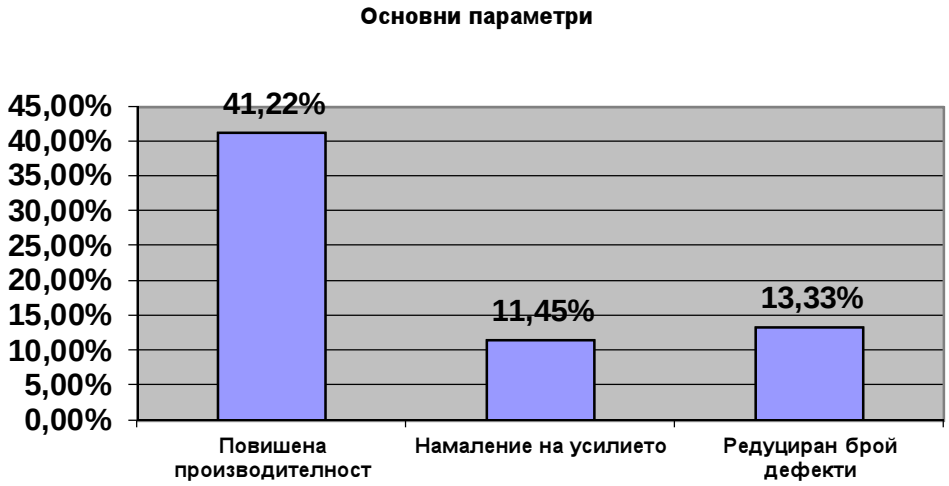
- *Отклонение от графика* - Отклонението от графика представлява отношението на реално изразходваното календарно време, спрямо планираното. Отклонението може да бъде измервано за всяко изискване, итерация или за целия проект.
- *Отклонение от разходите* Метриката за отклонението от разходите е подобна на описаната вече метрика за отклонение от графика, но гледна точка на разходите.
- *Оценяване чрез PROBE* - PROBE (PROxy Based Estimating) се използва за оценяване на необходимото време за разработка на всяко едно клиентско изискване.

Резултатите от прилагането на гъвкавата методология *e*XPERT в две български фирми са представени в публикации на автора 23 и 26.

При прилагане на eXPERT методологията в едната фирма са реализирани измерими



резултати, представени на фигура 9. Сравнението е направено на базата на два проекта – базисен (преди гъвкавата методология) и пилотен (прилагащ гъвкавата методология) и са представени в проценти. С цел представянето само на позитивни резултати на графиката, отрицателното изменение на усилието е представено като "Намаляване на усилието". Средната стойност на производителността за целият пилотен проект е 5.36 спрямо 3.8 за базисният проект. Цялостният брой дефекти за пилотния проект е 52 спрямо 60 за базисния.



Фигура 9 Сравнение на основните индекси за базисния и пилотния проект във фирма 1

Като обобщение, следните измерими резултати са постигнати след прилагането на eXPERT методологията в пилотен проект:

- Повишена производителност – според измерванията и представените резултати производителността на екипа е повишена с 41%. Основният фактор за това голямо повишение е намаленото време за поддръжка и промяна на съществуващ код и генерирането на по-голямо количество "нов" код на системата, което е породено от спецификата на клиентските изисквания, а именно реализацията на нови модули на системата.

- Намален брой дефекти – намаляването на дефектите е с 13.33%. Факторите за намаляването на броя на дефектите се дължат основно на програмирането по двойки, и модулните тестове на системата, реализирани според препоръчаната от eXPERT практика.

- Намалени разходи на компанията за реализация – като основа за намалените разходи за реализация на системата се явява показателят за намалени усилия за реализация. Намалението на усилията е средно с 11,45%.

- Не са регистрирани закъснения според плана (крайната дата за приключване на проекта) – Основният фактор според екипа е постоянната връзка с клиента и постоянното използване на данни предоставени от клиента след ежедневен преглед на работна версия на системата.

- Не са регистрирани допълнителни от предвидените разходи по проекта.

Намаляването на броя дефекти е значително - 41,9%. Факторите, които водят до това са стриктното прилагане на практиката тестване на единица, която спомага за ранното откриване и отстраняване на грешки в кода, както и прилагането на практиките прост проект (design), рефакторинг, програмиране по двойки и комуникация с клиента.

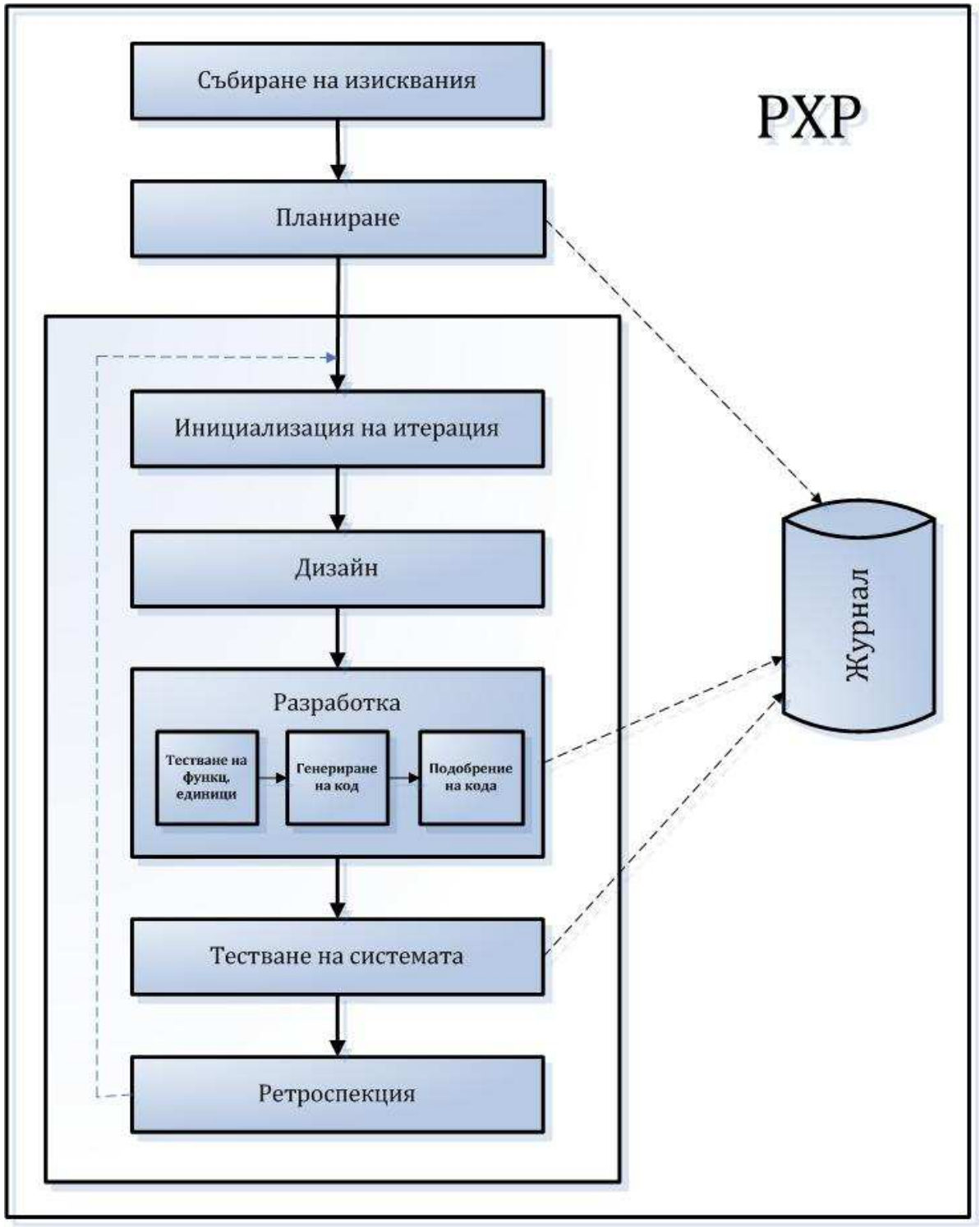
2.3.1.2. Методология за индивидуално програмиране - Personal Extreme Programming

След получените резултати за успешното прилагане на гъвкавия метод eXPERT в различни организации възникна идеята за комбинирането отново на ПСП и ЕП в нова методология, но този път с цел да бъде подходяща за индивидуално използване, вместо в екип. Това означв, че подходът ще трябва да е различен – да се стъпи върху ПСП и да се обогати с подходящи практики от ЕП. След развитието на идеята се стигна до предложение на нов гъвкав процес наречен Персонално Екстремно програмиране (ПЕП) (Personal Extreme Prgoramming - PXP). Основната цел на ПЕП е да бъде олекотен вариант на ПСП, като запази принципите му и акцента върху събирането на данни (за процес и проект), но в по-неформален формат. Разработката в ПЕП е итеративна и прилагайки доказаните практики от ЕП се очаква програмистът да успява да бъде по-гъвкав при появата на промени. ПЕП адресира нуждата от по-големи, по-сложни, и по-сигурни софтуерни системи, изградени според плана от автономни разработчици.

Основни принципи на ПЕП са:

- Качеството на софтуера, зависи от разработчика и софтуерния процес
- Разработчикът е отговорен за своят личен процес
- Разработчикът трябва да измерва, проследява и анализира своята работа
- Разработчикът трябва да се поучава от разликите в производителността си
- След всеки проект, разработчикът трябва да се стреми да подобрява процеса си, като се базира на натрупана информация от предишни проекти
- Разработката е на базата на постоянно тестване, като се цели откриване на дефекти на възможно най-ранен етап, когато отстраняването им струва по-малко
- Разработчикът трябва да автоматизира възможно най-много задачи от ежедневната си работа, като това му позволява да се фокусира върху проблемите и задачи от по-високо ниво

ПЕП съдържа 6 практики от Екстремното програмиране (Continuous Integration, Simple Design, Small releases, Refactoring, Test Driven Development, Spike Solutions) и 6 от ПСП (Time Recording, Defect Type Standard, Defect Recording, Size Measurement, Process Improvement Proposal, Code Reviews). Връзките между тези практики са показани на Фигура 11.



Фигура 10 – Фази на процеса за разработка PXP

Отделните фази на предложения процес за разработка ПЕП (фигура 10) са следните:

0. **Събиране на изисквания** – по време на тази фаза се генерира документ с функционални и нефункционални изисквания за системата, която ще бъде разработвана. Тази фаза е незадължителна – документът може да бъде генериран при (1) среща между служител на организацията, за която разработчикът работи, и клиента, или (2) може да бъде наследен от друг разработчик, в случай на разширяване на вече съществуваща система.

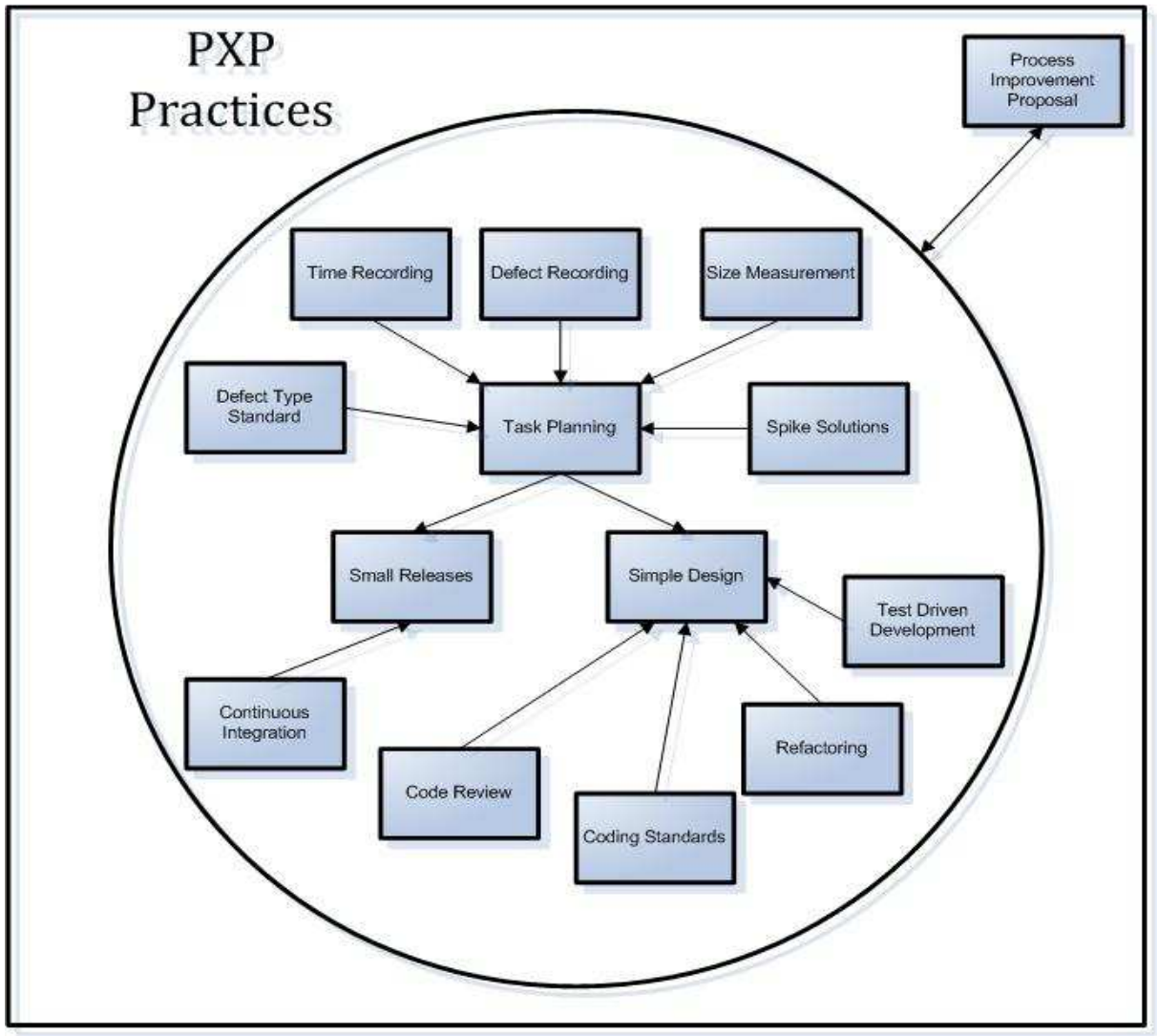
1. **Фаза на Планиране** – възоснова на документа с функционални и нефункционални изисквания, разработчикът генерира списък от задачи за проекта. Всяка задача трябва да бъде разделена на множество малки задачи, които могат да бъдат категоризирани. Всяка малка задача се оценява на базата на журнал от предишни проекти с оценка на задачи от същата категория (ако все още не е създаден журнал, разработчикът се опитва да направи най-добро предположение). Сумата от оценката на множеството малки задачи, е оценката на общата задача. По време на фазата на планиране, преди дефинирането на задачи и оценката им, се вземат основни архитектурни решения относно език за програмиране, платформа за разработка, модел на приложението (напр. уеб или десктоп приложение) и др.
2. **Инициализация на итерация** – разработката на софтуер е итеративна. Всяка итерация стартира с избор на задачи, които ще бъдат фокус на итерацията, от общия план на проекта. Продължителността на една итерация може да варира между 1 и 3 седмици, в зависимост от обхвата на проекта и начина на разбиване на планираните задачи. Итерация може да завърши както с кандидат-версия на проекта за преглед от клиента, така и версия за пускане в реална употреба.
3. **Фаза на Проектиране** – през фазата на проектиране автономният разработчик моделира софтуерните компоненти и програмни единици, които трябва да бъдат разработени в текущата итерация. Програмистът трябва да има за цел да изгради проект (design) на системата само за текущите изисквания поставени от клиента, без да се опитва да предполага какви функционалности ще се разработват за в бъдеще. Методът на проектиране е избор на разработчика, като препоръката е да се използват възможно най-простите средства като диаграми, бели дъски, CRC карти и други.
4. **Фаза на Разработка** – през фаза на разработка се осъществява същинското конструиране на кода. Разработчикът кодира всички компоненти, програмни единици и бази данни дефинирани в предната фаза на проектиране, както и реализира (implements) тестове за тях. Тази фаза е разделена на три подфази, които се изпълняват в следният ред:
    - **Тестване на функционални единици (Unit Testing) –** за всеки един клас, дефиниран във фазата на проектиране, се разработват тестове за функционални единици, преди реалната му реализация
    - **Генериране на код (Code) –** реализира се тялото на класовете и методите дефинирани в предната фаза на проектиране. Изпълняват се всички тестове и кодът се модифицира докато те преминат успешно.
    - **Подобрение на кода (Refactor)** – внасят се промени в кода с цел да се подобри проекта (design).

За да се премине към следващата фаза, кодът трябва да се компилира без грешки и всички създадени тестове в тази фаза да се изпълняват успешно.

5. **Фаза на Тестване на системата (System Testing)** – извършва се тестване на разработената до момента функционалност. Проверява се до каква степен реализираното решение отговаря на функционалните и нефункционални изисквания. Всички открити дефекти се записват и отстраняват от автономния разработчик.
6. **Фаза на Ретроспекция (Retrospective)** – с тази фаза завършва една итерация от процеса на разработка. Прави се анализ на всички данни събрани по време на останалите фази на итерацията. Автономният разработчик трябва да си направи равносметка дали първоначалната оценка на задачите е била адекватна, какви са били причините за евентуални забавяния, с цел планирането да се подобри в

прилагането на процеса за следващи проекти. Разработчикът анализира и всички предложения за подобряване на процеса и ако се налага внася промени както в процеса така и в съпътстващите го практики. Промените трябва да бъдат направени възможно най-скоро, за да се отстранят евентуални възникнали проблеми. Адресирането на тези проблеми в края на проекта може да бъде прекалено късно и да доведе до провал.

Тази фаза може да завърши както с кандидат-версия (release candidate) така и с версия за пускане в употреба. Кандидат-версията е работеща версия, която се представя на клиента за получаване на обратна връзка. Фазата на ретроспекция може да премине отново във фаза на „Инициализация на итерация", което маркира стартирането на нова итерация, или да маркира приключването на разработката на проекта. Приключването на проекта, настъпва когато всички дефинирани изисквания за софтуерната система са изпълнени, няма останали непоправени дефекти (всички открити дефекти са поправени) и клиентът е приел проекта.



Фигура 11 – Практики на ПЕП и начини на взаимодействие

Метрики

Следните метрики са събирани и анализирани при прилагането на ПЕП методологията за разработване на реален софтуер:

· Усилия за планиране – времето, отделено за фазата планиране

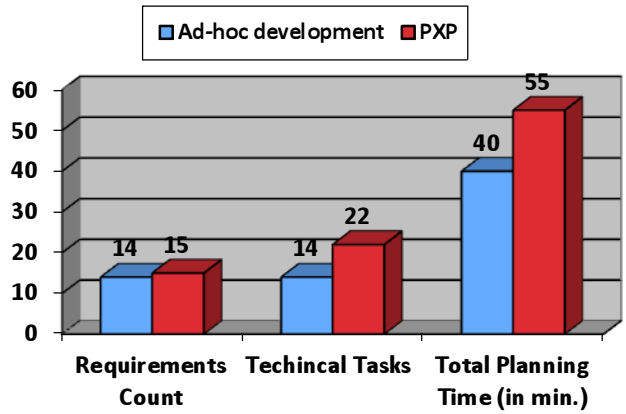
· Ефективност на планирането – отношението между планираното и действителното време, отделено за дадена задача

. Отношение между намерени дефекти и брой функционални и нефункционални изисквания

· Покритие на кода от тестове на функционални единици – процентът на кода, който е покрит когато се изпълняват тестовете. Тази метрика позволява проследяване коя част от приложентието може да се тества автоматично

Резултати

Фигура 12 представя информация за усилията за планиране в две фази. И двете фази покриват приблизително еднакъв брой изисквания за всеки метод на разработване (начален - 14 изисквания, и ПЕП – 15). Тъй като няма събрани предишни данни за други проекти, разработвани с ПЕП, то планирането и в двете фази става относително по един и същ начин. Основната разлика е в броя на технически задачи, които са дефинирани – ПЕП дефинира по-дребнозърнести задачи в сравнение с началния метод на работа. На първа фаза да дефинирани по 1 задача на изискване, т.е. 14 задачи, докато за втората фаза са дефинирани 22 задачи. При ПЕП времето, отделено за планиране е 37.5% повече, отколкото при началния метод, защото броят на задачите за оценка е по-голям.



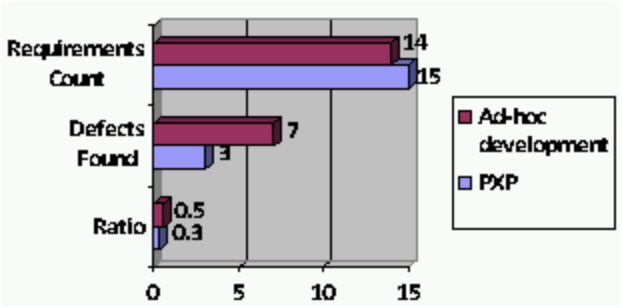
Фигура 12 – Резултати от фазата планиране при начален процес и ПЕП

Таблица 4 илюстрира ефективността на планирането на двете фази. При началния метод не се следи времето по задачи, но допълнителното време за разработване е с 16ч повече, което означава 20% повече от планираното. При ПЕП следенето на времето е една от основните практики и има повече информация – 5 задачи са завършени преди планираното време, 3 задачи са завършени навреме, и 14 were задачи са завършили след планираното време. Или общо действителното време за разработване при ПЕП е 67,5 при планирано 58ч. Резултатите показват, че ефективността на планирането и в двете фази на проекта е подобна, като причината за това е, че самото планиране се извършва по подобен начин. ПЕП има малко по-добра ефективност на планирането поради това, че са дефинирани по-галям брой задачи – по-лесно е да се оцени по-добре множество от малки задачи в сравнение с оценката на една голяма задача.

Таблица 4 Данни за ефективност на планирането при начален процес и ПЕП

	Начално	ПЕП
Брой задачи, които са надценени	-	5
Брой коректно оценени задачи	-	3
Брой задачи, които са подценени	-	14
Общо планирано време за разработване (в часове)	80	58
Разлика между действително и планирано време	16ч (20%)	9,5ч (16,3 %)

Броят на откритите дефекти по време на втората фаза е значително по-малък в сравнение с първата фаза. (Фигура 13). ПЕП се фокусира върху практиката ръководена от тестове разработка, което позволява дефектите да се предотвратят още по време на фазата имплементация. При началния метод на разработване се е извършвало само ръчно тестване е част от дефектите са въведени при разработване на нова функционалност.



Фигура 13 – Отношение между намерени дефекти и изисквания и в двете фази на проекта

Последната метрика е процентно покритието на кода с тестове. При началния метод тестовете на функционални единици не са били реализирани и се смятат за 0%. При ПЕП една от основните приложени практики е ръководена от тестове разработка, което води до покритие на кода 85.29%. Колкото по-високо е нивото на покритие на кода толкова по-лесно е разширяването и поддържането на софтуерната система и съответно с по-малко ръчни тестове.

2.3.1.3. Методология Екстремно програмиране и Скрум

Целта на предлаганата методология е да се подобри съществуващ процес, като се отчете спецификата на проекта. Текущият процес в организацията е вариация на процеса RUP – Rational Unified Process.

Особеностите на проекта, към който ще се прилага са:

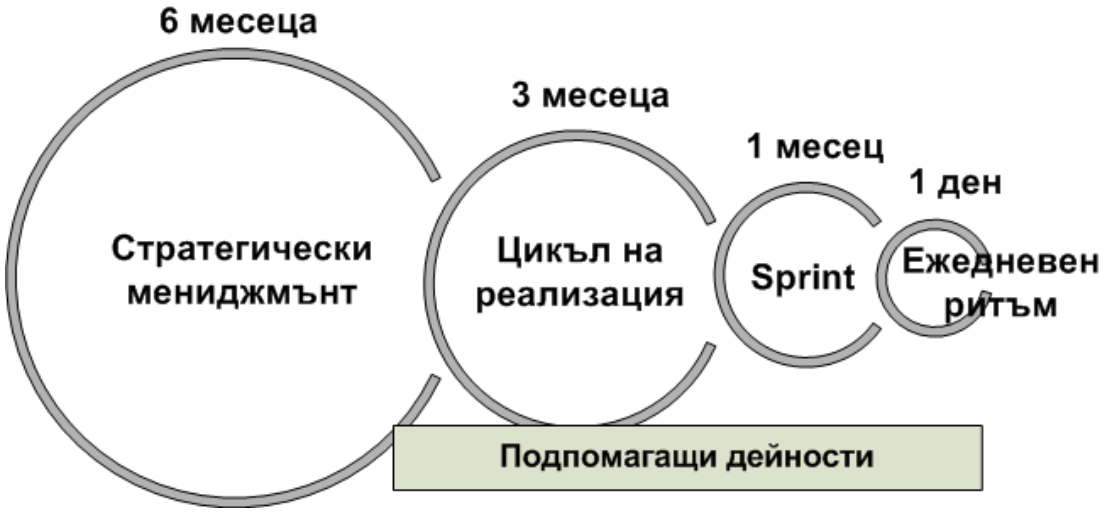
- Предназначен за държавната администрация
- Непълна спецификация и изисквания към продукта
- Необходимост от използване на технологии и продукти, с които екипът разработчици няма предишен опит
- Интеграция с външни системи

Изискванията към новия процес са:

- Процес водещ до подобряване качеството на софтуерния продукт
- Процес, който прави разработката предвидим за клиента. Позволява пряка комуникацията между разрабочиците и клиента
- Подобрява комуникацията вътре в екипа
- Налага ефективен контрол върху процеса на разработка

Новата методология, представлява комбинация от практики за управление на Скрум и практики за разработка на Екстремно програмиране. Тя се проектира в съответствие с горните изисквания и се основава на рамката 4CC http://www.allbusiness.com/professional-scientific/architectural-engineering/1003074-1.html. Основните концепции са (Фигура 14):

- *Стратегически мениджмънт.* Формира цялостната визия за проекта. Изготвя времева рамка за проекта, съобразена с технологията и пазарните изисквания. Фокусиран е върху дългосрочни задачи, като например стартирането на даден продукт или прекратяване/замразяване работата по даден продукт.
- *Цикъл на реализация*. Свързан с разработката на нов продукт или нова версия на даден продукт. Разработка на визията за продукта, която може да е синтез от обратната връзка с клиента, проучвания върху пазара, пътната карта за продукта и други. Завършва с произвеждането на нов продукт или нова версия на продукта.
- *Спринтове (Sprints).* Произвежда следващия артефакт (документация или функционалност) заложен в пътната карта на продукта. Спринтовете имат различни атрибути и организация, които обясняват основната цел на спринта. Спринтовете винаги завършват със спринт ревю сесия, където новата функционалност бива демонстрирана. Съответно спринтовете започват със сесия за планиране, при която се избира кои функционалности да бъдат разработвани в рамките на настоящия спринт, както и определяне на задачите и времето им за изпълнение на базата на избраните функционалности.
- *Ежедневен ритъм.* Управляване и мотивиране на разработката спрямо целите на спринта. Работата на отделните членове и екипи участващи в разработката се синхронизира и продукта се интегрира и тества непрекъснато.
- *Подпомагащи дейности.* Непрекъснати дейности, които подпомагат и насочват разработката. Тяхната основна цел е да предоставят ефективни методи и инструменти нужни за управление на конфигурациите, управление на изисквания, проследяване на дефектите, ежедневни работещи версии и тестване.



Фигура 14 Диаграма на нова методология

Метрики

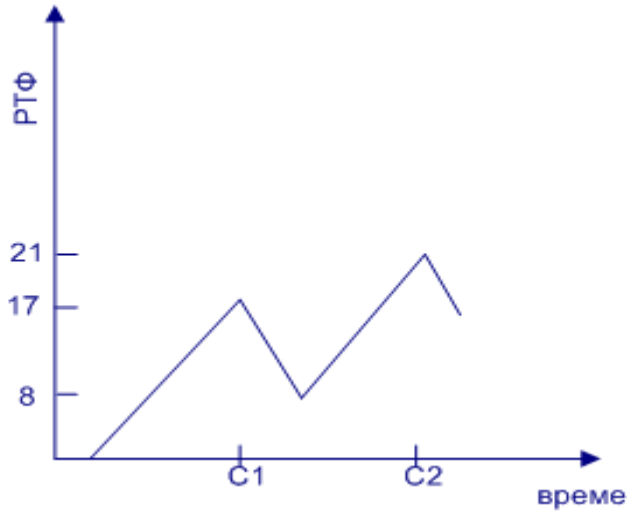
За новата методология са идентифицирани следните метрики, които са фокусирани върху три качествени аспекта на софтуерния процес – произвеждане на исканата функционалност в срок, програмният код и неговата поддръжка, контрол на дефектите в софтуерния продукт:

- RTF метрика - Идеята на всеки софтуерен проект е да създаде софтуер, който работи и в максимална степен покрива изискванията заложени в началото на проекта (Jeffries, 2004). Тази идея може да се нарече Работещи Тествани Функционалности (РТФ), за която е валидна следната дефиниция:
  - Разработваният софтуер е разделен на части, т.нар. именувани функционалности (изисквания, истории),
  - За всяка от разработваните характеристики има един или повече автоматизирани теста за приемане. Ако те работят това значи, че функционалностите са реализирани правилно.
  - РТФ метриката показва във всеки момент от изпълнението на проекта колко от разработваните характеристики покриват тестовете за приемане на системата (acceptance tests).
- Метрики за изходния код
  - *Сцепление* - Сцеплението (cohesion) е мярка за дизайна на софтуера, която показва доколко отделните компоненти изпълняват целите, за които са създадени и комуникират добре. При добрия дизайн, сцеплението между отделните компоненти е голямо, те добре взаимодействат и изпълняват функциите си.
  - *Кохезия* - Свързаността (coupling) е мярка за дизайна на софтуера, която показва доколко отделните компоненти са свързани помежду си. Наличието на връзка между два компонента води до зависимост, което е от съществено значение при поддръжката и използването на всеки от компонентите. Под връзка се разбира всякакъв вид, зависимост вследствие на която, промяната на единия от компонентите може да предизвика промяна в другия. При добрия дизайн, свързаността между отделните компоненти е минимална, всеки компонент е максимално независим от останалите.
  - *Цикломатична сложност* - Това е метрика на ниво метод и се използва за измерване на структурната сложност на методите.

Резултат

Както се вижда от фигура 15 (С1 означава първата итерация, а С2 втората) РТФ кривата спада в началото на втората итерация С2. Това означава, че има функционалност и тестове за тази функционалност, които не минават. Причина за този спад е възприетата практика в началото на всяка итерация да има период от два дни до седмица, в който се прави рефакторинг и подобрения на изградената през предходния цикъл функционалност. Идеалният случай е, когато има постоянно растяща РТФ крива. Отчитането на този спад води следните последствия върху процеса на разработка:

- Обсъждане и рефакторинг на вече реализирана функционалност. Това е много важно за подобряване качеството на проекта (design) и реализацията (implementation).
- След всяка итерация работата по рефакторинг на кода намалява. Това е в резултат на извършения вече рефакторинг на предишната итерация. По тази начин се получава постепенно изправяне на РТФ кривата.



фигура 15 РТФ крива за две итерации

Данни от наблюдаваните метрики за изходния код са представени в (XVII).

В резултат на прилагане на предложената методология процесът на разработка в организацията става по-отворен към клиента; обратната връзка с клиента след всяка итерация спомага за подобряване на разбирането за функционалността на системата. Инкременталният подход за реализация на продукта дава възможност след всяка една итерация да се демонстрира стабилен и работещ софтуер.

Въведените в процеса гъвкави практики за разработка, тестване на готовата функционалност, събиране и анализиране на различни метрики водят до подобрения и в техническата реализация на проекта. Реализацията на нова функционалност става предвидим и прозрачен за клиента процес.

2.3.1.4. Методология RUP и Скрум

Подобно на методологията, предложена в предишната секция и тази има за цел да въведе гъвкави практики към текущия утвърден процес. Въведени са индикатори, по които дадена организация може да категоризира проектите си:

- *технически аспект на проект* – има различни типове проекти според тяхната сложност, технологияи тяхното място сред другите аспекти на индустрията и технологията

- *тип на клиента* – някои клиенти знаят от самото начало точна как трябва да изглежда крайния продукт, а други имат бегла представа за функционалността на системата и изискванията се уточняват в процеса на разработване

Организациите търсят универсална методология за разработване, която да е подходяща за цялата фирма, като се адаптира за различните типове проекти спаред горните индикатори. Целта на изследването е да се установи успешно използване на нов метод, който е комбинация от RUP и Скрум и още по-важно – дали новият процес е съвместим със сертифициран по ISO процес. Известни са данни да такава съвместимост, но както вече беше казано няма два еднакви проекта и фирми и затова трябва конкретно да се прецени в зависимост от текущия проект валидността на предположението. Конкретният продукт е софтуерна система за управление на битови уреди и системи, която се инсталина в дома на клиента и той има достъп през браузър за използване на системата.

И двата процеса RUP и Скрум са итеративни и имат общи характеристики. Като гъвкав процес Скрум е проектиран за малки екипи – не повече от 10 човека. От друга страна RUP е итеративен, но няма дефинирани спринтове с ограничения във времето за итерациите. Също така не поставя ограничения за размера на екипа.

Новата методология съчетава предимствата и на двата процеса, като дефинира спринтове с продължителност 4 седмици, като има дефинирано време за специфициране и одобрение на спецификацията в началото на проекта. След като се одобри от клиента архитектурата на системата се стартира постъпкова разработка по спринтове. Накрая на всеки спринт е определена дискусия с клиента, която не може бъде пропусната. Създава се списък с всички известни до момента изисквания. Изискванията могат да са от клиента, от отдел продажби, от отдел поддръжка (съпровождане) или други заинтересовани лица.

Много важен фактор в процеса т.нар. срещи, които се провеждат накрая на всяка итерация. Тези срещи продължават 2 дена, като се представят постигнатите резултати по време на завършилата итерация и се определят и приоритизират изискванията за следващия спринт.

Проектът, за който е предназначена методологията има следните характеристики (таблица 5)

Таблица 5 Основни характеристики на проекта

Параметър	Стойност
LOC	190kLOC
Действителни усилия	15 700 човеко часа
Размер на екипа	до 20 човека

Усилията по проекта включват не само времето за реално разработване, но и времето за управление, командировки и срещи, и др. Линиите код са само кода, който е предназначен за клиента, без да се има предвид кода наредица фирмени продукти (напр. технологичната рамка OSGi), които се използват от фирмата и са част от цялата софтуерна система.

Разпределението на усилията по типове дейности в проекта е следното (таблица 6)

Таблица 6 Разпределението на усилията по типове дейности

Тип дейност	Усилия
Командировки	5,15%
Спецификация	18,45%
Проектиране	5,18%
Разработване	49,61%
Осигуряване на качеството	10,61%
Управление на проекта	6,52
Други	4,48%
Общо	100%

В категорията други са усилията за администриране на работната среда, проучване и усвояване на нови технологии и др. Както се вижда въвеждането на спринтове и месечни срещи след спринт води до чести командировки, които са 5,15% от времето за изпълнение на проекта. Очаквано след въвеждането на гъвкавите принципи усилията се изместват повече към разработването, отколкото за проектиране. Тъй като това е първи проект по новата методологии все още усилията за изясняване на спецификацията са значителни, но приемливи предвид удовлетвореността на клиента и качествената работа в срок.

2.3.2. Внедряване на гъвкавите методологии

Възоснова на проучвания и личен опит се установява, че процесът на внедряване на нова методология в дадена организация не е тривиален. Не е досатъчно да се познава новата методология, а трябва да се определи каква е целта, и съответно да определят стъпките, и процедурите по внедряването.

2.3.2.1. Процедура по внедряване на eXPERT методологията

В резултат на пилотно прилагане на eXPERT методологията в организация под наблюдението на автора са систематизирани следните няколко основни стъпки, през които се препоръчва да мине внедряването на гъвкава методология във фирма със сертифициран софтуерен процес:

1. Идентифициране и анализиране на разликите между съществуващия процес на работа в компанията и избраната гъвкава методология. Този анализ, повлиян от английската терминология, е т.нар. Анализ на различията – (GAP) Анализ;
2. Дефиниране на промените, които трябва да бъдат приложени;
3. Прилагане на новата методология за разработване в рамките на текущата фирмена система за управление на качеството (СУК);
4. Създаване на ръководства за екипа относно новият процес на работа в компанията;
5. Обучение на персонала – обучение по новия фирмен процес (а в конкретния случай и обучение по Екстремно програмиране и Персонален софтуерен процес);
6. Прилагане на новата методология при разработката на реален проект;
7. Представяне на ясни и сравними резултати на ръководството на компанията относно ефекта от прилагането на новата (eXPERT) методология.

Особено критични за успеха на внедряването са първите 2 стъпки, на които се отделя специално внимание. Извършеният анализ дава детайлна информация за това какви са

различията между съществуващата методология, размерът на тези различия и тяхното отражение върху работата по проектите водени с новата методология.

Анилизът в конкретния случай се извърша на базата на следните основни елементи:

- Документация;

След извършения анализ на различията относно документацията, първият очевиден резултат, който облекчава текущия процес на разработка на софтуер, е намаления брой документи, съпътстващи всеки проект. Броят на поддържаните документи за проект е намален със седем. Така общият брой на поддържаните документи за проект е 43, но основното постижение е, че най-тежките документи по време на жизнения цикъл на проекта, а именно "Детайлна спецификация на изискванията" и "Техническа спецификация" не се подготвят със всички детайли.

- Ръководство на процеса на реализация;

Като цяло, последователността от действия и работния процес в разработката на решението съобразно eXPERT във фирмата се различават от настоящия процес в организацията в следните по-важни аспекти:

- Нивото на детайлност на двата процеса е различно, подходът eXPERT предефинира ролите, отговорностите и стъпките на ниво личност, докато сертифицираният процес на организацията прави същото на ниво проект.
- Проектът съобразно Екстремното програмиране се провежда на малки итерации и версии. Съответно това включва модификации на общия работен процес съобразно итеративните процеси на eXPERT за изпълнение на проекта. Дейностите в рамките на итерациите изглеждат подобни на дейностите по проекта, използвани в компанията. Модификациите трябва да се прилагат по начин, който да отговаря на поддържаните документи и резултатите от итерационните задачи.
- Една от най-големите разлики в двата подхода е нивото на детайлност на данните, които се събират за проекта. Чрез eXPERT данните, които трябва да се съберат, са много по-детайлни и проследяват ежедневната работа на всеки член от екипа. Поради това разработчиците променят своя стереотип на работа и трябва да предоставят информация за постигнатото на интервали от 15 минути. Данните, събрани по време на всяка итерация, се използват за подобряване на предвижданото за всяка следваща итерация, както и цялостната зрялост на компанията.

В обобщение може да се каже, че няма да има промени в целите на процеса, а ще има значителни промени в начина, по който ще бъдат постигнати. Изисква се стриктна процедура за прилагане на промените в този процес.

- Работна среда;

В резултат на спецификите на eXPERT и изискванията за следене, измерване, контролно тестване и клиентски прегледи, трябва да се дефинира нова среда за работа на екипа.

- Промени в автоматизираните системи - Система за следене на времето, Система за следене на грешките
- Промени в стандартите за програмиране
- Нови ръководства - Ръководство за Екстремно програмиране, за Персонален софтуерен процес, за eXPERT, за процеса на фирмата

- Работни дейности в проекта;

Прилагането на нова методология със сигурност изисква модификации в работния процес на проекта, който се провежда. Дефинират се следните промени в работните дейности на проекта:

- Промени в управлението и планирането на проекта – итеративният процес, изискван от практиките на Екстремното програмиране, въвежда планирането на итерациите и версиите. Оценките, основаващи се на измерванията на Персоналния софтуерен процес (PROBE метода), предоставят база за планиране на следващите версии и итерации, основаващи се на събраните клиентски изисквания.
- Промени в управлението на изискванията – детайлната спецификация на решението е опростена до обхвата на проекта. Трябва непрекъснато да има връзка с клиента за преглед на системата. Трябва да се изработят и специфицират изискванията на клиента за всяка версия.
- Промени в анализа и дизайна – детайлната техническа спецификация на документа не е изготвена в началото на проекта. Гъвкавата практиката "Метафора на Системата" спомага при изграждане на системата в широк план в началото на проекта, а всяко изискване на клиента се анализира по време на планирането на итерациите. Проектирането се извършва съгласно изискването на клиента по време на итерацията.
- Промени в планирането на тестване и контрол на качеството – изисква се от самите разработчици да изготвят модулни тестове за модулите, които разработват. Оценката на времето за разработените модулни тестове трябва да се включи в оценката на изискванията. Това изисква извършване на тестово планиране за всяка версия.

- Роли в проекта.

Най-важната цел на този анализ е да определи разликите в дейностите и отговорностите на различните роли и предназначението им в различните фази на изпълнение на проекта. Най-важните резултати от анализа на ролите са изброени накратко:

- Клиент – при подхода eXPERT и в резултат на практиките на екстремното програмиране клиентът участва във всички фази на проекта. Следователно тази роля заема централно място в изпълнението на проекта. Спецификите за новата "клиентска" роля са модифицирани според методологията Екстремно програмиране (Beck, 1999). Най-голямата модификация на "клиентската" роля е, че според Екстремно програмиране клиентът трябва винаги да присъства на място и да е посветен на проекта.
- Продажби – персоналът по продажби трябва да информира клиента за използваната методология. Това е единствената модификация за тази роля. Отговорностите остават същите. eXPERT не се обръща към тази роля, но тъй като фирмения процес приема целия жизнен цикъл на проекта, като се започне от дейности преди продажбите, тази роля остава в последващия процес.
- Ръководител на проект (Project Manager) – поема всички отговорности на ролята "Лидер на проекта" що се отнася до eXPERT. Отговорен е и за управлението на всяка итерация, както и за плановете за итерации. Запазва отговорността за цялостното управление на проекта.
- Отговорник по качеството (QA) – ролята QA включва дейности по изготвяне и изпълняване на планове за тестване за всички итерации. QA помага на клиента да дефинира тестовете за приемане и поддържа разработчиците по време на имплементацията на тестовете. Поддръжката на контрола на качеството в процеса гарантира високо качество на продукта.
- Програмисти – влиянието от практиките на Екстремно програмиране при тях е най-силно. Промените включват програмиране по двойки, разработка на модулни тестове, използване на подробни графици и отчети за грешки.

- Системен Архитект – практиките на Екстремно програмиране “Колективно притежание на кода” и “Метафора на системата” преместват тази роля в екипа на разработка.

В резултат от извършеният анализ на различията, прилагането на гъвкава методология оказва най-голямо влияние и предполага най-много промени в основния сертифициран процес за разработка на софтуер в организацията - “Процеса за реализация”. Друг повлиян от практиките на eXPERT в значителна степен е сертифицирания процес по осигуряване на качеството. Това е в резултат от тестовите практики, които дефинира eXPERT гъвкавата методология. Относно несертифицираните процеси, като анализ и проектиране, промените са по-големи в сравнение с несертифицираните. Това се дължи на факта, че несертифицираните фирмени процеси до голяма степен са обвързани със самите дейности в процеса на реализация на конкретен софтуерен продукт, които се покриват и от избраната гъвкава методология. Промените отново са свързани с прилаганите конкретни практики, препоръчвани от eXPERT методологията – клиент при екипа, метафора на системата, рефакторинг и прост дизайн.

2.3.2.2. Прилагане на гъвкави практики от студенски екипи

С цел проучване и придобиване на знание за влиянието на различни фактори при внедряването на гъвкавите методологии, са проведени няколко емпирични изследвания като е използван подхода на казуса (case study).

До голяма степен внедряването на гъвкави методологии оказват влияние фактори, които са налични в конкретен проект, както и в конкретна организация. Предлага се факторите на средата да бъдат разделени в следните групи:

- Проектни
- Личностни
- Организационни

За изследване на влиянието на различни фактори от тези групи е направено проучване в три екипа от студенти, в които се внедрява гъвкав метод. Гъвкавият метод е комбинация от Екстремно програмиране и Скрум. Практиките, които се очаква да се следват са:

- От Скрум – Таблица на продукта (Product Backlog), Спринт (Sprint), Среща за планиране на спринт (Planning Meeting), График на спринт (Sprint Backlog), Преглед на спринт (Sprint Review Meeting) и Отзвук на спринт (Sprint Retrospective Meeting).
- От Екстремното програмиране – Програмиране по двойки (Pair Programming), Предварително писане на тестове (Test-Driven Development), Непрекъсната интеграция (Continuous Integration), Общи стандарти за писане на код (Coding Standards), Споделена отговорност за кода (Collective Ownership), Минимална архитектура (Simple Design) и Подобряване качеството на кода (Refactoring).

Отделните екипи работят в подобни условия – еднакъв размер на екипа и продължителност на проекта, еднаква методолгиия и еднакви роли, еднакво знание за гъвкавите методологии.

Сравнението между отделните екипи е извършено по следните фактори (таблица 7):

- Тип на проекта
- Разпределение на професионалните умение между членовете от екипа
- Професионализъм

- Установени връзки между членовете на екипа
- Ентусиазъм и заинтересованост на членовете на екипа

Таблица 7 Сравнението между отделните екипи

	Екип 1	**Екип2**	**Екип3**
Проект	Труден	Лесен	технологично сложен
Умения	еднакви	относително еднакви	различни
Професионализъм	относително висок	среден до относително нисък	от относително нисък до висок
Установени връзки	да	не	не
Отношение	ентузиазирано	заинтересовано	индеферентно (с изключение на 1)

При анализа на резултатите бяха открити зависимости на тези фактори към някои от практиките (Споделена отговорност за кода) и принципите на гъвкавите методи Скрум и Екстремно програмиране, които са по-детайлно представени в (XIV).

За екипи, които нямат опит и задълбочени знания в областта на гъвкавите методологии се установяват проблеми при прилагане на гъвкави практики. По-опитните и ентусиазирани членове на екипи в определени моменти не спазват гъвкавите принципи, а предпочитат изпитаните традиционни методи, като по този начин очакват, че ще завършат проекта в срок и качествено. В резултат от проведеното изследване възникнаха редица въпроси. Например

‘Дали амбицията и всеотдайността са невъзможни в Скрум екипа?, ‘Дали амбицията трябва да бъде изкоренена от разработващите екипи за да е успешно възприемането на Скрум? Много често клиентите искат бързи частични решения за сметка на неща като качество, добър интерфейс или функционалност. Как може да бъзе мотивиран разработчка и да даде всичко от себе си в проекти като този? Възможно ли е да се избере амбициозен и всеотдаен клиент? Възможно ли е винаги да имаме технически образован клиент?

Всички тези въпроси паказват важността и необходимостта от предложената в точка 3.2.3 процедура за внедравяне. Необходими са ясни и конкретни указания за въвеждане на гъвкавите практики в организаците, включително ръководства, обучение.

2.3.3. Избор на методология

Съществуват много гъвкави методологии, които решават определени проблеми свързани с разработката на софтуер. Всяка от тях има различни характеристики и области на приложение. Проблемът е да се намери най-подходящата от тях. Как да се изберат гъвкави практики, които са най-подходящи за приложение в конкретен проект, е важен научноизследователски въпрос. Изборът зависи от много фактори и в следващите подточки ще се представят различните подходи, които са предложени с активното участие на автора.

На по-високо ниво на абстракция първо се разглежда избор на цялостни методологии, а след това на по-ниско ниво на абстракции се изследва избора на гъвкави практики, които да формират нова методология.

2.3.3.1. Матрица за сравнение на методологии

Изборът на гъвкав подход през 2002г е направен според текущите условия в конкретна организация и конкретен проект (или множество проекти, които се разработват в компанията).

С цел подпомагане взимането на такова решение и извършването на подходящ избор на гъвкава методология в една организация се предлага съставяне на матрица за сравнение на гъвкавите методологии според определени проблеми, рискове, негативни фактори и обстоятелства, съществуващи в организацията (таблица 8). За всеки един проблемен показател се поставя съответствие – покриван или непокриван от всяка една методология:

Проблем	Гъвкава методология
Проблем на проекта, риск, фактор за неуспех	Как методологията адресира този проблем и/или спомага за разрешаването му и минимизиране на негативните последствия

Таблица 8 Матрица за сравнение на методологии според проблемите

Тази матрица не представлява универсална система за сравнение на гъвкавите методологии, а се съставя според нуждите на компанията и специфичните й проблеми. Това е първата стъпка на автора към адресиране на проблема с избор на подходяща гъвкава методология.

Примерна матрица за конкретната организация е представена в таблица 8. Проблемите и рисковете са групирани в зависимост от три фази от изпълнението на проекта – започване на проект, реализация и приключване на проект. Ръководителят на проекта може да използва тази таблица като ръководство за избиране на най-подходяща гъвкава методология.

По този начин, на базата на количествена оценка, се класират методологиите. Тази, която е на първо място, с най-много покривани проблематични аспекти е най-вероятният кандидат за избор при прилагането й в следващ фирмен проект. На базата на представената матрица, като най-подходящата гъвкава методология в организацията за прилагане за конкретния проект е избран метода eXPERT.

	Гъвкава методологии					
Проблем	XP	SCRUM	FDD	ASD	DSDM	eXPERT
Започване на проект						
Неясни цели на проекта	Покрива се от практиката клиент при екипа	Покрива се от практиките “Списък с изисквания за продукта” и “Списък с изисквани за итерация	Не се покрива	Не се покрива	Провежда се “Бизнес проучване” и “Проучване за състоятелност” на проекта	Покрива се от практиката клиент при екипа
Липса на компетентност в областта	Очаква се голямата част от екипа да бъдат компетентни, а чрез практиката “Програмиране по двойки” да бъдат обучени и останалите	Дефиниран основно за опитни разработчици, като задачите се разпределят в рамките на спринта	Не се обръща внимание на подбора на екипа и тяхната компетентност, но се дефинират “Експерти в областта” като роля	ASD насърчава комуникацията и обучението в екипа	Покрива се по време на фазата “Бизнес проучване”	Очаква се голямата част от екипа да бъдат компетентни, а чрез практиката “Програмиране по двойки” да бъдат обучени и останалите
Нереалистичен график	Покрива се частично при планиране на итерация	Покрива се от практиките “Списък с изисквания за продукта”, който се поддържа актуален	Количеството функционалност се избира според времевите ограничения	Обръща се внимание на наложените ограничени и избора на функционалност според тях	Отчитат се всички ограничения към проекта при дефинирането на всеки прототип	Покрива се изцяло е помощта на практиката планиране и методологията за оценяване PROBE
Реализация						
Непълна спецификация	Основната идея на XP е да се справя с проблема. Покрива се от практиката клиент при екипа	Изисква клиента да присъства при планирането на итерация	Използването на “Експерти от областта” помагат при изчистването на детайлите	Чрез фазата “Научаване” се допълват пропуските след всяка итерация	Задължително участие на клиента през всички фази на проекта	Покрива се от практиката клиент при екипа
Промени в изискванията	Една от основните черти на гъвкавите методологии	Една от основните черти на гъвкавите методологии	Една от основните черти на гъвкавите методологии	Една от основните черти на гъвкавите методологии	Една от основните черти на гъвкавите методологии	Една от основните черти на гъвкавите методологии
Неподходящи	Очаква се екипът да	Очаква се екипът да	Не дефинира	Не дефинира	Не дефинира	Очаква се екипът да

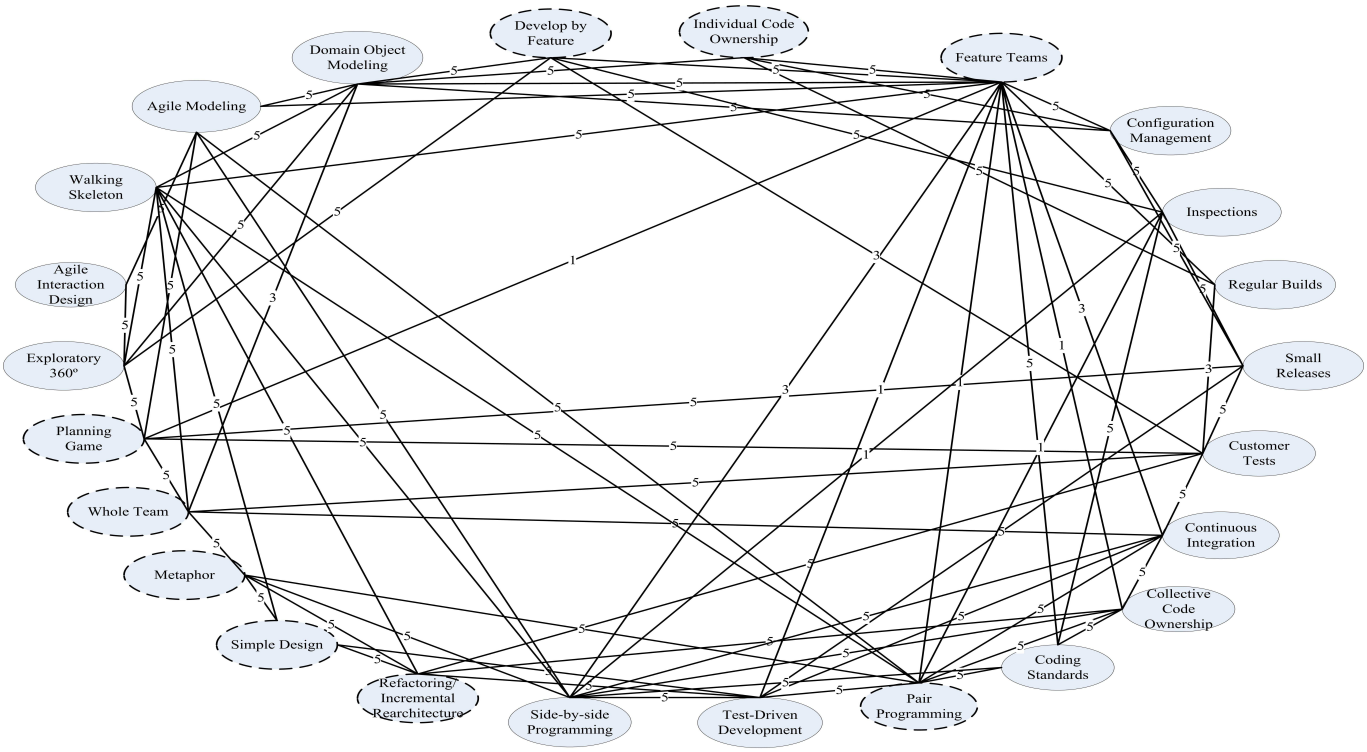
инструменти	дефинира инструментите	дефинира инструментите	ползването на инструменти	ползването на инструменти	ползването на инструменти	дефинира инструментите
Трудности при интеграцията	Ежедневна интеграция на системата	Всеки спринт реализира интегрирана система	Не дефинира процедура по интеграция	Не дефинира процедура по интеграция	Насърчава изграждането на интегрирана система след всяка итерация	Ежедневна интеграция на системата
Затруднено следене на напредъка	Клиентът може да следи постоянно напредъка по проекта	Ежедневно се следи напредъка по време на ежедневните срещи на екипа	Напредъка се измерва според реализирани функционалности	Не дефинира подобрение относно следенето на проектите.	Напредъка се измерва според реализирани функционалност	Клиента може да следи постоянно напредъка по проекта
Лоша комуникация в екипа	Малки е екипи и отворено работно пространство	Ежедневни срещи за следене на напредъка подобряват комуникацията в екипа	Използването на “Експерти от областта” помагат при изчистването на детайлите	ASD основно набляга на комуникацията в екипа	Съществува като изискване, но не предлага практики	Малки е екипи и отворено работно пространство
Лоша комуникация с клиента	Основната идея на XP е да се справя с проблема. Покрива се от практиката клиент при екипа	Изисква клиента да присъства при планирането на итерация	Не дефинира практики за комуникация с клиента, освен заложените в манифеста	Задължително присъствие на клиента при прегледа на качеството	Задължително участие на клиента през всички фази на проекта	Покрива се от практиката клиент при екипа
Липса на критерий за завършване на проекта	Според приоритизираните изисквания от клиента и разбиването им по версии се определя завършването на всяка версия	Според “Списък с изисквания за продукта” и ограниченията клиента определя пускането на версии	Дефинира се пълен списък с изброените функционалности, който се поддържа	Фаза за окончателен качествен анализ и преглед от клиента	Задължително участие на клиента през всички фази на проекта. Допълнително е дефинирана фаза с цел внедряване на системата	Според приоритизираните изисквания от клиента и разбиването им по версии се определя завършването на всяка версия
Липса на мотивация	Програмиране по двойки и 40 часова работна седмица се определят, като фактори решаващи	Не дефинира практики за повишаване на мотивацията	Не дефинира практики за повишаване на мотивацията	Не дефинира практики за повишаване на мотивацията	Не дефинира практики за повишаване на мотивацията	Програмиране по двойки и 40 часова работна седмица се определят, като фактори решаващи

	проблема					проблема
Голямо количество документация	XP минимизира документацията до възможният минимум	Не дефинира детайлна документация използвана в проекта	Обхвата на документацията се определя от мениджъра на проекта	Не дефинира детайлна документация използвана в проекта. Залага на взаимодействие	Обхвата на документацията се определя от мениджъра на проекта	Олекотена документация
Разпределени екипи	Не се препоръчва	Не се препоръчва	Не са дефинирани	Според ASD е напълно приемливо да функционират няколко екипа	Съществуват разработки с разпределени екипи	Не се препоръчва
Приключване на проекта						
Проблеми при валидация от клиента	Покрива се от практиката клиент при екипа	Покрива се от практиките “Списък с изисквания за продукта” и участие на клиента	Не дефинира практики за валидация	Фаза за окончателен качествен анализ и преглед от клиента	Дефинирана е фаза с цел внедряване, която също е итеративна	Покрива се от практиката клиент при екипа
Нестабилна версия	Дефинира практики за тестване и рефакторинг	Не се адресира	Дефинира инспекции на дизайна, кода и модулни тестове	Отчасти се покрива. Дефинира инспекции на кода	Постоянно подобрение на системата	Дефинира практики за тестване и рефакторинг
Липсваща функционалност	Покрива се от практиката клиент при екипа.	Покрива се от практиките “Списък с изисквания за продукта” и участие на клиента	Дефинира се пълен списък с изброените функционалности, който се поддържа	Фаза за окончателен качествен анализ и преглед от клиента	Задължително участие на клиента през всички фази на проекта	Покрива се от практиката клиент при екипа
Предпоставка за нов проект	Не се адресира	Не се адресира	Не се адресира	Не се адресира	Не се адресира	Не се адресира

Таблица 9 Матрица за сравнение на методологии според проблемите за конкретната организация

2.3.3.2. Рамка, подпомагаща избора на гъвкави практики

При създаването на нови методологии, представени в секция 2.3.1 обикновено се комбинират практики от различни съществуващи разпространени гъвкави методи. Следващият логичен въпрос, на който авторът търси отговор е какви са зависимостите между практиките от различните методологии и каква е приложимостта на дадена съвкупност от гъвкави практики. Това е друг аспект на избора на методология, допълващ аспекта на външните фактори. В резултат на проучвания и анализ на гъвкавите практики от разпространени гъвкави методологии в (X) се предлага рамка за оценка съвместимостта и противоречивостта между множество от гъвкави практики. Имат се предвид практики, които дават практически насоки за активностите от жизнения цикъл на проект, а не практики за управление. Поради тази причина са подбрани гъвкави методологии, които дефинират практики, подпомагащи същинското разработване на софтуер – Екстремно програмиране, РФ (FDD) и Кристъл (Crystal Clear). Добавен е и подхода Гъвкаво моделиране (Agile Modeling), който не е цялостен процес, но е интересно да се оцени приложимостта на гъвкавото моделиране (тъй като моделирането в областта на софтуерните технологии и широко застъпено по принцип) с другите гъвкави практики. Изследването се промежда на няколко стъпки. Първо се идентифицират дефинираните практики от избраните гъвкави методологии и се прави таблица на съответствието спрямо следните фази от жизнения цикъл – фаза проучване (feasibility study), спецификация на изискванията, проектиране, кодиране, тестване на функционална единица, тестване на система, тестване за приемане на системата и интеграция. Като втора стъпка се проучват вътрешните зависимости между практиките – първо в една методология, след това последователно зависимостите между практиките на Екстремното програмиране и РФ (FDD). На следваща стъпка към тях се добавят и зависимостите от Кристъл (Crystal Clear) и накрая се идентифицира кои практики могат да имат полза от гъвкаво моделиране. Като последна стъпка се конструира граф със съответните тегла, който представя всичките идентифицирани гъвкави практики и зависимостите между тях (фигура 16)



Фигура 16 Граф на зависимостите между гъвкавите практики

Накрая се изследва влиянието на външни фактори върху приложимостта на дадена практика. След проучване на литературата са класифицирани следните три групи фактори:

- *социални фактори* – свързани са с хората и техните взаимодействия
- *бизнес фактори* – свързани са с характеристиките на проекта и бизнес средата
- *технологични фактори* – показват до каква степен е необходима техническа помощ за прилагането на даден практика

Пример за идентифицираното влияние на социалнита фактори е представен в таблица 10 от (X) .

		Ниска ангажираност на хората	Ниска индивидуална компетентност	Висока специализация	Ниска степен на сътрудничество
	Целият екип	-1			-1
	Игра на планиране	-1	-1		-1
	Клиентски тестове	-1			-1
	Минимална архитектура				
	Програмиране по двойки	-1	-1	-1	-1

	Предварително писане на тестове			-1	
	Подобряване качеството на кода		-1	-1	
	Метафора			-1	
	Моделиране на обектите от областта	2	-1	4	
	Разработване по функционалност	2			
	Индивидуална клас принадлежност				4
	Функционални екипи	2	2	2	2
	Регулярна интеграция				
	Управление на конфигурациите				2
	инспектиране	2			2
	Проучване на 360°	2	-1		-1
	Постепенно подобрение на архитектурата.		-1		
	Гъвкав проект на взаимодействие	2	-1		-1
	Програмиране рамо-до-рамо	2	-1	-1	-1
	Гъвкаво моделиране	-1	-1	1	-1

Таблица 10 Влияние на социалните фактори върху гъвкавите практики

2.3.3.3. Подход за избор на гъвкави практики

След като бяха идентифицирани вътрешните зависимости на гъвкавите практики, през периода 2007–2010г. изследванията се фокусират върху влиянието на външни фактори избора на гъвкави практики.

Както вече беше отбелязано в предишната секция приложимостта на гъвкавите практики зависи силно от характеристиките на проектите. Дадена практика може да бъде приложена в един контекст, докато в друг не може. Поради тази причина се предлага подход за създаване и поддържане на база от знания, в която се съхранява информация за приложимостта на дадена гъвкава практика в дадена проектна ситуация. Предлага се структура на базата от знания, както и различни механизми за анализирането ѝ. Допълнително се предлага ръководство за избор на подходящи гъвкави практики за даден проект.

Изграждане и структура на базата от знание

Понастоящем няма стандартна таксономия на проекти в софтуерната индустри, която може да се използва за идентифициране и категоризиране на проекти на базата на общи фактори. Редица изследвания са публикувани, но няма консистентен и цялостен избор от фактори, които да опишат една проектна ситуация. Джоунс (Jones, 2000) идентифицира 36 важни фактора, които са документирани от световната организация Software Productivity Research (SPR) и които се използват за сравнителни анализи на проекти. В тази класификация обаче липсват някои важни фактори, които се отнасят до гъвкавите

методологии, а има и такива които не са толкова важни. Ето защо този списък от фактори се актуализира с фактори, които оказват влияние на използването на гъвкави методологии по следната процедура:

- Идентифициране на изследванията, които имат отношение към различни ситуационни фактори, оказващи влияние на внедряването на гъвкавите методи
- Създаване на списък от всички идентифицирани фактори и определяне на скали за факторите и множествата от възможни стойности за скалите
- Минимизиране на множеството чрез отстраняване на повтарящи се фактори и фактори с близко значение. Обединение на стойностите от скалите на факторите, където е необходимо.
- Допълване на списъка от фактори с фактори, идентифицирани от Джоунс (Jones, 2000)
- Класифициране на факторите в две основни категории – Тип на проекта и Контекст.
- Определяне на подходящи под-категории и групиране на факторите от категориите в тях.

Процедурата, описана по-горе е приложена в общо 6 научни разработки (Klooster, Brinkkemper, Harmsen, & Wijers, 1997) (Qumer & Henderson-Sellers, 2009) (Kruchten, 2008) (Turk, France, & Rumpe, 2005) (Cockburn & Highsmith, Agile software development, the people factor, 2001) (Misra, Kumar, & Kumar, 2009). В резултат на процедурата са идентифицирани 47 ситуационни фактора като характеристики на проектна ситуация, от които осем попадат в категорията Контекст, а останалите 39 – в категорията Тип на проекта.

След като вече са определени факторите се преминава към проектиране на базата знания, която да поддържа анализа на приложимостта та гъвкавите практики според:

· теоретична неприложимост на гъвкавите практики

· доклади, описващи опита на други организации за прилагането на гъвкави практики

· личен опит на организацията

Теоретичната неприложимост на гъвкавите практики е базирана на характеристиките на практиките, на чиято основа са направени предположения за приложимостта на дадена практика при наличието на определени ситуационни фактори.

Направените заключения са обосновани чрез преглед и анализ на различни литературни източници (Boehm B. , 2002) (Cockburn & Highsmith, Agile software development, the people factor, 2001) (Abrahamsson, Salo, Ronkainen, & Warsta, 2002). Пълният анализ е описан в (Krasteva & Ilieva, 2007). За измерване на теоретичната неприложимост на гъвкавите практики е въведена 4-степенна скала {0,1,3,5}, където със стойност ‘5’ се измерва най-високото ниво на неприложимост на практика. Ако определен фактор не пречи или способства за прилагането на дадена практика се обозначава с нулева стойност.

Независимо колко задълбочен и систематичен е теоретичния анализ, предположенията извлечени от него могат да се валидират чрез следващи емпирични изследвания. Съществуват редица доклади, които представят как отделни гъвкави практики се възприемат в някои проекти.

Базата от знания съдържа също информация за приложимостта на гъвкавите практики в предишни проекти на организацията. Това е най-надеждната информация, обаче не е налична при първо прилагане на гъвкави практики.

Информацията от различните доклади, описващи приложни изследвания за прилагането на гъвкавите практики, и личния опит от прилагането на практиките се систематизира в базата от знания по следната процедура:

Определя се типа на записа – ‘Публикуван доклад’ или ‘Опит от реална експлоатация’.

Създава се нов запис в базата с описание на източника - библиографска информация за доклада, състояща се от името на доклада, авторите и годината на публикуване. Ако предоставената информация не е публикувана, се въвежда само името на човека, предоставящ информацията.

Въвеждат се име на проекта и организацията, в която е реализиран. Тази информация е задължителна при записите, описващи реална експлоатация. Точната информация е много важна, тъй като тези две полета се използват за да се определи уникалността на записа. За публикуваните доклади името на проекта и организацията се използва заедно с библиографските данни за да се разграничат проектите, които са описани в един и същи източник. Ако не е налична точна информацията за името на проекта се използва индекс например Проект_1 и Организация_1 за стойности на името на проекта и организацията съответно.

Ако запис от тип ‘публикуван доклад’ със същата библиографска информация и име на проекта и организация не съществува в базата, се продължава със стъпка 5. В противен случай се извежда информацията за съществуващия запис в базата.

Ако запис от тип ‘опит от реална експлоатация’ със същото име на проект и организация не съществува в базата се преминава към стъпка 6. В противен случай се извежда информацията за съществуващия запис в базата.

Характеристиките на проекта се описват чрез набор от дефинираните по-горе ситуационни фактори. Ако стойността на определен ситуационен фактор не може да бъде определена от наличната в доклада информация, то този ситуационен фактор не се включва в описанието на ситуацията на проекта..

Успешното използване на дадена гъвкава практика в проекта се описва в базата чрез 5 степенна скала на приложимост {N/A, 0, 1, 3, 5}, в която с положителните цифри 1,3 или 5 се отразява степенна на прилагане на практиката в проекта, стойността ‘0’ показва, че практиката не е използвана в проекта или не е била приложена успешно, а с N/A се означава, че не е налична информация за приложението на практиката в проекта.

За описание на гъвкавите практики в проекта се използва предварително дефинирано множество от стандартни гъвкави практики. Ако в даден проект се използва адаптирана практика, това се отразява чрез скалата на приложимост. Частичното прилагане на дадена практика също се отразява чрез степените на скалата на приложимост.

Стойностите на изброимия тип Скала на приложимост на практика от структурата на базата от знания се описват със скалата на приложимост - {N/A, 0, 1, 3, 5}. Стойностите на изброимите типове Гъвкави практики и Гъвкави методи се взимат от стандартните гъвкави практики и методи, описани в Приложение 2. Изброимият тип Тип на източник има две стойности ‘Публикуван доклад’ или ‘Опит от реална експлоатация’.

Процес на избор на гъвкава практика

Изборът на гъвкава практика, която да се използва в проект е итеративна процедура и се прилагат няколко различни анализа на данните в базата от знания.

Преди да започне процедурата по избор се определят гъвкавите практики, които да се анализират. След това се определя спецификата на проекта чрез множество ситуационни фактори. На следващата стъпка се извършват анализите.

След като са определени проектите с подобни проектни ситуации изборът на гъвкави практики за конкретен проект става на база на изпълнението на два типа анализ – качествен и количествен.

Качественият анализ има за цел да представи в систематизиран вид информацията за приложението на гъвкавите практики в проекти с подобни проектни ситуации. Предоставената информация се анализира допълнително от инженера по конструиране на методи за да се изберат най-подходящите за прилагане в конкретния проект набори от практики. В синтезиран вид е представена информация за прилагането на всяка една гъвкава практика в подобните на целевия проект ситуации. Анализът съдържа:

- общ брой от базови проекти, в които определена практика е приложена в най-голяма степен
- общ брой от базови проекти, в които определена практика е приложена в средна степен
- общ брой от базови проекти, в които определена практика е приложена в най-ниска степен
- общ брой от базови проекти, в които определена практика не е приложена или е приложена неуспешно
- общ брой от базови проекти, в които няма информация за приложението на определена практика.

Предложеният метод за *количествен анализ* е клъстерния анализ. Анализът се извършва върху подобните на базовия проект проекти чрез класификация на клъстери от практики.

2.4. Заключение

Ползите от постигнатите резултати в областта на гъвкавите методологии могат да се обобщят по следния начин:

Предизвикателствата пред приложението на гъвкавите методологии в софтуерните проекти показват необходимостта от създаването на малко и качествени методологии, които удовлетворяват множество ситуации на приложение. Бъдещото развитие на изследванията в областта следва две

основни направления. Първото направление е използването на знание под формата на предходен опит. Второто направление е съобразяването на характерните особености на отделните проекти при създаването на метода на разработка.

Анализирането на разликите между сертифицираният процес в организацията и определена гъвкава методология представлява огромен интерес от страна на управлението на организациите и улеснява внедряването на нетрадиционни гъвкави практики в организациите. Дефинирането на последователност от стъпки за прилагане на гъвкавите методологии и извличането на ползи, както от гъвкавите, така и от утвърдените управленчески практики допринася за по-ефективното функциониране на една организация и повишава качеството на предлагания продукт.

Прилагането на eXPERT, като гъвкава методология дава положителни резултати и е насочен към проблематичните области на компанията, а именно:

- Успешно реализиране на проекти, при които желаните резултатите не са напълно точно и ясно дефинирани от самото начало на проекта;
- Успешно реализиране на проекти с кратък срок на реализация;
- Подобряване на оценяването на количеството работа според изискванията;

Като измерими резултати, прилагането на гъвкав подход намалява усилието за реализация на системата, което редуцира и фирмените разходи. Повишава се производителността на програмистите с над 40 %, което оказва влияние върху скоростта на разработка. Повишава се качеството на реализирания продукт с над 10 %, което допринася за повишена удовлетвореност на клиента.

Подобрената комуникация с клиента и намаленият брой подържани документи за всяка фаза от проекта определено оказват положително влияние върху усилията за разработка и минимализирането на разходите (с над 10 %). Това не само намалява броя на необходимите експерти, които създават спецификациите на клиентските изисквания, но и намалява усилията необходими по осигуряването на качеството и намалява времето за изпълнение на приемните тестове.

Основните резултати са дефинирани препоръки и стъпки за прилагането на eXPERT методологията и в други компании, които биха желали да променят своя процес на работа. Това е извършено на базата на представения анализ на различията, натрупаният практически опит и анализа на резултатите от експерименталното прилагане на eXPERT в конкретна софтуерна организация.

III. Тестване на софтуерни системи

Настоящата глава е посветена на тестването на софтуерни системи. Авторът работи активно в това направление от 2007 г насам като през този период са публикувани 15 научни статии (9 от тях са представени за участие в конкурса – IV, XII, XVI, XIX, XX, XXII, XXIV, XXV, XXVI), на които са намерени 17 цитирания по непълна библиографска справка. Под ръководство на автора са защитени успешно 14 магистърски дипломни работи, а един докторант е отчислен с право на защита. По тази тематика авторът е ръководител на един национален и участник в един европейски научно изследователски проекта, а именно:

- проект ДО-02-182/2008 “Архитектура на висококачествен софтуер на базата на шина за услуги” с Фонд "Научни изследвания", Министерство на образованието и науката, 2008-2012
- FP7-REGPOT-2007 SISTER - Strengthening the IST Research Capacity of Sofia University - **(**Укрепване на научния капацитет на Софийски университет в областта на технологиите на информационното общество**)** с Европейската комисия, 7-ма Рамкова програма, 2008-2011

Качеството на софтуерните системи е в основата (Pressman, 2005) на софтуерните технологии. Тестването е един от начините за осигуряване на качеството на софтуерните системи. От друга страна тестването е важна дейност от цялостния процес на разработване на софтуер. Тестването на съвременните софтуерни системи е затруднено, а в същото време очакванията към качеството им са високи, тъй като те засягат все повече аспекти на живота ни.

3.1. Обзор на предметната област

През 2007 в (XIII) са идентифицирани проблемите и потенциални решения при тестване на SOA. Изследват се проблемите и решенията на различни етапи от разработването на софтуер – тестване на функционална единица, тестване на интегрирани функционални единици и тестване на системата. Тук ще дадем пример само с тестването на функционална единица, а останалите 2 етапа са представени в (XIII).

Тестване на функционална единица

Тестването на функционална единица е критична активност при осигуряване на функционална съвместимост на софтуерни приложения, базрани на услуги. В значителна степен тестване на функционална единица е подобно на тестване на отделни компоненти. Тестовете трябва да се изграждат и изпълняват на базата на описанията на софтуерните услуги и ако е възможно всеки тип тестова техника може да се прилага.

Предизвикателства

Следните характеристики на софтуерните услуги, участващи в SOA въвеждат специфични предизвикателства при тестване:

• Софтуерните услуги нямат потребителски интерфейс. This means that QA team should have good development skills to create appropriate test harnesses that provide test data to the tested objects.
• Разработчиците на услуги, които ще се използват от други компоненти обикновено имат достъп само до интерфейсите и липсва достъп до кода на услугите.

Решение

Тестването на функционална единица е много подобно на тестването на нормални софтуерни компоненти, то в областта има вече свършена значителна изслователска работа и редица работещи решения съществуват. Описанието на софтуерната услуга може да се използва за автоматично генериране на тестови случаи за функционално тестване "Черна кутия". Техники като например генетични алгоритми също могат да бъдат използвани, стартирайки от WSDL описания или, ако има от семантични анотации [5]. Всъщност тази област се покрива от предлагани на пазара автоматични инструменти. Почти всички автоматизирани SOA тестови инструмента всъщност извършват тестване на функционална единица. Те могат да четат описанията на услугите и да създават тестове, на базата на тези описания. Освен това тези инструменти осигуряват подходящи потребителски интерфейси за създаване на тестове, организирането им в позитивни или негативни, и попълване на тестовете с подходящи тестови данни. В този смисъл проблемите с тестване на услуги на ниво функционална единица могат да се решат лесно и бързо чрез използване на готови инструменти, но единственото, което отстава е да се избере автоматизиран тестов инструмент, който най-добре съответства на нуждите на проекта. the project needs.

Водещите фирми в тази посока са IBM и Mercury заедно с IBM Rational Tester for SOA Quality и Mercury Service Test tools, които позволяват всеки екип по тестването да се завършват функциално и тестване за производителност на услугите. Други полезни инструмента са Parasoft's SOAtest, and ITKO's Lisa. Разбира се това не означава, че само търговските инструменти са най-добрите – съществуват редица свободни инструменти като TestMaker, WS Chess, SoapUI, и друго. http://www.infoworld.com/article/07/05/11/19TCwebservicetest_3.html

След като в края на 2008 вече има финансиране по национални проект изследванията се разширяват и задълбочават. Вече не се мисли за отделни частични решения, а за цялостен подход – тестова рамка.

Темата за осигуряване на качеството на софтуерните системи привлича особен интерес сред академичните среди и изследователските центрове на големите индустриални фирми. Потвърждение на това са огромният брой подадени за публикуване доклади в областта на осигуряване на качеството на софтуерните системи на най-престижната международна конференция по софтуерно инженерство ICSE (International Conference on Software Engineering). Според представените статистики през последните 3-4 години най-много статии има именно в тази област, а се появяват редица специализирани конференции и форуми по проблемите на осигуряване на качеството на софтуерните системи.

Тази тематика е особено актуална предвид навлизане на софтуерните системи във всички области от живота на хората и обществото. Също така тенденцията е към разработване на все по-сложни системи на базата на интегриране на съществуващи в т.нар. системи от системи. Проблеми в качеството на една система трудно могат да останат изолирани, а и локализирането им е затруднено. Евентуална некачествена работа на софтуерните системи би имала сериозни последици както за потребителите, така и за други свързани софтуерни системи. **Ето защо наличието на цялоста тестова среда би спомогнало за ефективно разработването на качествен софтуер**.

3.2. Мотивация за изследването

3.2.1. Сложност на SOA инфраструктурата

В последните години се наблюдава широкото навлизане на архитектури, ориентирани към услуги (Service-oriented Architectures SOA) в ИТ инфраструктурата на големите предприятия. Глобализацията на съвременните организации води до разрастването и окрупняването на предприятията, което от своя страна води до наличието на голям брой *независими* една от друга информационни системи, въведени в експлоатация по различно време от различни доставчици, базирани на различни технологии, които се предполага да работят съвместно. Неминуемо в тази ситуация определени абстрактни идеи и обекти се дублират в няколко системи. Например, повечето системи имат собствен обект и представа за "потребител", който представлява отражението в рамките на системата на лицето, клиент на някоя услуга или продукт. Когато предприятието се сдобие с нов клиент, се налага извършването на обмен на данни (ръчен или автоматичен) за потребителя между всички системи, които имат нужда от това. Това усложнява връзките между системите и често се стига до ситуация, в която всяка система е свързана с всяка една друга и то по няколко различни интерфейса (напр. за обмен на потребители, за обмен на тарифни планове, за обмен на финансова информация и т.н.), при това понякога се налага и двупосочен обмен. Следователно, при така описаната ситуация инфраструктурата за обмен на данните има квадратична асимптотична сложност

$$O(kn(n-1)), \tag{1}$$

където k е средния брой връзки между две системи, а n е броят на отделните информационни системи. Това означава, че добавянето на нова система в предприятието ще доведе до

$$\frac{kn(n-1)}{k(n-1)(n-2)} = \frac{n}{n-2} = 1 + \frac{2}{n-2}, \tag{2}$$

пъти абсолютно увеличаване на сложността на инфраструктурата. Вторият член от дясната страна на равенството представлява относителното увеличаване на сложността на инфраструктурата, което ще отбележим с X:

$$X = \frac{2}{n-2} \tag{3}$$

Например, ако към настоящият момент предприятието разполага с три информационни системи и се въведе четвърта, то инфраструктурата ще се усложни със 2/(4-2) = 100%. Ако след това се въведе пета система, то инфраструктурата ще се усложни с нови 2/(5-2) = 67% или общо със 234% спрямо началната ситуация. Така, с въвеждането само на две нови системи, инфраструктурата за обмен на данните се усложнява трикратно. Това води до увеличени разходи за внедряване, експлоатация и повишаване на риска от грешки при обмена на данни.

За решаване на проблема с квадратичната сложност на връзките между различните системи се използва комбинация между добре познати архитектурени похвати, между които:

- скриване на знанието и предоставяне на добре познати *интерфейси* – детайлите относно работата на даден "модул" са скрити от всички останали "модули"; достъпът до функционалността се осъществява *само* посредством документирани интерфейси. По този начина, дори да се промени реализацията на функционалността, останалите "модули" не трябва да се променят;
- използването на *медиатор* (посредник) за обмен на информация – когато даден "модул" А трябва да предаде информация на друг "модул" Б, той го предава вместо това на специален медиатор "М"; вторият "модул" Б взема информацията от медиатора. Така, вместо един канал за обмен на информация А → Б, се изграждат два канала за обмен на информация: А → М и Б ← М. Така топологията се променя от "всеки със всеки" към "звезда" и добавянето на нов "модул" или режим на комуникация налага само изграждането на връзките му с медиатора, а не с всички останали.
- късно свързване – отдалечаване на момента на създаване на зависимост на "модул" А от "модул" Б възможно най-късно във времето.

При използването на топология "звезда", инфраструктурата за обмен на данните има линейна асимптотична сложност

$$O(k2n) \tag{4}$$

Това означава, че добавянето на нова софтуерна система в дадена организация ще доведе до

$$\frac{k2n}{k2(n-1)} = \frac{n}{n-1} = 1 + \frac{1}{n-1} \tag{5}$$

пъти абсолютно увеличаване на сложността на инфраструктурата или до

$$X = \frac{1}{n-1} \tag{6}$$

относително увеличаване на сложността на инфраструктурата. Така, ако използваме отново същия пример, при наличието на 3 софтуерни системи и добавянето на 4-та, инфраструктурата ще се усложни с 1/3 = 33%, а при добавянето на 5-та с нови 1/4 = 25%. Общото увеличаване на сложността спрямо началната ситуация е 67%, в сравнение с над- трикратното увеличение при ситуацията с квадратична сложност.

Именно за решаване на проблемите със сложността е широкото навлизане на софтуерни приложения, базирани на SOA (архитектури, ориентирани към услуги). Те от своя страна поставят редица проблеми.

3.2.2.Проблеми

Основен проблем при софтуерни приложения, базирани на SOA, който представлява интерес, е свързан с ***осигуряване на качеството, и по-конкретно тестването им***. Осигуряването на качеството е затруднено поради следните причини:

- Основна характеристика на софтуерни приложения, базирани на SOA (SBA) е възможността отделните софтуерни услуги, от което се състои приложението, да бъдат предоставени от различни производители, като за тях *няма механизъм за контрол* и друг начин на достъп, освен обявения чрез WSDL протокол. Съответно, *липсва възможност софтуерните услуги, които не са под собствен контрол да бъдат инструментирани* по подходящ начин, така че да предоставят специфични интерфейси за целите и нуждите на осигуряването на качество.

- Не само софтуерните услуги, но и инфраструктурата, по която се осъществява достъп до тях може да бъде извън собствен контрол, тъй като Услугите в общият случай се достъпват по Интернет и дори могат физически да се намират на друг континент. В сравнение с локалната мрежа, сложната Интернет инфраструктура предполага повече звена, които могат да дефектират и следователно вероятността за комуникационен срив е по-голяма. Поради тази причина, при създаването на софтуерни приложения, базирани на SOA **е необходимо да се обърне специално внимание на поведението на системата при отпадане или влошаване на качеството на част от комуникационните канали**, което събитие е с голяма вероятност. **Функционалност за автоматизация на подобен вид тестове не се среща** при нито една от съвременните ESB (Enterprise Service Bus).

- Опитът показва, че много често част от софтуерните услуги, от които зависи софтуерната система, се разработват паралелно с нея. Ако се случи така, че някоя от софтуерните услуги, която се извиква рано в процеса на изпълнение, не е готова, това може да забави драстично тестването, тъй като последващите активности в процеса могат да зависят от резултатите от изпълнението на несъществуващата софтуерна услуга. За да се предотврати това забавяне е *необходим механизъм за временно премахване на зависимостта на софтуерната система от съответната услуга(и).*

- Базираните на услуги приложения по същество представляват разпределени хетерогенни системи. Това, че функционалността е разпределена върху различни сървърни компоненти, води до *трудности при идентифицирането на основните причини за даден срив*, тъй като налага анализирането на множество журнални файлове. Хетерогенността на системите допълнително усложнява този анализ, тъй като предполага наличието на *специфичен подход за анализ за всяка една различна платформа.*

Съгласно горното изложение се идентифицират следните основни ***конкретни проблеми, на които се предлага решение:***

1. Липса на възможност за инструментиране на софтуерни услуги, върху които няма собствен контрол
2. Липса на функционалност за автоматизация на тестване на поведението на системата в условия на отпадане или влошаване на качеството на част от комуникационните канали с отдалечени услуги
3. Забавяне на тестването поради паралелна разработка на различните компоненти на приложението въпреки наличието на зависимости между тях
4. Трудности при идентифициране на основните причини за срив и нужда от специфичен подход за анализ за всяка една различна технологична платформа

От идентифицираните проблеми, 1), 2), 3) и отчасти 4) са предмет на осигуряване на качеството по време на проектирането и създаването на системата. Проблем 4) е предмет предимно на осигуряване на качеството по време на експлоатацията на софтуерните приложения, ориентирани към услуги и неговото решаване е бъдеща задача.

3.3. Резултати

3.3.1. Тестова рамка

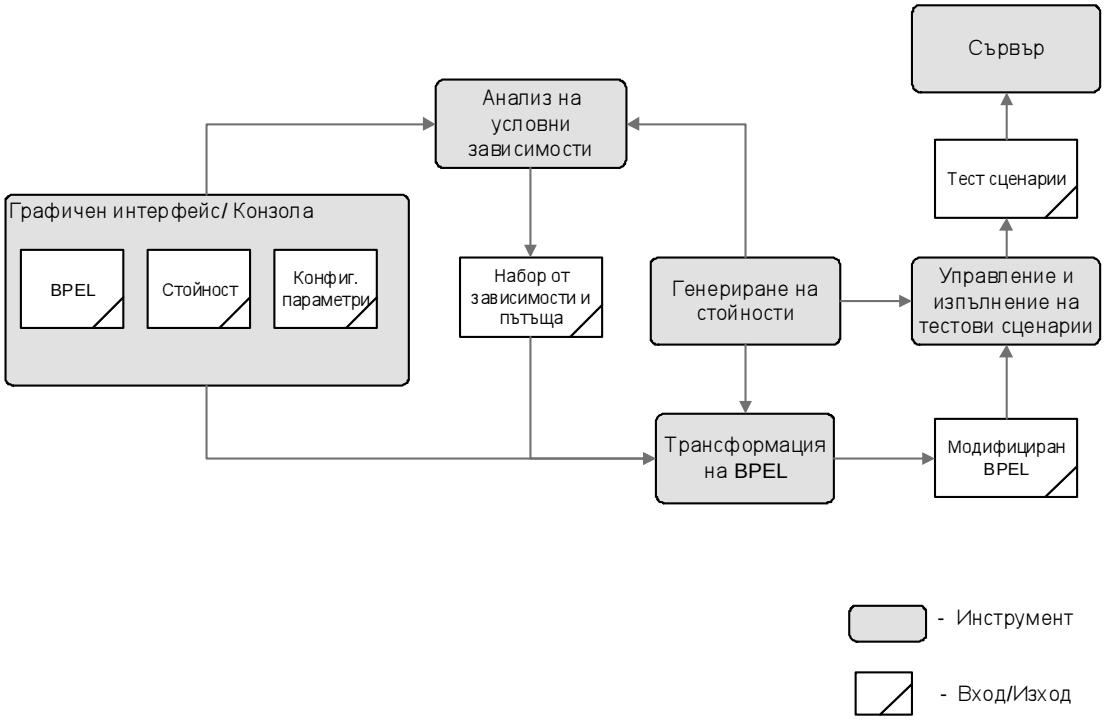
Подходът, избран с оглед решаването на основните проблеми, специфицирани в секция 2 от настоящия документ, свързани с тестването по време на проектирането е:

- да се предостави софтуерен инструмент наречен **Изолационен инструмент**, посредством който за целите на тестването да може да се премахва временно зависимостта на приложението, което се тества, от една или повече външни софтуерни услуги (върху който нямаме собствен контрол). Това ще позволи на специалистът по качеството да контролира резултатите, връщани от софтуерната услуга, като по този начин предопределя поемането на различни пътища в процеса (проблем 1), както и да продължи тестването на приложението дори при липсата на дадена софтуерна услуга (проблем 3).
- Преценката за това коя софтуерна услуга как да се държи, за да се поеме по точно определен път в процеса може да бъде доста трудоемка. Поради това, следва да се предостави инструмент **Инструмент за анализ на зависимости**, посредством който специалистът по качество да може да прецени кои точно софтуерни услуги да бъдат изолирани и какви резултати от тяхното изпълнение да се симулират, за да се тества конкретна последователност в бизнес логиката.
- Предоставя се инструмент, базиран на горните два, посредством който да се направи анализ на бизнес процеса с цел да се генерират тестови сценарии (**Инструмент за управление и изпълнение на тестови сценарии**), с които да се обходят всички пътища в процеса.
- Инструментът за управление и изпълнение на тестови сценарии се нуждае от друг софтуерен инструмент - **Инструмент за генериране на**

стойности, посредством който да се генерират стойности за входните данни (вкл. и случайни такива) за различните сценарии.

- За автоматизация на тестването на поведението на системата в условия на отпадане или влошаване на качеството на част от комуникационните канали, се предоставя инструмент (**Инжекционен инструмент**), посредством който тези явления се симулират в ESB средата, върху която е наличен собствен контрол.

Чрез софтуерните инструменти, описани до тук, се решават проблеми 1) и 3) (дефинирани в секция 3.2) и до голяма степен се постига автоматизация на функционалното тестване. Освен това се решава до голяма степен проблем 2) и може да се постигне автоматизация на негативното тестване. Проблем 4) се проявява предимно по време на експлоатация на приложението и съответно ще бъде предмет на изследване в бъдеще. *В резултат на проведените проучвания към момента не се предлага подобен цялостен подход за решаване на горните проблеми и в този смисъл предложената рамка от интегрирани инструменти е принос*.



Фигура 17 Тестова рамка

На Фигура 17 е представена обобщена структура на взаимодействието между предложените инструменти. Приложението, което подлежи на тест се представя посредством BPEL описанието на бизнес процеса, което от своя страна включва всички необходими WSDL и XSD описания на операциите, които се извикват, структурите, които се използват и съобщенията, които се обменят.

3.3.2. Тестова методология

Предложената тестова рамка може да се използва чрез следния тестов алгоритъм. В научната литература не съществува еднозначно приета дефиниция

за алгоритъм. Авторът се придържа към следното определение: **Последователности от елементарни действия, необходими за решаването на проблем или задача се наричат алгоритми** [16].

Съгласно Кнут [27] алгоритмите притежават следните свойства:

- Дискретност (изброими елементарни действия)
- крайност – алгоритъмът приключва след краен брой стъпки
- формалност – позволява автоматично изпълнение
- определеност – еднозначно изпълнение на стъпките при едни и същи данни
- масовост – един алгоритъм може да се прилага за клас задачи
- изпълнимост – състои се от изпълними стъпки
- резултатност – при какви да е входни данни довежда до резултат
- ефективност – рационално решение на задачата- възможно най- бързо, с най- малко памет .

На базата на горната дефиниция се предлага следния алгоритъм за тестване в етапа на проектирането на софтуерна система, базирана на услуги посредством предложената тестова рамка:

1. Чрез използване на Инструмент за анализ на зависимостите (ИАИ) (Covering Scenario Generation Algorithm) се дефинират набор от пътища (Path_N) и данни, свързани с тези пътища (Data_N), гарантиращи преминаването през тях. Проложената техника се категоризира като White-box. За генерацията на данни се използва инструмент TAXI. TAXI предоставя генерация на данни в два режима: Разделяне по произволен начин (Random) и Разделяне по категории (Category Partition). Така дефинираната двойка Път+Данни за изпълнението му ще наричаме тестов сценарий.
2. На базата на предварително дефинирани критерии (Criteria_N), свързани с конкретния тестов цикъл, се идентифицира подмножество от тестови сценарии. С тях трябва да валидират и верифицират поставените критерии. За целта се прилага GQM (Goal Quality Metric) парадигма за определяне на цели и метрики за измерване на резултатите. На практика всеки сложен процес, представен чрез BPEL, се декомпозира до множество от елементарни BPEL процеси (BPEL_N), които се изпълняват еднозначно (без възможности за варианти при изпълнението им като наличие на if-then-else конструкции например) За всеки елементарен процес се дефинира очакван резултат за предварително дефинираните входни данни.
3. Този набор BPEL_N+data_N се обозначава като тестови случаи N (Test_Cases_N) за конкретния тестов цикъл. Прилага се техника Черна кутия (Black-box) за тестване и оценка.
4. Изпълнението на позитивните тестови сценарии се подпомага от инструмент за Изолация на услуги. С него се симулира изпълнението на услуги, които не са готови или не са достъпни към момента на

изпълнение на тестовите активности. Използването на симулация е известен подход, новото е, че се използва механизъм за изолация/подмяна на обръщението към услуга, а не се реализира чрез подходящо генерирани stubs.

5. Изпълнението на негативните тестови сценарии, включващи:
    a. Липса на отговор от услугата;
    b. Закъснение в изпълнението на услугата;
    c. Нарушаване на структурата на съобщенията;
    d. Нарушаване верността на данните в съобщенията (за модификацията на данните също се прилага TAXI).

се постига с помощта на механизъм за инжектиране на грешки и закъснения при обръщение към услуга. Детайлите, свързани с реализацията му, са описани в съответните инструменти за тестване.

6. Изпълнение на позитивни и негативни тестове в интегрирана тестова среда
7. След всяко изпълнение на тестовият случай се сравнява получения с очаквания резултат.
8. На базата на получената информация по време на изпълнението на сценариите (log files) се дефинират шаблони за поведение на системата. Те служат за превенция или идентифициране на проблеми чрез средствата на мониторинг на системата в режим на реална експлоатация. Шаблоните за поведение се съхраняват в отворена база от знания (данни), която се обновява и разширява в процеса на експлоатация.

Предложеният алгоритъм използва добре познати тестови подходи като техники за реализация и изпълнение Бяла кутия (White-box) Покритие на пътища (Path coverage), Черна кутия (Black-box), оптимизация на тестовите сценарии на базата на Category Partition (Разделяне по категории).

Следните типове тестване могат да се покрият от тестовата рамка:

- · Функционално тестване, където са включени 3 възможни сценария:
    - Покритие на пътища (Path coverage) според функционалните зависимости – тестване на пълното покритие на всички клонове на пътища с най-късата дължина на път на бизнес процес, за верифициране, че покрива функционалните изисквания и верифициране, че очакваните резултати са получени;
    - Процес, силно зависим от данните - тестване на пълното покритие на всички клонове на пътища с най-късата дължина на път на бизнес процес с различни множества от стойности, прилагане на контролно точки и верифициране, че очакваните резултати са постигнати;
    - Покритие на пътища според нова функционалност.
- Тестване на надеждност, където тестовата рамка осигурява симулации на възможни повреди на ненадеждни софтуерни усуги и тестване на тези пътеки в бизнес процеса, който включва клонове с надеждни услуги.

- Тестване на устойчивост, където тестовата рамка осигурява симулации на възможни повреди на ненадеждни софтуерни усуги и тестване на тези пътеки в бизнес процеса, който включва клонове с ненадеждни услуги.
- Тестване за разширяване, където се правят промени в конфигурацията следвани от тестване на пълно покритие на пътища с най-късата дължина на път в процес, за да се верифицира, че ще продължи да работи нормално.

3.3.3. Софтуерни инструменти

Предложените по-горе софтуерни инструменти за тестване ще бъдат представени чрез два аспекта - *задачата*, която решават и *подхода*, който прилагат.

3.3.3.1. Изолационен инструмент

Задача

Изолационният инструментът (ИзИ) решава следната задача: за целите на тестването да може да се премахва временно зависимостта на приложението, което се тества, от една или повече външни софтуерни услуги.

Зависимостта на приложението от външни софтуерни услуги може да се проявяви по един от следните начини:

- посредством *синхронно изпълнение на операция*, предоставена от външна софтуерна услуга (Invoke активност в BPEL описанието). *Invoke* активността получава набор от входни данни, извиква външната операция с тези входни данни, изчаква приключването на изпълнението и връща набор от изходни данни. Съответно, *Invoke* активността влияе върху приложението посредством своите изходни данни (резултата от изпълнението);
- посредством *асинхронно изпълнение на операция*, предоставена от външна Услуга (комбинация от Invoke+Receive активност в BPEL описанието). При асинхронното изпълнение, Invoke активността получава набор от входни данни, извиква външната операция с тези входни данни и без да изчаква операцията да завърши, връща контрола на бизнес процеса, без да променя данни в него. Резултатът от изпълнението (набор от изходни данни) се получава на по-късен етап от изпълнението на процеса, посредством Receive активност, която получава съобщението за приключване на операцията и предоставя набора от изходни данни на процеса. Следователно, при асинхронно изпълнение, резултатите от Receive активността са тези, които влияят на приложението;
- посредством *непредизвикано получаване на съобщение* от външна софтуерна услуга (Pick активност). Pick активността има два клона – 1) получаване на съобщение и неговата обработка (OnMessage) и 2) неполучаване на съобщение в рамките на предварително установен интервал от време (OnAlarm). В първия случай, зависимостта от външната софтуерна услуга се проявява посредством входните данни, получени със съобщението и последващата обработка в съответния клон, а във втория случай само посредством обработката в съотетния клон;

- посредством изпращане на съобщение на външна софтуерна услуга (в резултат на получаване на входно съобщение от нея). В този случай изпращането на съобщението по никакъв начин не влияе на по-нататъшния ход на приложението;
- посредством активността HumanTask, която изисква човешка намеса и която влияе върху приложението посредством изходните си данни (напр. въведени от оператора стойности);

Горните активности изчерпват вариантите, посредством които изпълнението на бизнес процеса може да зависи ***явно*** от дадена външна софтуерна услуга. По принцип е възможно хода на изпълнение на приложението да зависи ***неявно*** от дадени външни Услуги, посредством механизъм за обмен на данни между външните софтуерни услуги, който е недостъпен за процеса (например софтуерна услуга А записва някакво състояние във външна база данни, от където по-късно софтуерна услуга Б го прочита). Такова поведение нарушава основния принцип за независимост на софтуерните услуги една от друга и води до компроментиране на идеята за SOA. Поради тази причина подобни зависимости няма да бъдат предмет на по-нататъшно разглеждане.

Подход

Invoke активността може да се моделира чрез следния израз:

(7) $$o = f(i_1, i_2, \ldots, i_n, R)$$

Тук чрез f е отбелязана функционалността, която се реализира от операцията, предоставена от външната Услуга, чрез i_1 до i_n – входните данни, които се подават на операцията, чрез o резултата от изпълнението, и чрез R допълнителни параметри на активността, които не са пряко свързани с изпълнението на операцията (например име на активността, координати на съответния блок в BPEL диаграмата и т.н.).

За да премахването на зависимостта от f има два подхода:

1. да се замени f с друга операция f, която е под собствен контрол и която има същият синтаксис (т.е. получава същите входни данни и връща същият тип изходни данни):

(8) $$o = \overline{f}(i_1, i_2, \ldots, i_n, R)$$

2. да се замени f с константа, чийто тип съответства на типа изходни данни на f:

(9) $$o = C$$

Подход 1) е известен и прилаган във всички разглеждани среди за създаване на SBA. В тези среди той е известен като *“Емулация”* и се състои в замяната на извикването на същинската операция с извикване на т.н. stub операция, която има същия синтаксис (т.е. не се налага промяна на модела на данните), но която е под контрола на интегратора и той може да указва какви са изходните данни на stub софтуерната услуга. За да се емулира действието на 1 операция f са необходими следните модификации:

- промяна на крайната точка за свързване (end-point) на софтуерната услуга, която предоставя операцията *f*, така че тя да сочи към stub реализацията, вместо към същинската Услуга;
- създаването на stub софтуерна услуга и инсталацията й в сървъра за приложение;
- реализация на stub операцията, чрез написване на код на (напр. на Java), който връща необходимия на потребителя резултат *o*;

Следователно, при емулирането на действието на 1 операция, в обкръжението (приложния сървър, средата за програмиране и т.н.) се появяват следните тестови артефакти:

- вариант на BPEL-а, с променена точка за свързване на Услугата;
- stub Услуга с реализирана 1 операция, която се емулира;

Тези артефакти са необходими само за процеса на осигуряване на качеството, а не за коректната работа на системата по време на експлоатацията.

Недостатъкът на подхода се проявява при нуждата да се тестват различни комбинации между емулирани и същински извиквания на различни операции по даден път в процеса, като при това трябва да се има предвид, че една софтуерна услуга може да предоставя множество операции. Броят на паразитните тестови артефакти може да нарастне в експоненциална зависимост от броя на активностите в процеса. Средствата за разработка на приложения обикновено предоставят автоматизация на процеса на създаване на емулация на дадена операция, но не предлагат средства за ефективно управление на тестовите артефакти, както и систематизирането им в тестови сценарии. Така може да се окаже, че сървърът за приложения е пълен със stub Услуги, всяка от тях с различни реализации на някакво подмножество от операциите си и управлението им е непосилна задача.

Намаляването на необходимите тестови артефакти е от особено значение за ефективното приложение на инструмента за премахване на зависимостите. Поради тази причина е възприет подход 2), а именно замяната на извикването на операцията *f* с константа от съответния тип данни, отговарящ на типа на резултата на *f*. При този подход (наричан по-долу *“Изолация”*), се налага модификация на процеса, като съответната Invoke активност се замени с Assign активност, която присвоява на изходната променлива *o* конкретни стойности, предписани от потребителя.

При изолацията на процеса от една активност се създава тестов артефакт – вариант на BPEL процеса, в който съответната Invoke активност е заменена с Assign активност. И тук броят на тестовите BPEL варианти може да расте експоненциално, но веднага се вижда, че в сравнение с подход 1) тук няма нужда от създаване, внедряване и реализиране на stub Услуга и съответно такива артефакти в средата лисват. При това различните варианти на BPEL процеса могат лесно да се управляват посредством систематизацията им в тестови сценарии, за което се грижи инструмента за генериране на тестови сценарии.

Към останалите зависимости се подхожда по подобен начин – напр. при асинхронното извикване на операция (Invoke + Receive), Invoke активността се

заменя с Empty активност (тъй като не оказва влияние върху процеса), а Receive активността се заменя с Assign. Таблица 11 по-долу дава начините, по които процеса се изолира от различните видове зависимости.

Таблица 11 Изолация на процеса от различни видове активности, водещи до зависимости

Зависимост (същинска активност в BPEL)	**Замяна с**
Invoke	Assign
Invoke+Receive	Empty+Assign
Pick/OnMessage	Assign
Pick/OnAlarm	Empty
Reply	Empty
HumanTask	Assign

Проучванията показват, че предложения подход не се среща в известните среди за разработка на софтуерни приложения и следователно *създаването и приложението му представлява принос*.

3.3.3.2. Инструмент за анализ на зависимости

В рамките на предлаганата архитектура се предлага инструмент, който да улеснява специалистите по качество при тестването на конкретна последователност от активности. Инструментът постига това чрез идентифициране на условните изрази, от които зависи преминаването по тази последователност, както и променливите, участващи в тях. Инструментът се нарича Инструмент за анализ на зависимости (ИАЗ) и представлява уеб услуга с няколко метода. За по-голямо удобство нататък в текста вместо Инструмент за анализ на зависимости ще се използва съкращението ИАЗ.

Задача

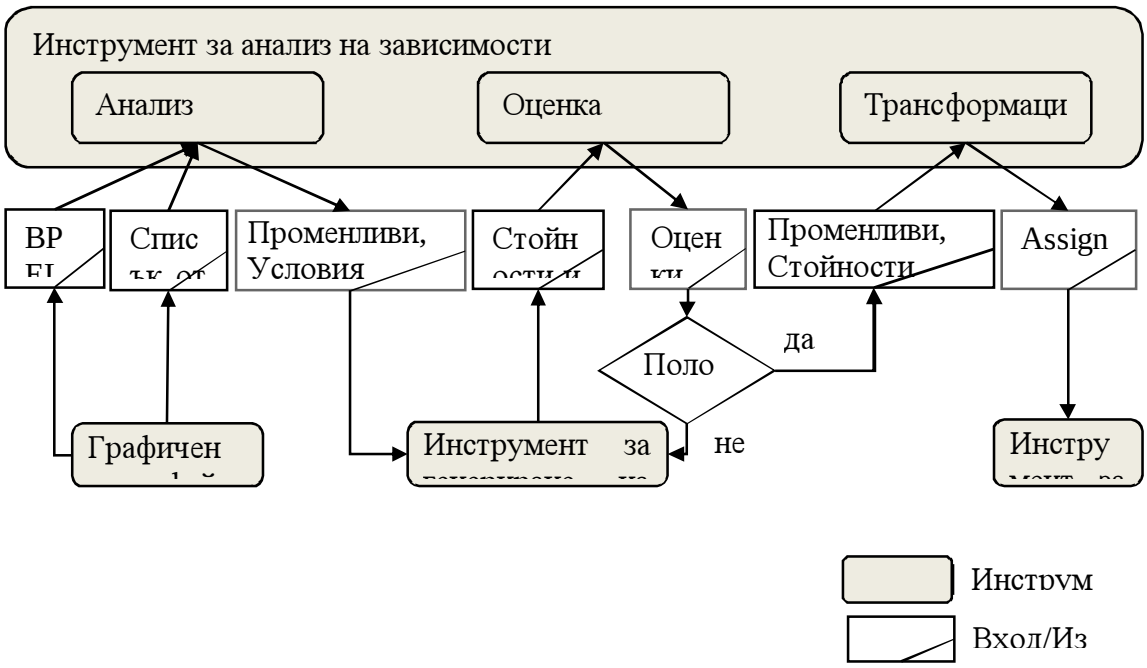
Една от основните дейности на специалистите по осигуряване на качеството по време на проектирането е тестване на резултатите от изпълнението на множество различни *конкретни* последователности от действия (пътища в процеса). Различните пътища се обуславят от наличието на условни активности, които водят до разклонение в дадени точки. Най-простата такава активност е IF(C, A, B) – ако условието C е изпълено, се продължава с последователност от активности A; в противен случай – с последователност B. Това води до два възможни пътя в процеса, като това, по кой път ще протече изпълнението, зависи от условието C. Условието C в общия случай е булева функция на една или повече процесни променливи (данни).

За да тества даден път в процеса, специалистът по качество трябва да идентифицира всички условия по пътя, за всяко условие да установи от кои променливи зависи и да се погрижи по време на изпълнението съответните променливи да имат подходящи стойности, така че условието да се изпълни или

да не се изпълни, така че процесът да продължи по дадения път. В сложни процеси това може да се окаже трудоемка задача. Всичко необходимо за решаването й обаче се намира в описанието на процеса (BPEL+WSDL+XSD), което може да се анализира и резултатите от този анализ да се използват за подпомагане на действията на специалиста по качество. Именно тази задача се решава от ИАЗ: **при зададен път в процеса, да се намерят всички условни активности по пътя, да се установи кои данни влияят върху тях и какви стойности трябва да имат данните, за да протече изпълнението по предварително зададения път.**

Инструментът за анализ на условните зависимости (ИАЗ) е софтуерен инструмент (Фигура 18), който улеснява специалистите по качество при тестването на конкретна последователност от активности (пътища) в BPEL процеси. Различните пътища се обуславят от наличието на условни активности, които водят до разклонение в дадени точки.

От алгоритмична гледна точка ИАЗ реализира ефективно обхождане на дадено XML дърво, описващо BPEL процес.



Фигура 18 Инструмент за анализ на условните зависимости

Подход

За формалното описание на подхода за решаване на задачата, се въвеждат:

Дефиниция 1: *D* – множеството от всички процесни променливи, независимо от техния тип. Всяка процесна променлива може да участва в описанието на условията и всяко условие работи само с процесни променливи и константи.

Дефиниция 2: *F* – множеството от всички булеви функции върху една или няколко процесни променливи. Функциите в *F* се изграждат от елементарните операции, достъпни в BPEL. В [BPEL] е дефинирана стандартна библиотека от

елементарни операции в различни области – математика, работа със символни низове, преобразование на дати и др., като тези елементарни операции се описват посредством XPath. Трябва да се отбележи, че IBM WebSphere ESB поддържа и функции, реализирани чрез Java, които обаче не са част от BPEL стандарта и съответно не се поддържат от DDA инструмента.

Дефиниция 3: *G* – Граф на изпълнението (Control Flow Graph), който отговаря на изследвания BPEL процес. Представлява двойката $G = (V, E)$, където V е множеството от върхове, а E – насочена условна бинарна релация между върховете $E \subseteq \{(u, f \times v) \mid u, v \in V, f \in F\}$ (дъги). В графа на изпълнението върховете отговарят на активности в BPEL процеса, а дъгите – на *реално* възможнен преход от активността *u* в активността *v*. Това, дали даден преход ще се реализира в действителност, зависи от условието на прехода *f*.

Дефиниция 4: Път (в графа на изпълнението) $p = [v_1, v_2, ..., v_n]$ – последователност от върхове в *G*, които отговарят на последователност от активности, които се изпълнява по време на работата на BPEL процеса.

Дефиниция 6: *cdr(p)* -- десен под-път на пътя *p*. Представлява пътят *p*, без първата активност: ако $p = [v_1, v_2, ..., v_n]$, то $cdr(p) = [v_2, ..., v_n]$.

Дефиниция 5: $P \subseteq 2^V$ – множеството на валидните пътища – такива, които *реално* могат да се изминат по време на изпълнението.

Лема 1: Ако $u \in V$ и $e = (u, f \rightarrow v) \in E$, то $[u, v] \in P$.

Доказателство: По дефиниция. Съгласно Дефиниция 3, преходът между *u* и *v* е възможен поради наличието на дъгата *e*. „Възможен преход“ е друго словосъчетание за „може реално да се измине по време на изпълнението“ Q.E.D.

Лема 2: Ако $w \in V$, $e = (w, f \rightarrow v_1) \in E$ и $p = [v_1, v_2, ..., v_n] \in P$, то $[w, v_1, v_2, ..., v_n] \in P$.

Доказателство: Съгласно Дефиниция 3, преходът между *w* и v_1 е възможен поради наличието на дъгата *e*. Преходите между v_1 и v_2, както и от v_2 до ... и от там до v_n са възможни, тъй като са част от валидния път *p*. Следователно, всички преходи между последователните активности в пътя $[w, v_1, v_2, ..., v_n]$ са възможни, т.е. той може да бъде *реално* изминат по време на изпълнението и съответно е валиден. Q.E.D.

Следствие от Лема 2: За всеки път p = [v1, v2, ..., vn] е вярна импликацията:

$$cdr(p) \in P \wedge \exists e = (w, f \rightarrow v_1) \in E \Rightarrow p \in P \ ,$$

или с други думи – ако десния под-път на p е валиден и в графа на изпълнението има дъга между първите два върха в p, то p е валиден път.

Основната задача на инструмента може да се раздели на 2 под-задачи:

1. Намиране на всички условни активности, включени в даден път в процеса и за всяко условие, намиране на променливите, които участват в него
2. При зададени стойности на променливите, открити в т. 1, проверка дали процесът ще поеме по искания разклонение, така че да трасира зададения път през условната активност.

Под-задача 1 се решава по следния **Алгоритъм 1**:

Вход: пътят $p \in P$, който се изследва.

Изход: множество от променливи *d* и условните изрази *f*, в които те участват.

1. init(D, G)
2. $d = \emptyset, f = \emptyset$
3. $\forall a \in p$
   - 3.1. $\forall e \in E \wedge e.u = a$
     - 3.1.1. ако match(e.v, cdr(p)), то
       - 3.1.1.1. $f = f \cup e.f$
       - 3.1.1.2. $d = d \cup dex(e.f)$
       - 3.1.1.3. изход от цикъла 3.1 чрез преход към следващата итерация на стъпка 3
   - 3.2. грешка – невалиден път. Край.
4. Край

В стъпка 1 се инициализират множествата *D* и *G*. Това се извършва чрез анализ на BPEL, WSDL и XSD представянето на процеса със средствата за XML парсиране, предоставени от средата и следване на съответните стандарти за описание.

Стъпка 2 продължава с инициализация на изходните променливи. След това, за всяка активност *a*, която е част от пътя *p*, (стъпка 3), се изпълнява стъпка 3.1, където за всяко възможно разклонение след активността *a*, се изпълнява стъпка 3.1.1. В нея, чрез функцията *match* се установява дали пътя [*e.v, cdr(p)*] е валиден. Това става чрез рекурсивно приложение на **Следствието от Лема 2**. Ако [*e.v, cdr(p)*] е валиден, то условието *e.f* трябва да е изпълнено, за да продължи процеса по пътя *p*, следователно *e.f* се добавя към множеството от условия *f* (стъпка 3.1.1.1), а процесните променливи, които участват в него *dex(e.f)* се добавят към множеството от променливи *d*. Ако нито едно условие не води до преминаване по пътя *cdr(p)*, то пътя *p* е невалиден, тъй като не може по никакъв начин да се стигне до края му (стъпка 3.2 завършва алгоритъма с грешка).

Извличането на променливите, които участват в дадено условие (стъпка 3.1.1.2.) става чрез функцията *dex*, която по същество представлява парсер, който интерпретира стандартните XPath изрази, чрез които се реализира BPEL библиотеката от елементарни операции. Създаването на тази функция е и основната дейност по създаване на инструмента, тъй като библиотеката е доста обемиста. За изпълнението й се използват стандартни похвати за създаване на интерпретатори, като изследването на граматиката показва, че LR(1) парсер е подходящ за целта. Потенциално решение е JavaCC инструментът за създаване на LR(1) парсери, за пресъздаване на граматиката на XPath изразите от стандартната библиотека. Полученият генератор на код реализира функцията *dex* – при зададен XPath израз, идентифицира променливите и ги връща като множество.

Под-задача 2 – при зададени стойности на променливите, открити в под-задача 1, проверка дали процесът ще поеме по исканото разклонение, така че да трасира зададения път през условната активност – представлява функционалност, която вече е налична в средата, тъй като е необходима за самото изпълнение на BPEL процеса. Следователно, под-задача 2 може да се

реши посредством извикването на подходящи API функции на средата. Съответните функции обаче не са стандартизирани и са различни за различните BPEL среди, така че в ИАИ инструмента е включен допълнителен слой, който реализира платформената независимост и скрива спецификите на извикване на съответните API средства.

3.3.3.3. Инструмент за генериране на стойности

В рамките на предлаганата архитектура е предвиден инструмент, който да генерира тестови данни, които ще се използват от останалите инструменти при създаването на тестови сценарии. Задачата за генериране на стойности по зададени правила е намерила конкретно решение в редица инструменти. Поради тази причина усилията са насочени към идентифицирането на подходящ софтуерен инструмент и интегрирането му в архитектурата на тестовата среда.

Задача

Основната задача, която решава Инструментът за генериране на стойности (ИГС) е **по зададена XSD структура на дадена променлива, да се генерират валидни стойности за всичките й полета.** Така генерираните данни намират приложение при създаването на тестови сценарии за самостоятелното тестване на уеб-услуги (генерация на входни данни), при цялостното тестване на BPEL процеса (генерация на входни данни) и при тестването на конкретен път в процеса (генериране на стойностите на променливите за съответните условия). Първите две приложения са добре познати и използвани, докато третото надгражда ИАИ инструмента и е специфично за настоящата платформа.

Подход/решение

Една голяма част от подходите за генериране на тестови данни използват WSDL спецификациите на услугите в комбинация с информацията за типовете данни в XML Schema [17, 18, 19, 20, 21, 22, 24]. Информацията за типовете данни в XML Schema се използва за определяне на ограниченията върху типовете, в резултат на което могат да се генерират тестови данни. Възможно е също така да се дефинират и комплексни типове данни, като се използва декомпозиция на прости типове. Генерират се тестови данни за простите типове, след което те се комбинират, за да се получат тестови данни за комплексните типове. Информацията от WSDL спецификацията може да се комбинира със знанието на тестера за данните. Подобен подход е приложен от авторите в [26].

Възприетият подход за решаване на задачата за генериране на тестови данни е избиране на съществуващ софтуерен инструмент, който да се интегрира в настоящата тестова рамка, като се надгради за специфичните й нужди. Като подходящ софтуерен инструмент е идентифицирана платформата WS-TAXI [18].

TAXI платформата прилага върху XML Schema добре известната техника ‘’Разделяне на категории” (Category Partition) [25]. От дефиницията на XML Schema се анализират типовете и честотата на срещане на елементите, като се определят и ограниченията. С помощта на получената информация и структурата на схемата TAXI генерира множество от междинни инстанции, комбинирайки стойностите за честотата на срещане на елементите. Крайните инстанции се получават от междинните посредством присвояване на стойности

към елементите. За целта могат да се използват два подхода: стойностите могат да се вземат от интегрирана към TAXI база от данни или да се генерират по случаен начин. В текущата си версия TAXI използва по една стойност за елемент на всяка инстанция.

Общата архитектура на Инструмента за генериране на стойности (ИГС) е показана на фигура 19.

Фигура 19 Инструмент за генериране на стойности

3.3.3.4. Инжекционен инструмент

До този момент, изброените по-горе инструменти осигуряват всичко необходимо за позитивното тестване на даден бизнес процес. За да се направи, обаче цялостна оценка е необходимо, да се обърне внимание и на негативното влияние на зашуменост или прекъсване на комуникационните канали.

За тази цел е предвиден инструмент наречен Инжекционен инструмент (ИнИ), който предоставя тази функционалност. Изработен е прототип, който следва подходът на останалите инструменти в проекта, а именно да се правят промени единствено в BPEL описанието на бизнес процеса.

Задача

Задачата, която решава Инжекционния инструмент, е свързана с определяне на поведението на изследвания процес при проблеми в комуникационните канали. Тъй като в общия случай системният интегратор няма контрол върху голяма част от инфраструктурата, използвана за връзка между софтуерните услуги, се налага проблемите в инфраструктурата да се симулират, вместо да бъдат създадени наистина. Като възможни и смислени за симулация са идентифицирани следните ситуации:

- претоварване на някой от комуникационните елементи – проявява се в забавяне на изпращането/получаването на дадено съобщение;
- повреда в някой от комуникационните елементи или спиране на тока – проявява се в прекъсване на комуникационния канал преди/по време на изпращане/получаване на дадено съобщение;
- зашуменост на комуникационен канал – проявява се в получаване на съобщение, съдържащо случайни грешки, вкл. в структуроопределящите елементи;
- грешна бизнес логика на дадена софтуерна услуга – получаване/изпращане на съобщение, съдържащо синтактични грешки в данните, но не и в структуроопределящите елементи (напр. запис на дата в целочислено поле);

Именно тази задача решава ИнИ – **при изпращането на дадено съобщение да може да се симулира произволна валидна комбинация от повреди**. Наличието на повреда от даден вид и нейните параметри да се контролира от потребителя. Наличието на функционалност за симулиране на подобни повреди ще позволи на специалистите по качество да проведат т.н. *негативни функционални тестове*, т.е. да проверят поведението на процеса при неблагоприятно стечение на обстоятелствата.

Подход/решение

Обхватът на различните повреди, които трябва да се симулират, това, че комуникационната инфраструктура не е под контрола на системния интегратор, както и факта, че повредите следва да се проявяват при някои от съобщенията, а при други – не, налагат да се търси софтуерно решение на проблема. Както и при останалите инструменти, така и тук се търси подход, който да надгражда създадените вече инструменти.

Първият проблем, който трябва да се реши е как да се идентифицира съобщението, при обмена на което трябва да се симулира повреда. Всяко съобщение е породено и следователно може да се идентифицира с дадена активност в BPEL процеса. Така например Invoke активността води до изпращане на съобщение за изпълнение на дадена операция. Посредством идентифициране на активността, която поражда съобщението може да се идентифицира и съобщението.

След като съобщението е идентифицирано, трябва да се предприемат стъпки за модификация на канала, така че да се случи исканата от потребителя комбинация от повреди със съответните параметри. За целта между изпращача и получателя на съобщението се инсталира посредник (прокси), който получава

съобщението и го препраща до истинския получател, но междувременно реализира повредата (напр. забавя изпращането, изобщо не го изпраща, модифицира съобщението, като внася грешки и др.).

За да се изпрати съобщението на посредника, вместо на истинския получател, се налага да се модифицира съответната активност в BPEL процеса, по начин, който не е много по-различен от модифицирането, извършвано от Isolation инструмента.

Посредникът трябва да е универсален – т.е. да може да замести произволно извикване на операция от произволна услуга. Поради тази причина се налага допълнително кодиране/декодиране на входно/изходните данни на активностите, които се модифицират. За целта могат да се използват вградените в BPEL функции за маршализация и демаршализация. Преди извикване на посредника XML структурите, които съставляват аргументите на истинския получател се линеализират по уникален начин в символен низ (маршализация). Този символен низ се подава като аргумент на посредника, който преди да изпрати съобщението на истинския получател изрършва обратната операция – демаршализация – и преобразува символния низ в оригиналната XML структура, която се подава на истинския получател. При получаване на отговор се действа по аналогичен начин – получената структура се маршализира в символен низ, който се връща на BPEL процеса, където, преди да се продължи с оригиналната бизнес логика, стрингът се демаршализира и се попълват изходните променливи на активността. Формално погледнато, се извършва следната модификация:

$$o = Invoke(i_1, i_2, ..., i_n) \Rightarrow o = UnMarshal(ProxyInvoke(Marshal(i_1, i_2, ..., i_n), R));$$

$$o = Invoke(i_1, i_2, ..., i_n) \Rightarrow o = UnMarshal(ProxyInvoke(Marshal(i_1, i_2, ..., i_n), R));$$

където с i_1, i_2, ..., i_n са отбелязани оригиналните аргументи на модифицираната операция *Invoke*; с o – оригиналните изходни данни; с *Marshal* и *UnMarshal* са отбелязани вградените в BPEL функции за маршализация и демаршализация, а с *ProxyInvoke* – извикването на посредника с параметризация относно исканите повреди *R*.

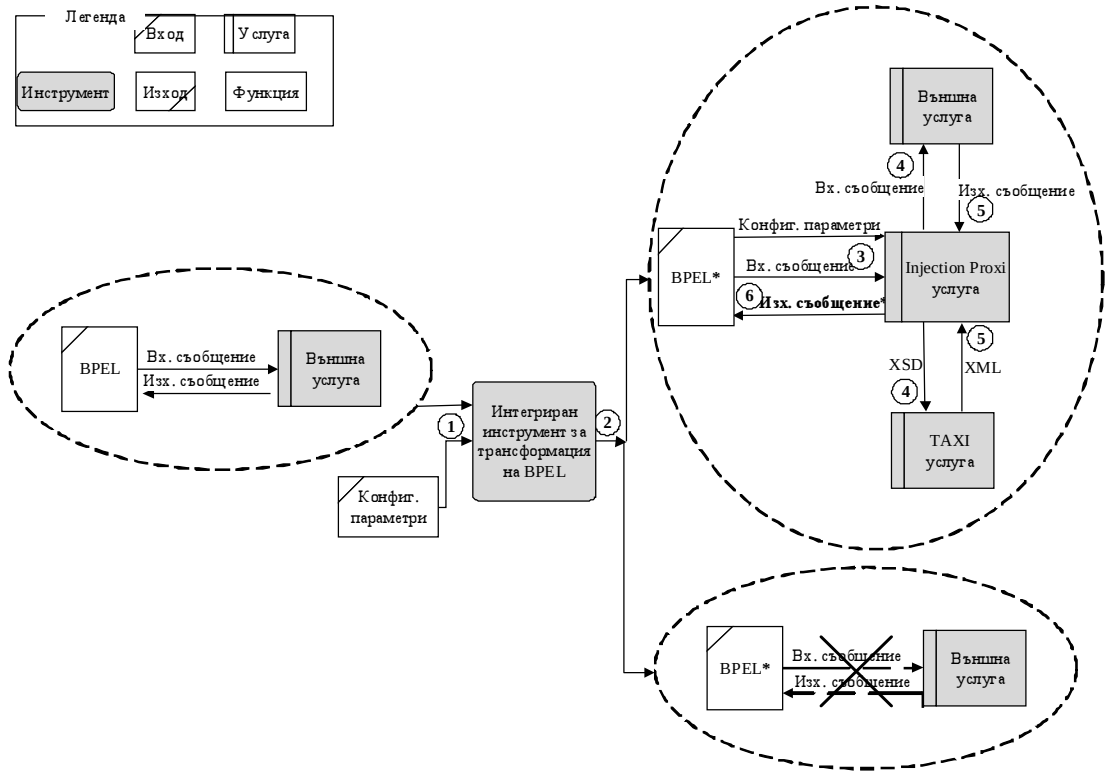
Самото действие на ProxyInvoke е тривиално:

- при необходимост да се симулира забавяне, се извършват следните действия:
  - извиква се оригиналната операция;
  - изчаква се зададения брой секунди (параметър на симулацията);
  - резултата, получен от извиканата оригинална операция се връща на BPEL процеса;
- при необходимост да се симулира прекъсване се генерира съответно изключение (exception), напр. посредством throw;
- при необходимост да се симулира зашуменост, по случаен начин се внася "боклук" в съобщението, като количеството му се контролира от параметър на симулацията;

- когато следва да се симулират грешки в бизнес логиката ситуацията е доста по-сложна, тъй като се налага синтактично разбиране на съобщението и генериране на случайни данни, които обаче няма да развалят структурата на самото съобщение. За щастие точно тази функционалност – по зададена структура да се генерират случайни данни – се реализира от инструмента ИГС. Именно той се използва при симулация на грешки в бизнес логиката, като количеството им се контролира от параметър на симулацията.

Трябва да се отбележи, че така избраният подход е приложим само към Invoke активности, тъй като те са единствените активности, при които обмяната на съобщения се инициира от бизнес процеса, който се изследва. Това е необходимо условие за реализация на подхода, тъй като се налага поставянето на активности за маршализация и демаршализация около инициатора на обмяната и когато инициаторът не е под собствен контрол, това е невъзможно. Възможни са и други подходи, при които такова ограничение няма, напр. добавяне на инфраструктурно ниво на обекти, отговарящи на комуникационни канали, зашумени по различен начин. Такъв подход обаче е свързан със сериозни модификации на ESB шината, при това тъй като реализацията на комуникационните канали не е стандартизирана, всяка ESB шина ги реализира по различен начин, което налага модификациите да се правят за всяка шина поотделно. Освен това управлението и параметризацията на комуникационните канали (при кое извикване кой канал да се ползва) се усложнява прекомерно, тъй като трябва да се реализира механизъм за корелация с инстанциите на процесите. Въобще, при претегляне на предимствата и недостатъците на този метод се оказва, че той е прекалено сложен за реализация и управление, а предоставя твърде малко в допълнение към избрания подход.

На фигура 20 са показани интегрирани изолационен и инжекционен инструмент



Фигура 20 Изолационен и инжекционен инструмент

3.3.3.5. Инструмент за управление и изпълнение на тестови сценарии

Досега разгледаните инструменти създават предпоставки за автоматизация на функционално тестване по време на проектирането на:

- самостоятелни услуги – WS-TAXI;
- процес, изолиран от комбинации от външни услуги – Isolation Tool;
- конкретни пътища в процеса – DDA Tool, Isolation Tool, VG Tool;
- поведение на процеса при проблеми в комуникационните канали – Injector Tool.

Следваща задача, необходима за пълното автоматизиране на тестовия процес, е проектирането и разработването на Инструмент за управление и изпълнение на тестови сценарии (ИУИТС).

Задача

При така изградената инфраструктура лесно се достига до следващата стъпка, а именно **автоматично тестване на всички възможни пътища в процеса** (full patch coverage). Освен това, в проучваните среди липсва единен механизъм за управление на тестови случаи – създаването на тестов случай, независимо от вида му, създаването на всички артефакти, свързани с него, запазването му в хранилището, последващо изпълнение на множество тестови случаи, филтрирани по дадени критерии.

Именно това са и основните задачи на Инструмента за управление и изпълнение на тестови сценарии:

- да може да идентифицира всички възможни пътища в процеса и да подпомогне специалиста по осигуряване на качество в създаването на тестови сценарии за пълно функционално тестване от край до край (full path coverage);
- да предостави механизъм за управление и хранилище с тестови случаи;

Подход/решение

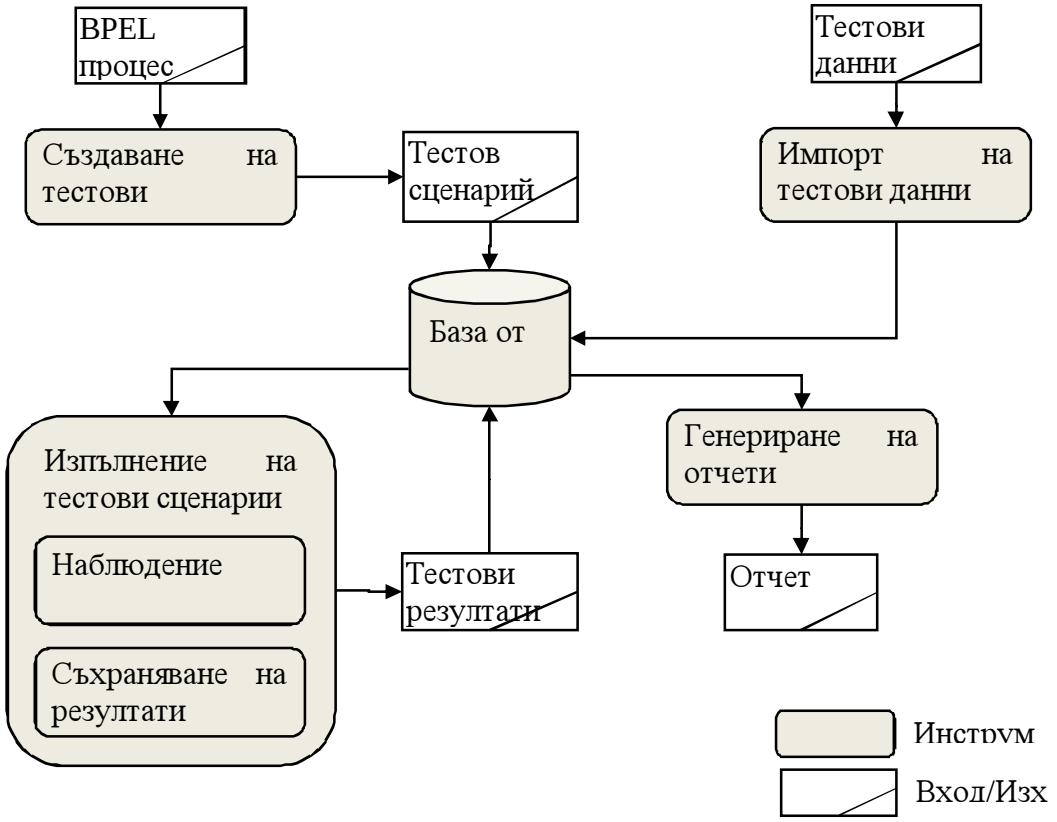
Към Инструмента за управление и изпълнение на тестови сценарии са поставени две основни задачи – генериране на тестови сценарии за всички възможни пътища в процеса (full path coverage); и среда за управление на хранилище от тестови случаи.

ИУИТС трябва да предоставя средните основни функционалности:

- Функционалност за описание на тестови сценарии за тестване на BPEL процес.
- Функционалност за групиране на тестови сценарий в тестови пакети. Всеки тестов пакет се обвързва с конкретен BPEL процес, който се тества.
- Функционалност за импортиране на тестови данни от външен CSV файл.

- Функционалност за изпълнение на тестови сценарии. Изпълнението на тестовите сценарии е свързано с интеграцията на ИУИТС със средата за разработка на Sun – NetBeans.
- Функционалност за наблюдение на изпълнението на тестовите сценарии.
- Функционалност за съхранение на резултатите от изпълнението на тестовите сценарии.
- Функционалност за получаване на статистическа информация за тестовия процес на база на дефинирани от специалиста по качество критерии.

На Фигура 21 е представена архитектура на прототип на инструмента ИУИТС.



Фигура 21 Инструмент за управление и изпълнение на тестови сценарии

Генерирането на тестови сценарии изисква трансформиране на BPEL описанието на процеса в междинен модел при използване на конкретен формализъм с цел генериране на тестови пътища. За целта е направено проучване на съществуващите подходи за тестване на BPEL процеси. Подходите са сравнение въз основа на няколко критерия:

- Формализъм, използван за трансформиране на BPEL описанието на процеса в междинен модел, подходящ за генериране на тестови пътища (Ф);
- Степен на покритие на BPEL дейностите, а именно обхващане на дейностите по прихващане на грешки (ПГ) и прихващане на изключения (ПИ);

- Валидиране на подхода чрез експериментални данни и примерни сценарии (В);
- Имплементация на подхода (И);
- Автоматично генериране на тестови сценарии (ТС);
- Поддръжка на регресионно тестване (РТ).

Таблица 12 по-долу обобщава резултатите от направеното проучване.

Таблица 12 Сравнение на съществуващите подходи за тестване на BPEL процеси

Подход	Ф	ПИ	ПГ	В	И	ТС	РТ
Yuan [13]	CFG	-	-	-	-	+	-
Lertphumpanya [5]	CFG	-	-	+	+	+	-
Dong [2]	HPN	-	-	+	+	+	-
Yuan [12]	UML2.0 activity diagram	-	-	+	+	+	-
Zhang [14]	UML2.0 activity diagram	+	-	-	+	+	-
Li [6]	CFG	+	+	+	+	+	+
Yan [11]	CFG	+	+	-	+	+	+
Mayer [9]	-	-	-	+	-	+	-
Choy [1]	-	-	-	+	+	+	-
Ma [8]	SXM	+	-	-	-	+	-
Zheng [15]	WSA	+	+	+	-	+	-
Fanjul [3]	PROMELA, LTL	-	-	+	+	+	-
Aalst [10]	PN, WF-net	+	+	+	+	-	-
Karam [4]	CFG, DFG, SFG	-	-	-	-	-	-
Li [7]	Java	-	-	+	+	+	+

При направеното изследване се установи, че най-ефективен е подходът, при който изследваният BPEL се представя посредством Control Flow Graph (CFG). За това има множество причини, сред които:

- от горната таблица се вижда, че CFG подхода е най-широко приложен;
- CFG подхода вече се използва в описваните по-горе инструменти, а именно в DDA инструмента. Това води до повторно използване на съществуваща функционалност и съответно до икономии;
- За намиране на всички пътища в граф съществуват добре познати алгоритми от комбинаториката, които могат да се използват „наготово“. Намирането на всички възможни пътища в граф обикновено се счита за трудна задача, поради факта, че нейната сложност е $O(n!)$, където n е броят на върховете в графа. Опитът обаче показва, че обикновено активностите в процесите, които се изследват, не надвишават няколко десетки и от тях около 1/3 са активности, които могат да водят до

разклонение. Поради тази причина *намирането на всички пътища в даден процес посредством пълно изброяване* е обозримо и съответно е избран този подход.

- генерирането на всички възможни пътища в граф може да стане и посредством итерационен механизъм – на всяка стъпка се генерира по случайност един път (който все още не е генериран след първоначална инициализация). Така всички възможни пътища (N на брой), се генерират след N извиквания на съответната операция. Този интерфейс може да се използва от други инструменти, за реализация на алгоритми за намиране на покритие в случаите, когато пълното изброяване не е възможно (напр. при силно свързан граф със 100 върха, може да се окаже, че възможните пътища са 100!). В случая може да се използват напр. еволюционни алгоритми за оптимизация на сценариите, така че да се направи възможно най-голямо покритие.

Хранилището за тестови сценарии по същество представлява база данни, в която за всеки тестови сценарии се пази определен набор атрибути, свързани с него, като напр.:

- уникален идентификатор на сценария, даден от специалиста по качество;
- вид на сценария – един от поддържаните видове тестови сценарии:
  - тест на самостоятелна Услуга (VG Tool, WS-TAXI);
  - тест на BPEL, изолиран от външни услуги (Isolation Tool);
  - тест на конкретен път в BPEL (DDA Tool, Isolation Tool, TAXI);
  - тест при условията на зашумен комуникационен канал (Injector Tool);
  - тест на всички възможни пътища в BPEL (TCG Tool);
- списък и указатели към тестовите артефакти, които са необходими за изпълнението на съответния сценарии – напр. модифицирани BPEL файлове, входни данни за инструментите и др. в зависимост от самия тест.
- Очакван резултат, с който се сравняват получените изходни данни;
- История – кога сценарият е бил ползван и какъв е бил резултата;
- Правила за приложимост на сценария (напр. Сценарият X не се прилага във версия Y);
- Средствата за управление на тестови сценарии предоставят функционалност за манипулиране на горния списък:
- създаване на нов тестови сценарий;
- редактиране на тестови сценарии, вкл. На различните свързани данни (артефакти, очакван резултат, правила за приложимост и др.);
- изтриване на тестови сценарии;
- пускане на един или повече тестови сценарии;

3.4. Заключение

Приложение на предлаганата тестова рамка

Предлаганата тестова рамка може да се използва **по време на интеграция на бизнес процеси**, когато не всички включени услуги са налични или достъпни, както и **за тестване** при позитивни и негативни тестови случаи.

Използвайки тестовата рамка **процесът на тестване се автоматизира**, като за целта се предлага разработката на софтуерни инструменти, на базата на специфицирани тествани прототипи.

В рамките на функционалното тестване, предложената тестова рамка подпомага работата на софтуерния специалист да тества:

- покритие на пътища според функционалните изисквания и

- покритие на нововъведени пътища, които могат да се определят като силно-зависими от входните данни процеси.

Благодарение на възможностите за симулация на негативни тестове продуктовата среда може да се използва **за симулация на повреди** и по този начин да провери благонадежността и устойчивостта на системата.

Други две важни направления, в които може да се използва предлаганият продукт е **за тестване и определяне нивата на издръжливостта на системата** при натоварване над допустимото по проект.

Тези възможности за приложение на продуктовата среда и предават характер на **цялостно решение** за тестване на SOA системи при проектиране и разработване.

След направеният преглед на предметната област може да се обобщи, че предлаганите методологии и софтуерни инструменти засягат предимно качеството на ниво услуга. Съществуват и разработки, в които са създадени тестови модели, свързани с определени аспекти на качеството като обмяната на съобщения между услугите, изпълнение на вписаните в договора на услугата параметри, симулация на липсващи услуги на ниво бизнес процеси. Предлаганите решения са частични и решават проблеми само в отделни направления. Може да се обобщи, че приносът на предложената тестова рамка се състои в следните аспекти:

- Осигуряване на цялостно интегрирано решение - според проучванията не се предлага **цялостна среда** за изследване качеството на SOA системи по време на проектиране, а частични решения засягащи изолирани проблеми.
- Предлаганият изолационен инструмент е **оригинален** по отношение на решението за симулация на уеб услуги – чрез реализация на модифициран BPEL, а не чрез създаване на тестови услуги за симулация на реалните такива (stubs). Именно с този подход се цели **повишаване на производителността** на тестването и **неговото качество** – при него не се налага създаванането на допълнителни услуги (единствено за целите на тестване) и се облекчава процедурата за поддръжка на тестовата среда (не се поддържат тестови услуги с техните версии и

конфигурационна информация). Тези дейности повишават **качеството на крайния продукт**.

- Тестовата рамка/среда осигурява цялостно решение за изследване на позитивните и негативни сценарии за даден бизнес процес.

Идентифицираните потенциални ***потребители на методологията и тестовата рамка*** са:

- Системни интегратори на SOA системи;
- Разработчици и тестери на SOA системи;
- Потребители и в частност персоналът за поддръжка на SOA системи.

Тестовата рамка може да се прилага както при разработка на SOA системи, така и при трансформация на наследени софтуерни системи към съвременни софтуерни системи, ориентирани към услуги (SOA).

Може да се обобщи, че използването на тестовата рамка води до:

- Оптимизация на разработването на софтуер;
- Увеличаване на производителността;
- Повишаване на качеството на разработения софтуер.

IV. Заключение

Постигнатите резултати от работата в областта на гъвкави методологии за разработване на софтуерни системи могат да се обобщят по следния начин:

Създаване на гъвкава методология. В резултат на изследвания в тази област бяха предложени различни гъвкави методологии за разработване на софтуер – комбинация от PSP и XP за екип, разработващ е-приложения; комбинация от PSP и XP за индивидуален разработчик; комбинация от Скрум и Екстремно праграмиране за проект на държавната администрация; комбинация от RUP и Скрум за проект за вградени системи. Всеки един от предложените гъвкави методологии води до подобрения спрямо съществуващия преди това процес на работа, макар и в различна степен. Изследванията засягат различни по вид фирми – със сертифициран процес на разработване или дефиниран, с не сертифициран, както и за отделен автономен разработчик. Засегнати са и софтуерни приложения в различни области – за електронно управление, за вградени системи, системи за управление на ресурси и др. Идентифициране са подходящи метрики, които да се проследяват и въз основа на данните от тях се извеждат зависимости и насоки за следващи подобрения.

Избор на гъвкава методология или гъвкави практики. Изследванията в тази област преминават през три стъпки:

- предложена е матрица за избор на подходяща методология, която е специфична за дадена организация и проект. Матрицата може да се разшири, да се концептуализира, да се въведат тегла и да се направи адаптируема.
- Идентифициране са зависимости между гъвкавите практики на избрани гъвкави методологии, като са представе чрзе построен граф с тегла, допълнително се идентифицира и отразява влиянието и на 3 групи фактори – социални, бизнес и технологични при избора на гъвкави практики. Възможни насоки за бъдещо развитие са да се потърси и друг начин за описание на зависимостите, както и да се отчете влиянието на повече фактори.
- Последните изследвания предлагат създаване (предложена е структура) и използване на база знания от натрупан предишен опит и управление на знанието, за избор на подходящи гъвкави практики за разработване на софтуер в зависимост от идентифицирани ситуационни фактори.

Резултатите от работата в областта на тестването на софтуерни системи могат да се обобщят по следния начин:

- Предложена е среда/рамка за тестване на софтуерни приложения, базирани на услуги по време на проектирането. Рамката е съставна, разширяема и базирана на стандарти. Рамката за тестване се състои от интегрирани прототитпа на пет софтуерни инструмента, реализирани като уеб услуги. Към момента не е известна подобна интегрирана тестова рамка и в този смисъл тя представлява принос.
- Предложен е подход за тестване на софтуерни приложения, базирани на услуги, като основно обхваща тестване на ниво процеси/оркестрация на услуги (Process/Orchestration). Процесите се

описват посредсвом стандарти (BPEL+WSDL+XSD). Подходът е ориентиран към тестове по време на проектиране.

- Предложен е тестов алгоритъм, който използва добре познати тестови подходи, като техники за реализация и изпълнение Бяла кутия (White-box) Покритие на пътища (Path coverage), Черна кутия (Black-box), оптимизация на тестовите сценарии на базата на Разделяне по категории (Category Partition). В същото време прилаганият подход за изолация на определени услуги чрез модификация на изходния BPEL е определено ново предложение, което влияе значително върху ефективността (на базата на брой часове за проектиране, реализиране и изпълнение на тестов сценарий) и намаляване на сложността на тестовите активности.
- Ситематизирана е Тестова методология на предложената рамка за тестване

V. Литература към Глава Гъвкави методологии

Манева, Н. Е. (2001). *Софтуерни Технологии.*

Рендал, Й. Т. (1987). *Технология на програмирането*. Техника.

Иванов, Р. (2006). *Прилагане на гъвкави методологии за разработка на софтуер в ISO сертифицирани компании.* София: Дисертация.

Кръстева, И. (2011). *Гъвкава методология за разработка на софтуерни приложения*. София : Дисертация.

Abrahamsson, P., Oza, N., & Siponen, M. (2010). Agile Software Development Methods: A Comparative Review. От *Agile Softawre Development: Current Research and Future Directions*. Springer-Verlag Berlin Heidelberg.

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile software development methods review and analysis. *VTT Publications* (478), 3-107.

Abrahamsson, P., Wasta, J., Siponnen, M., & Ronkainen, J. (2003). New Directions on Agile Methods: A Comparative Analysis. От IEEE (Ред.), *International Conference on Softawre Engineering*, (стр. 244-254). Portland, USA.

Beck, K. (1999). *Extreme Programming Explained: Embrace Change* (US ed. изд.). Addison_Wesley Professional.

Boehm, B. (2002). Get ready for agile methods, with care. *Computer* , стр. 64-69.

Boehm, B., & Turner, R. (2006). *Balancing Agility and Discipline - A Guide for the Perplexed* (4th изд.). Addison-Wesley.

Cao, L., Mohan, K., Xu, P., & Ramesh, B. (2009). A Framework for Adapting Agile Development Methodologies. *European Journal of Information Systems , 18*, стр. 332–343.

Cockburn, A. (2006). *Agile Software Development: The Cooperative Game* (2nd Edition изд.). Addison-Wesley Professional.

Cockburn, A. (2004). *Crystal Clear: A Human-Powered Methodology for Small Teams*. Addison-Wesley Professional.

Cockburn, A. (1998). *Surviving Object-Oriented Projects: A Manager's Guide*. Addison-Wesley Professional.

Cockburn, A., & Highsmith, J. (2001). Agile software development, the people factor. *Computer , 34* (11), стр. 131-133.

Conboy, K. (2009). Agility from first principles: Reconstructing the concept of agility in information systems development. *Information Systems Research , 20* (3), стр. 329-354.

Conboy, K., & Morgan, L. (2010). Future Research in Agile Systems Development: Applying Open Innovation Principles Within Agile Organisation. От *Agile Softawre Development: Current Research and Future Directions*. Springer-Verlag Berlin Heidelberg.

Dingsoyr, T., Dyba, T., & Moe, N. B. (2010). *Agile Software Development - Current Reserach and Future Directions* (1 изд.). Springer-Verlag Berlin Heidelberg.

DSDM Atern. (н.д.). Изтеглено на 3 2011 г. от http://www.dsdm.org/

Gartner. (2010). *Perspective:IT Spending 2010*. Garthner.

Ge, X., Paige, R., & McDermid, J. (2010). An Iterative Approach for Development of Safety-Critical Software and Safety Arguments. От I. C. Society (Ред.), *2010 Agile Conference (AGILE 2010)*, (стр. 35-43).

Highmith, J. ,. (2001). *Agile Software Development: The Business in Inovation*.

Highsmith, J. (1999). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. Dorset House Publishing Company, Incorporated.

Hildenbrand, T., Geisser, M., Kude, T., Bruch, D., & Acker, T. (2008). Agile Methodologies for Distributed Collaborative Development of Enterprise Applications. *International Conference on Complex, Intelligent and Software Intensive Systems*, (стр. 540 -545).

IST-2001-33488. (2005). *D1: eXpert Aproach*. EC.

Jeffries, R. (2004). *jatRtsMetric*.

Jones, C. (2000). *Software Assessments, Benchmarks, and Best Practices*. Addison-Wesley Professional.

Klooster, M., Brinkkemper, S., Harmsen, F., & Wijers, G. (1997). Intranet facilitated knowledge management: a theory and tool for defining situational methods. *Conference on Advanced Information Systems Engineering*, (стр. 303-317). Barcelona, Spain.

Krasteva, I., & Ilieva, S. (2007). Rush into Agile- Analytical Framework for Agile Practices Applicability. *IET International Conference on Agile Manufacturing (ICAM 2007)*, (pp. 229-237). Durham, UK.

Kruchten, P. (2008). Keynote: Situated Agility. *9th International Conference on Agile Processes and eXtreme Programming in Software Engineering*.

McConnell, S. (1994). *Code Complete*. Microsoft Press.

Misra, S. C., Kumar, V., & Kumar, U. (2009). Identifying some important success factors in adopting agile software development practices. *The Journal of Systems and Software* (82), стр. 1869–1890.

Object Management Group. (н.д.). Unified Modeling Language (UML). Изтеглено на 04 11 2007 г. от http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF/

Paige, R. ,. (2008). Towards agile engineering of high-integrity systems. От Springer-Verlag (Ред.), *Computer Safety, Reliability, and Security. 27th International Conference, SAFECOMP 2008*, (стр. 30-43).

Palmer, S., & Felsing, J. (2002). *A Practical Guide to Feature-Driven Development*. Prentice Hall.

Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit* . Addison Wesley.

Pressman, R. (2005). *Software Engineering: A Practititioner's Approach* (6th Edition изд.). McGraw-Hill Higher Education.

Project, A. (2011). Изтеглено на 5 2011 г. от The AGILE ITEA Project: http://www.agile-itea.org/public/info.php

Project, F. (н.д.). Свалено от FLEXI ITEA2 Project: http://flexi-itea2.org/index.php

Qumer, A., & Henderson-Sellers, B. (2009). Agile software solution framework: An analysis of practitioners' perspectives. *Information Systems: Modeling, Development, and Integration: Third International United Information Systems Conference, UNISCON 2009* (стр. 41-52). Lecture Notes in Business Information Processing.

Schwaber, K. (1995). Scrum Development Process. От Springer (Ред.), *OOPSLA Business Object Design and Implementation Workshop*.

Stapleton, J. (1997). *DSDM, Dynamic Systems Development Method: the Method in Practice*. Cambridge University Press.

Takeuchi, H., & Nonaka, I. (1986). The New New Product Development Game. *Harvard Business Review* (1), стр. 1-11.

Turk, D., France, R., & Rumpe, B. (2005). Assumptions underlying agile software-development processes. *Journal of Database Management , 16* (4), стр. 62-87.

VersionOne. (2010). *2010 State of Agile Development Survey*. Свалено от http://www.versionone.com/pdf/2010_State_of_Agile_Development_Survey_Results.pdf

VersionOne. (2008). *3rd Annual State of Agile Survey*. Свалено от http://www.versionone.com/pdf/3rdAnnualStateOfAgile_FullDataReport.pdf

VI Литература към Глава Тестване на софтуерни системи

[1] *Data Flow-based Validation of Web Services Compositions: Perspectives and Examples*, Cesare Bartolini, Antonia Bertolino, Eda Marchetti, Ioannis Parissis, ISBN 978-3-540-85570-5, 2008

[2] IREB Foundation Level, Professional for requirements Engineering, © DELTA, 2010-V1(2.0)

[3] http://www.soa-standards.org

[4] Web Services Business Process Execution Version 2.0. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.

[5] He, H. (30 септември, 2003). *What Is Service-Oriented Architecture?* Изтеглено на 27 Октомври 2007г. от XML.COM: http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html

[6] OASIS. (12 Октомври, 2006a). *Reference Model for Service Oriented Architecture 1.0*. Изтеглено на 27 Октомври 2007г. от www.oasis-open.org : http://docs.oasis-open.org/soa-rm/v1.0

[7] The Open Group. (2 Юни, 2006). *Service-Oriented Architecture (SOA)*. Изтеглено на 27 Октомври 2007г от www.opengroup.org : http://www.opengroup.org/projects/soa/doc.tpl?gdid=10632

[8] Microsoft. (1 Декември, 2006). *Learn About Service Oriented Architecture (SOA)*. Изтеглено на 27 Октомври 2007г от www.Microsoft.com : http://www.microsoft.com/biztalk/solutions/soa/overview.mspx

[9] W3C. (27 Април, 2007a). *Simple Object Access Protocol (SOAP) 1.2*. Изтеглено на 24 Ноември 2007г от www.w3.org: http://www.w3.org/TR/soap/

[10] W3C. (15 Март, 2001). *Web Services Description Language (WSDL) 1.1*. Изтеглено на 24 Ноември 2007г от www.w3.org: http://www.w3.org/TR/wsdl

[11] W3C. (30 Август, 2002) *W3C XML Schema Definition Language (XSDL) 1.1 Part 1: Structures*. Изтеглено на 24 Ноември 2007г от www.w3.org : http://www.w3.org/TR/xmlschema11-1/

[12] W3C. (17 Февруари, 2006). *XML Schema 1.1 Part 2: Datatypes*. Изтеглено на 24 Ноември 2007г от www.w3.org : http://www.w3.org/TR/xmlschema11-2/

[13] RELAX NG (н.д.). *RELAX NG Description*. Изтеглено на 24 Ноември 2007г от http://www.relaxng.org/

[14] OASIS. (1 Февруари, 2005a). *eXtensible Access Control Markup Language (XACML) Version 2.0*. Изтеглено на 24 Ноември 2007г от www.oasis-open.org : http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

[15] W3C. (16 Ноември, 1999a*) XSL Transformations (XSLT) Version 1.0*. Изтеглено на 24 Ноември 2007г от www.w3.org : http://www.w3.org/TR/xslt

[16] http://www.techterms.com/definition/algorithm

[17] X. Bai, W. Dong, W.-T. Tsai, and Y. Chen, “WSDL-based automatic test case generation for web services testing,” in SOSE 2005: Proceedings of the IEEE International Workshop on Service-Oriented System Engineering, pp. 207–212, Beijing, China, Oct. 2005, IEEE Computer Society

[18] C. Bartolini, A. Bertolino, E. Marchetti, and A. Polini, “WS-TAXI: A WSDL-based testing tool for web services,” in ICST ’09: Proceedings of the International Conference on Software Testing Verification and Validation, pp. 326–335, Denver, Colorado, USA, 2009, IEEE Computer Society.

[19] A. Bertolino, J. Gao, E. Marchetti, and A. Polini, “Automatic test data generation for XML Schemabased partition testing,” in AST ’07: Proceedings of the 2nd International Workshop on Automation of Software Test, pp. 4–4, Minneapolis, Minnesota, USA, May 2007, IEEE Computer Society.

[20] S. Hanna and M. Munro, “Fault-based web services testing,” in ITGN: 5th International Conference on Information Technology: New Generations (ITNG 2008), pp. 471–476, Las Vegas, NV, USA, April 2008, IEEE Computer Society.

[21] Z. J. Li, J. Zhu, L.-J. Zhang, and N. Mitsumori, “Towards a practical and e ective method for web services test case generation,” in Proceedings of the ICSE Workshop on Automation of Software Test (AST’09), pp. 106–114, May 2009.

[22] C. Ma, C. Du, T. Zhang, F. Hu, and X. Cai, “WSDL-based automated test data generation for web service,” in CSSE ’08: Proceedings of the 2008 International Conference on Computer Science and Software Engineering, pp. 731–737,Wuhan, China, Dec. 2008, IEEE Computer Society.

[23] J.Offutt and W. Xu, “Generating test cases for web services using data perturbation,” ACM SIGSOFT Software Engineering Notes, vol. 29, no. 5, pp. 1–10, 2004.

[24] H. M. Sneed and S. Huang, “WSDLTest - a tool for testing web services,” in WSE ’06: Proceedings of the Eighth IEEE International Symposium on Web Site Evolution, pp. 14–21, Philadelphia, PA, USA, Sept. 2006, IEEE Computer Society.

[25] T.J. Ostrand and M.J. Balcer, “The Category-Partition Method for Specifying and Generating Functional Tests,” Comm. ACM, vol. 31, no. 6, June 1988.

[26] Z. J. Li, J. Zhu, L.-J. Zhang, and N. Mitsumori, “Towards a practical and e ective method for web services test case generation,” in Proceedings of the ICSE Workshop on Automation of Software Test (AST’09), pp. 106–114, May 2009.

[27] Knuth D. Art of Computer Programming, Volume 1: Fundamental Algorithms, Addison-Wesley, 1969.

VII. Публикации по темите на хабилитационния труд

No	ID	Публикация
I	23	**Ilieva S.,** P. Ivanov, E. Stefanova, (2004) Analyses of an agile methodology implementation, Proceedings of the 30th EUROMICRO Conference, pp. 326-333, Rennes, France, August 31-September 3, 2004, pp. 326-333
II	25	Ivanov P., **S. Ilieva,** (2005) Gap analyses between certified SME's process and an agile approach, International conference Software Engineering’2005, Innsbruck, Austria, February, 2005, pp. 220-225
III	26	Ivanov P., **S. Ilieva,** (2005) Agile Software Development - Two Industry Companies Experience, International Conference on Agile Manufacturing - ICAM 2005, Helsinki, Finland, 27-28 July, 2005, pp. 209-215
IV	30	Krasteva I., I. Manova, **S. Ilieva,** (2006) Methodology for Automated Functional Testing of Internet Applications, Proceedings of 20th International Conference on Systems for Automation of Engineering and Research (SAER’2006), pp. 89-97, Varna - St. Konstantin resort, 24-26 September, 2006
V	31	**Илиева С.,** В. Лилов, И. Манова. (2006) Изграждане на софтуерни приложения, Университетско издателство “Св. Кл. Охридски”, ISBN-13: 978-954-07-2452-2, 2006 (книга)
VI	32	Ivanov P., **S. Ilieva.** (2007) Agile Case Study Evaluation in Middle Size Project, Serdica Journal of Computing, issue #3, pp 241-254, 2007
VII	33	Dimov A., **S. Ilieva**. (2007) Generalized Nets model of the software architecture of ARCADE, Serdica Journal of Computing, issue #3, pp 255-266, 2007
VIII	34	Dimov A., **S. Ilieva**. (2007) Reliability Models in Architecture Description Language, Int. Conference on Computer Systems and Technologies - CompSysTech’07, University of Rousse, Bulgaria, pp. II.11-1 to II.11-6, 14-15 June 2007
IX	35	Nikolov R., **S. Ilieva,** (2007) Education in Informatics at Sofia University - current status and future plans, Electronic Journal Innovation in Teaching And Learning in Information and Computer Sciences (ITALICS), Volume 6, Issue 3, June 2007, pp. 65-77, http://www.ics.heacademy.ac.uk/italics/vol6iss3.htm

X 36 Krasteva I., **S. Ilieva**. (2007) Rush into Agile - Analytical framework for agile practices applicability, Proceedings of the The IET International Conference on Agile Manufacturing (ICAM 2007), Durham, UK, pp 229-237, 9 - 11 July 2007

XI 37 Nikolov R., **S. Ilieva**. (2007) Building a research university ecosystem: the case of software engineering education at Sofia University. In Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (ESEC-FSE '07). ACM, New York, pp. 491-500

XII 39 Dimov A., **S. Ilieva**. (2007) Modeling of Component-Based Software Systems with Generalized Nets, International Conference on Information technologies (Info-Tech 2007), St. Constantine and Elena resort, Bulgaria, pp 221-230, 21-22 September 2007

XIII 40 Ribarov L., I. Manova, **S. Ilieva.** (2007) Testing in a service-oriented world, International Conference on Information technologies (Info-Tech 2007), St. Constantine and Elena resort, Bulgaria, pp. 23-32, 21-22 September 2007, ISBN:

XIV 41 Krasteva I., **S. Ilieva.** (2008) Adopting an Agile Methodology - Why It Didn't Work, Proceedings of the International Conference on Software Engineering (ICSE'08), APSO "Scrutinizing Agile Practices: Or Shootout at the Agile Corral" workshop, Leipzig, Germany, May 2008, pp. 33-36, ISBN: 978-1-60558-194-1

XV 42 Nikolov R., **S. Ilieva**. (2008) A Model for Strengthening the Software Engineering Research Capacity, Proceedings of the International Conference on Software Engineering (ICSE'08), SEESE workshop, Leipzig, Germany, May 2008, ISBN: 978-1-60558-194-1, pp. 107-115

XVI 46 Krasteva, I., I. Manova, **S. Ilieva**. (2008) Une approche du test des applications SOS, Genie Logiciel, No 87, December 2008, pp 37-44, in french, Testing Approach for SOA E-Government Application, Proceedings of 21st International Conference on Software & Systems Engineering and their Applications (ICSSEA 2008), Paris, France, 9-11 December 2008, pp. 37-44

XVII 49 Dzhurov Y., I. Krasteva, **S. Ilieva,** (2009) Personal Extreme Programming – An Agile Process for Autonomous Developers, International Conference SOFTWARE, SERVICES & SEMANTIC TECHNOLOGIES (S3T), Sofia, Bulgaria, October 28-29, 2009, pp. 252-259

XVIII 50 Iliev M., I. Krasteva, **S. Ilieva,** (2009) A Case Study on the Adoption of Measurable Agile Software Development Process, International Conference SOFTWARE, SERVICES & SEMANTIC TECHNOLOGIES (S3T), Sofia, Bulgaria, October 28-29, 2009, pp. 152-159

XIX 51 **Ilieva S.,** V. Pavlov, I. Manova, (2010) A Composable Framework for Test Automation of Service-Based Applications. 7th International Conference on the Quality of Information and Communications Technology (QUATIC), IEEE Computer Society Washington Publisher, Porto, Portugal, (2010). ISBN:978-0-7695-4241-6, pp.286-291

XX 52 Pavlov V., B. Borisov, **S. Ilieva,** D. Petrova-Antonova, (2010) Framework for Testing Service Compositions. Proc. of the 1st Workshop on Software Services (WOSS'2010) within 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'2010), Timisoara, Romania, October 2010, Romania, pp. 557-560

XXI	55	Krasteva I., **S. Ilieva,** (2010) Experience-Based Approach for Adoption of Agile Practices in Software Development Projects, 22-nd Conference on Advanced Information Systems Engineering CAISE-10, Springer Lecture Notes in Computer Science, Volume 6051, ISBN-13: 978-3-642-13093-9, Hammamet, Tunisia, 9-11 June, pp. 266-280
XXII	57	**Илиева С.**, В. Лилов, И. Манова. (2010) Подходи и методи за реализация на софтуерни системи, Университетско издателство "Св. Кл. Охридски", ISBN: 978-954-07-2999-2, 2010 (книга)
XXIII	60	Manova D., **S. Ilieva,** F. Lonetti, A. Bertolino, C. Bartolini. (2011) Towards automated robustness testing of BPEL orchestrators. In Proceedings of the 12th International Conference on Computer Systems and Technologies (CompSysTech '11), 16-17 June 2011. Boris Rachev and Angel Smrikarov (Eds.). ACM, New York, NY, USA, pp. 659-664
XXIV	62	Spassov I., V. Pavlov, D. Petrova-Antonova, **S. Ilieva** (2011) DDAT: data dependency analysis tool for web service business processes. In Proceedings of the 2011 international conference on Computational science and Its applications - Volume Part V (ICCSA'11), Beniamino Murgante, Osvaldo Gervasi, Andrés Iglesias, David Taniar, and Bernady O. Apduhan (Eds.), Vol. Part V. Springer-Verlag, Berlin, Heidelberg, 232-243
XXV	67	**Ilieva S.,** Manova D., Manova I., Bartolini C., Bertolino A., Lonetti F. (2011) An automated approach to robustness testing of BPEL orchestrations, IEEE 6th International Symposium on Service Oriented System Engineering (SOSE'11), 12-14 December 2011, Irvine, CA, USA, print ISBN 978-1-4673-0411-5, pp. 193-203
XXVI	68	**Ilieva S.,** Manova I., Pavlov V. (2012) Framework for Design Time Testing of Service Based Applications at BPEL level, Serdica Journal of Computing, 2012

VII. Съществени цитирания на публикациите, включени в хабилитационния труд

Статията:

***Ilieva S.**, P. Ivanov, E. Stefanova (2004). Analyses of an agile methodology implementation, Proceedings of the 30th EUROMICRO Conference, pp. 326-333, Rennes, France, August31-September 3, 2004*

е цитирана в:

1. Alves Nélio Muniz Mendes, Integração de Princípios de Desenvolvimento Ágil de Software ao RUP - Um Estudo Empírico, Ano de Obtenção, Disertation, Universidade Federal de Uberlândia, UFU, Brasil
2. Armbrust Ove , Dieter Rombach (2011). The Right Process For Each Context: Objective Evidence Needed, Proceedings of the ICSSP'11, May 21–22, 2011, Waikiki, Honolulu, HI, USA. ACM 978-1-4503-0580-8/11/05
3. Asnawi, A. L., A. Gravell, G. Wills (2010). An Empirical Study: Understanding Factors and Barriers for Implementing Agile Methods in Malaysia. In: 5th International Doctoral Symposium on Empirical Software Engineering (IDoESE), 15 September 2010, Bolzano-Bozen Italy
4. Auvinen J., R. Back, J. Heidenberg, P. Hirkman, L. Milovanov (2005). Improving the Engineering Process Area at Ericsson with Agile Practices. A Case Study, TUCS (Turku Centre for Computer Science) Technical Report, No 716, October 2005
5. Auvinen J., R. Back, J. Heidenberg, P. Hirkman, L. Milovanov (2006). Software Process Improvement with Agile Practices in a Large Telecom Company, Lecture notes in Computer Science, Product-Focused Software Process Improvement, Springer Berlin / Heidelberg, Volume 4034.
6. Back Ralph-Johan, L. Milovanov, I. Porres (2005). Software Development and Experimentation in an Academic Environment: The Gaudi Experience, Lecture notes in Computer Science, Springer Berlin / Heidelberg, Volume 3547.
7. Back Ralph-Johan, L. Milovanov, I. Porres (2007). Software Development and Experimentation in an Academic Environment: The Gaudí Factory, Journal of Systems and Software, Elsevier, April 2007
8. Capaldo Guido , Pierluigi Rippa, Valerio Teta (2008). Effects of technological change on the skills and competencies of software development professionals: a case study on the transition from ABAP to JAVA, International Journal of Information Technology and Management, Volume 7, Number 4 / 2008, pp 440 – 451
9. Carvalho William Chaves de Souza , Pedro Frosi Rosa, Michel dos Santos Soares (2011). A hybrid approach to integrate agile and traditional

software development processes, The Jornadas Chilenas de Computación (JCC), November 2011

10. Dzamashvili Fogelström N., T. Gorschek, M. Svahnberg, P. Olsson (2010). The impact of agile principles on market-driven software product development, Journal of Software Maintenance and Evolution: Research and Practice, Volume 22, Issue 1, pages 53–80, January 2010
11. González Pilar Rodríguez, ESTUDIO DE LA APLICACIÓN DE METODOLOGÍAS ÁGILES PARA LA EVOLUCIÓN DE PRODUCTOS SOFTWARE, Master thesis, UNIVERSIDAD POLITÉCNICA DE MADRID, 2008
12. Jingyue Li, Nils B. Moe, Tore Duba (2010). Transition from a plan-driven process to Scrum: a longitudinal case study on software quality, Proceeding ESEM '10 Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ACM New York, NY, USA ©2010 ISBN: 978-1-4503-0039-1
13. Kim S., S. Hwang, S. Song (2007). An Empirical Analysis on the Effects of Agile practices on Motivation and Work Performance of Software Developers, 2007
14. Kocher Geeta, Mehta N., ANALYSIS AND DESIGN OF A NOVEL AGILE METHODOLOGY, International Journal of Research in IT & Management, Volume 2, Issue 2 (February 2012)(ISSN 2231-4334)
15. Korhonen K. (2010). Exploring Defect Data, Quality and Engagement during Agile Transformation at a Large Multisite Organization, AGILE PROCESSES IN SOFTWARE ENGINEERING AND EXTREME PROGRAMMING, Lecture Notes in Business Information Processing, 2010, Volume 48, Part 1, 88-102, DOI: 10.1007/978-3-642-13054-0_7
16. Korhonen, K (2010). Evaluating the Effect of Agile Methods on Software Defect Data and Defect Reporting Practices - A Case Study, This paper appears in: Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the Issue Date: Sept. 29 2010-Oct. 2 2010 On page(s): 35 - 43 Location: Porto Print ISBN: 978-1-4244-8539-0 References Cited: 29 INSPEC Accession Number: 11679296 Digital Object Identifier: 10.1109/QUATIC.2010.18
17. Kuirinlahti, Hanna, Scrum- ja XP-käytänteiden käyttö haastattelututkimus, Jyväskylä: Jyväskylän yliopisto, 2011, 166 p
18. Li Jingyue, Nils B. Moe, and Tore Dybå. 2010. Transition from a plan-driven process to Scrum: a longitudinal case study on software quality. In Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '10). ACM, New York, NY, USA, , Article 13 , 10 pages. DOI=10.1145/1852786.1852804
19. Malik F. Saleh (2011) An Agile Software Development Framework, International Journal of Software Engineering (IJSE), Volume (2) : Issue (5) : 2011, pp 97 – 106, November-December 2011, Publisher CSC Journals, Kuala Lumpur, Malaysia, ISSN (Online): 2180-1320

20. Martínez L., A. Rodríguez-Díaz, G. Licea and J. Castro (2010). Agile Documents: Toward Successful Creation of Effective Documentation, AGILE PROCESSES IN SOFTWARE ENGINEERING AND EXTREME PROGRAMMING, Lecture Notes in Business Information Processing, 2010, Volume 48, Part 2, 196-201, DOI: 10.1007/978-3-642-13054-0_18
21. Maruping, Likoebe M; Zhang, Xiaojun; Venkatesh, Viswanath (2009). Role of collective ownership and coding standards in coordinating expertise in software project teams, European Journal of Information Systems, Volume 18, Number 4, August 2009, pp. 355-371(17), Publisher: Palgrave Macmillan
22. Mazni, Omar, Sharifah-Lailee, Syed-Abdullah, Azman, Yasin, Agile Documents: Toward Successful Creation of Effective Documentation, Agile Processes in Software Engineering and Extreme Programming, Springer Berlin Heidelberg, Isbn: 978-3-642-13054-0, 2010, pp 196-201
23. Meerbaum–Salant O., O Hazzan (2010). An Agile Constructionist Mentoring Methodology for Software Projects in the High School, Journal ACM Transactions on Computing Education (TOCE), Volume 9, Issue 4, Article No.: 21, January 2010
24. Melo, C.O.; Ferreira, G.R.M.; ,Adoção de métodos ágeis em uma Instituição Pública de grande porte-um estudo de caso,Mensagem do Coordenador de Programa do WBMA,,,112,2010,
25. Montenegro Rueda, Gema y Gómez del Castillo, Arturo, Estudio de metodologías ágiles y aplicación de extreme programming en el desarrollo de una aplicación para la gestión de proyectos ágiles, Estudio de metodologías ágiles y aplicación de extreme programming en el desarrollo de una aplicación para la gestión de proyectos ágiles. Proyecto Fin de Carrera,E.U. de Informática (UPM), Madrid, España
26. Overhage, S., Schlauderer, S., Birkmeier, D., Miller, J. (2010). On the Developer Adoption of Scrum: A New Acceptance Model for Agile Methodologies, Proceedings of JAIS Theory Development Workshop . Sprouts: Working Papers on Information Systems, 10(76). http://sprouts.aisnet.org/10-76
27. Petersen K., C. Wohlin (2008). Issues and Advantages of Using Agile and Incremental Practices, Software Engineering Research and Practice, 2008
28. Petersen K., C. Wohlin (2010). The effect of moving from a plan-driven to an incremental software development approach with agile practices - An industrial case study, EMPIRICAL SOFTWARE ENGINEERING Journal, Volume 15, Number 6, 654-693, DOI: 10.1007/s10664-010-9136-6, ISSN 1382-3256
29. Petersen Kai , Claes Wohlin, A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case, Journal of Systems and Software, Volume 82, Issue 9, September 2009, Pages 1479-1490, ISSN 0164-1212, 10.1016/j.jss.2009.03.036.

30. Saleh Malik F., An Agile Software Development Framework, International Journal of Software Engineering, November / December 2011, ISSN: 2180-1320, pp 97-106

31. Tarhan Ayça, Seda Güneş Yılmaz, Çevik Süreç ile Artırımsal Sürecin Nicel Karşılaştırması: Bir Durum Çalışması, 5. ULUSAL YAZILIM MÜHENDİSLİĞİ SEMPOZYUMU - UYMS'11, pp 9-16

32. Santos Mariana de Azevedo , Paulo Henrique de Souza Bermejo, Adriano Olímpio Tonelli and André Luiz Zambalde (2011). Challenges of Teams Management: Using Agile Methods to Solve the Common Problems, ENTERPRISE INFORMATION SYSTEMS Communications in Computer and Information Science, 2011, Volume 220, Part 4, 297-305, DOI: 10.1007/978-3-642-24355-4_30

33. Senapathi M. (2010). Adoption of Software Engineering Process Innovations: The Case of Agile Software Development Methodologies, AGILE PROCESSES IN SOFTWARE ENGINEERING AND EXTREME PROGRAMMING, Lecture Notes in Business Information Processing, 2010, Volume 48, Part 2, 226-231, DOI: 10.1007/978-3-642-13054-0_23

34. Sfetsos, P., I. Stamelos (2010). Empirical Studies on Quality in Agile Practices: A Systematic Literature Review, This paper appears in: Proc. of the Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the Issue Date: Sept. 29 2010-Oct. 2 2010 On page(s): 44 - 53 Location: Porto Print ISBN: 978-1-4244-8539-0 References Cited: 68 INSPEC Accession Number: 11688355 Digital Object Identifier: 10.1109/QUATIC.2010.17 Date of Current Version: 03 December 2010

35. Stålhane T., G. Hanssen (2008). The application of ISO 9001 to agile software development, Product-Focused Software Process Improvement, 9th International Conference, PROFES 2008, Monte Porzio Catone, Italy, June 23-25, 2008, Proceedings, Lecture notes in Computer Science, Springer Berlin / Heidelberg, Volume 5089.

36. 황순삼, 김성근. 2010. "개발방법론이 개발직무의 동기유발성에 미치는 영향에 대한 분석.", Journal of Information Technology Applications & Management, 17(2): 1-18. http://dbpia.co.kr/view/ar_view.asp?arid=1467941

Статията:

*Ribarov L., I. Manova, **S. Ilieva** (2007). Testing in a service-oriented world, International Conference on Information technologies (Info-Tech 2007), St. Constantine and Elena resort, Bulgaria, pp. 23-32, 21-22 September 2007, ISBN*

е цитирана в:

37. ALAI N GEORGES VOUFFO FEUDJ. O, INA SCHI EFERDECKER (2009). Availability Testing for Web Services, Telektronikk Volume 1,

Telenor ASA 2009

38. Ayaz Farooq, Konstantina Georgieva and Reiner R. Dumke (2008). Challenges in Evaluating SOA Test Processes, *Lecture Notes in Computer Science, Springer* Berlin / Heidelberg, book Software Process and Product Measurement, Volume 5338/2008,
39. A.-G. Vouffo Feudjio (2009). Model-driven functional test engineering for service centric systems, tridentcom, pp.1-7, 2009 5th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities and Workshops, 2009, ISBN: 978-1-4244-2846-5
40. Bo Yang; Ji Wu; Chao Liu; Luo Xu;, A Regression Testing Method for Composite Web Service, International Conference on Biomedical Engineering and Computer Science (ICBECS), 2010, ISBN: 978-1-4244-5315-3
41. Steve Bridges and Danielle Mackenzie, Kathleen Powers and Jonathan Snyder (2009). Capabilities in Context: Evaluating the Net-Centric Enterprise, ITEA Journal 2009; 30: 57–64
42. Kabbani, N.; Tilley, S.; , Evaluating the capabilities of SOA security testing tools, IEEE International Systems Conference (SysCon), 2011, pp 129-134, Montreal, CA
43. Sander Triebert (2005). Centralized and Automated Testing within a Maintenance Environment, Master Thesis, *Delft University of Technology*, 2005
44. Bo Yang, Ji Wu, Chao Liu, Luo Xu (2010). A Regression Testing Method for Composite Web Service, *Proc. of the Biomedical Engineering and Computer Science (ICBECS), 2010 International Conference on* 23-25 April 2010 On page(s): 1 - 4 Wuhan ISBN: 978-1-4244-5315-3 Cited: 13 INSPEC Accession Number: 11292299 Digital Object Identifier: 10.1109/ICBECS.2010.5462461
45. Yinsheng Li, Han Chen, Mu Zhu, Jen-Yao Chung (2010). Evaluating a Service-Oriented Travel Portal, *Proc. of the Fifth IEEE International Symposium on Service Oriented System Engineering* (SOSE'10), pp. 229-235, 2010
46. Kabbani, N.; Tilley, S. (2011). Evaluating the capabilities of SOA security testing tools, *Proc. of the IEEE International Systems Conference (SysCon'2011), 2011* Issue Date: 4-7 April 2011 On page(s): 129 - ISBN: 978-1-4244-9494-1 □References Cited: 17 □INSPEC Accession Number: 12075832 □Digital Object Identifier: 10.1109/SYSCON.2011.5929125
47. Muhammad Suhaizan Sulong, Andy Koronios, Jing Gao, Azlianor Abdul-Aziz (2011). Information Quality Activities WITHIN THE INDUSTRY INITIATIVE OF SERVICE-ORIENTED ARCHITECTURE IMPLEMENTATION (Research-in-progress), *Proceedings of the 16th International Conference on Information Quality (ICIQ-11)*, November 2011, MIT.edu

Статията:

*Krasteva I., **S. Ilieva**. 2008. Adopting an agile methodology: why it did not work. In Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral (APOS '08). ACM, New York, NY, USA, 33-36. DOI=10.1145/1370143.1370150 http://doi.acm.org/10.1145/1370143.1370150*

е цитирана в:

48. Ani Liza Asnawi, Andrew M. Gravell and Gary B. Wills (2011). Empirical Investigation on Agile Methods Usage: Issues Identified from Early Adopters in Malaysia, Agile Processes in Software Engineering and Extreme Programming, Lecture Notes in Business Information Processing, 2011, Volume 77, Part 1, 192-207, DOI: 10.1007/978-3-642-20677-1_14
49. Asnawi, A. L., Gravell, A. and Wills, G. (2010) An Empirical Study: Understanding Factors and Barriers for Implementing Agile Methods in Malaysia. In: 5th International Doctoral Symposium on Empirical Software Engineering (IDoESE), 15 September 2010, Bolzano-Bozen Italy
50. Jonas Schild, Timo Göttel, Paul Grimm (2011). Game Development Inhalte in der Hochschulinformatik, Workshop-Proceedings der Tagung Mensch & Computer 2011 überMEDIEN|ÜBERmorgen, Universitätsverlag Chemnitz 2011, Maximilian Eibl, Marc Ritter (Hrsg.), Technische Universität Chemnitz/Universitätsbibliothek, Universitätsverlag Chemnitz, pp 385-390, ISBN 978-3-941003-38-5
51. Pohjonen, Antti, Test automation scheme for LTE core network element, Lappeenranta University of Technology, PhD Thesis, 2010
52. Vesa Kettunen, Jussi Kasurinen, Ossi Taipale, and Kari Smolander. 2010. A study on agility and testing processes in software organizations. In Proceedings of the 19th international symposium on Software testing and analysis (ISSTA '10). ACM, New York, 231-240. DOI=10.1145/1831708.1831737 http://doi.acm.org/10.1145/1831708.1831737

Статията:

*Krasteva I., **S. Ilieva**, A. Dimov (2010) Experience-Based Approach for Adoption of Agile Practices in Software Development Projects*, ADVANCED INFORMATION SYSTEMS ENGINEERING, Lecture Notes in Computer Science, 2010, Volume 6051/2010, 266-280, DOI: 10.1007/978-3-642-13094-6_22

е цитирана в:

53. Hesam Chiniforooshan Esfahani, Eric Yu, and Maria Carmela Annosi (2011). Strategically balanced process adoption. In *Proceedings of the 2011 International Conference on on Software and Systems Process* (ICSSP '11). ACM, New York, 169-178. DOI=10.1145/1987875.1987902 http://doi.acm.org/10.1145/1987875.1987902
54. Ruiz-Rube Iv´an, Juan Manuel Dodero, Mercedes Ruiz and David Gawn

(2011). Uses and Applications of SPEM Process Models. A Systematic Mapping Study, *JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE, J. Softw. Maint. Evol.: Res. Pract*. 2011; 00:1–29, Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/smr

Статията:

*Krasteva I., **S. Ilieva**. 2007. Rush into Agile - Analytical framework for agile practices applicability, Proceedings of the The IET International Conference on Agile Manufacturing (ICAM 2007), Durham, UK, pp 229-237, 9 - 11 July 2007*

е цитирана в:

55. Christopher Thomson, Mike Holcome, Tony Cowling, Tony Simons, and George Michaelides. 2008. A pilot study of comparative customer comprehension between extreme x-machine and uml models. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement* (ESEM '08). ACM, New York, NY, USA, 270-272. DOI=10.1145/1414004.1414048 http://doi.acm.org/10.1145/1414004.1414048

Статията:

***Ilieva S**., V. Pavlov, I. Manova. 2010. A Composable Framework for Test Automation of Service-Based Applications. 7th International Conference on the Quality of Information and Communications Technology (QUATIC), IEEE Computer Society Washington Publisher, pp.286-291, Porto, Portugal, (2010). ISBN:978-0-7695-4241-6*

е цитирана в:

56. Mustafa Bozkurt, Mark Harman, Youssef Hassoun,Testing & Verification In Service-Oriented Architecture: A Survey, Journal Software Testing, Verification and Reliability (STVR), To Appear 2012
57. M. Todorova. Verification of Procedural Programs via Building there Generalized Nets Models. Proceedings of the 41. Spring Conference of the Union of Bulgarian Mathematicians, MATHEMATICS AND EDUCATION IN MATHEMATICS, 2012.
58. М. Тодорова. 2011. Построяване на коректни обектно-ориентирани програми чрез изграждане на техни обобщени мрежови модели, Конференция на съюза на учените в България, секция Информатика, Годишник на секция “Информатика”, Съюз на учените в България, ISSN 1313-6852 том 4, 2011 (1-28).
59. M. Todorova, Methodological Aspects of an Approach for Verification of Object-Oriented Programs, 6th IEEE International Conference Intelligence Systems 2012, Sofia, 6-8 sept., 2012 (приета за печат).

Статията:

*Pavlov V., B. Borisov, **S. Ilieva**, D. Petrova-Antonova. 2010. Framework for Testing Service Compositions. Proc. of the **1st Workshop on Software Services** (WOSS'2010) within 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'2010), Timisoara, Romania, October 2010, pp. 557-560*

е цитирана в:

60. М. Тодорова. 2011. Построяване на коректни обектно-ориентирани програми чрез изграждане на техни обобщени мрежови модели, Конференция на съюза на учените в България, секция Информатика, Годишник на секция "Информатика", Съюз на учените в България, ISSN 1313-6852 том 4, 2011 (1-28).
61. M. Todorova, Methodological Aspects of an Approach for Verification of Object-Oriented Programs, 6th IEEE International Conference Intelligence Systems 2012, Sofia, 6-8 sept., 2012 (приета за печат).

Статията:

*Petrova-Antonova D., I. Krasteva, **S. Ilieva**. 2010. Approaches Facilitating WS-BPEL Testing.,17th Conference on European Systems & Software Process Improvement and Innovation (EuroSPI 2010), Grenoble Institute of Technology, pp. 5.1-5.17, France, September 2010,*

е цитирана в:

62. M. Todorova. Verification of Procedural Programs via Building there Generalized Nets Models. Proceedings of the 41. Spring Conference of the Union of Bulgarian Mathematicians, MATHEMATICS AND EDUCATION IN MATHEMATICS, 2012.