Модели, базирани на йерархични композиции от пространства, за управление на софтуерни версии

**Тίποτε δεν αντέχει στο χρόνο όσο η αλλαγή** Няма нищо постоянно, с изключение на промяната.   
Херакъл 500 г. пр. Хр.

Чернова № 1017

Дата: 2012-01-07

Съдържание

[УВОД 5](#_Toc312755893)

[Методология на изследването 6](#_Toc312755894)

[Глава първа Управление на версията при създаването на софтуерни системи 7](#_Toc312755895)

[1.1. Място на управлението на версии 7](#_Toc312755896)

[1.1.1. Методологии създаване на софтуер 7](#_Toc312755897)

[Управление на документни 10](#_Toc312755898)

[1.1.2. Цели, задачи и предизвикателства пред системите за контрол и управление на версии 12](#_Toc312755899)

[1.1.3. Управление на версиите при създаването на софтуер 16](#_Toc312755900)

[1.2. Модели на управление и контрол на версиите 17](#_Toc312755901)

[1.2.1. Модел на извличане/записване 18](#_Toc312755902)

[1.2.2. Композиционен модел 18](#_Toc312755903)

[1.2.3. Модел на версионизиране чрез използване на дълги транзакции 19](#_Toc312755904)

[1.2.4. Версионизиране чрез набор от промени 20](#_Toc312755905)

[1.2.5. Обектно-ориентиран модел за контрол на версии 20](#_Toc312755906)

[1.3. Модели на версионизиран обект 21](#_Toc312755907)

[1.3.1. Файлово базиран модел 22](#_Toc312755908)

[1.3.2. Обектно-ориентиран модел 23](#_Toc312755909)

[1.3.3. Съхраняване на промени 23](#_Toc312755910)

[1.3.4. Интеграция между разклонени версии 27](#_Toc312755911)

[1.3.5. Съставност и гранулираност на версионизирани обекти 29](#_Toc312755912)

[1.4. Съвместна работа и работни пространства 29](#_Toc312755913)

[1.4.1. Модели за осигуряване на конкурентен/паралелен достъп 30](#_Toc312755914)

[1.4.2. Същност на работното пространство 33](#_Toc312755915)

[1.4.3. Файлово базирани работни пространства. 34](#_Toc312755916)

[1.4.4. Модел на работни пространства със съхранена версия 36](#_Toc312755917)

[1.4.5. Модели на йерархично композиране на работни пространства 38](#_Toc312755918)

[1.5. Методи за проследимост на промените 39](#_Toc312755919)

[1.5.1. Същност на проследимостта на промените... 39](#_Toc312755920)

[1.5.2. Проследяване на промените 40](#_Toc312755921)

[1.5.3. Управление на изискванията и исканията за промени 43](#_Toc312755922)

[1.5.4. Методи за проследяване на промени 43](#_Toc312755923)

[1.6. Изводи 47](#_Toc312755924)

[Глава втора Модели за управление на версии в среда с йерархична композиция на работни пространства 49](#_Toc312755925)

[2.1. Модел на версионизиран обект 49](#_Toc312755926)

[2.1.1. Версионизиране на съставен версионизиран обект 54](#_Toc312755927)

[2.1.2. Референция към обект 56](#_Toc312755928)

[2.1.3. Примерен ER модел на същността „клас” 57](#_Toc312755929)

[2.2. Йерархично композирани работни пространства. Модел на видимост на версионизирани обекти 60](#_Toc312755930)

[2.2.1. Модел на йерархично композирани работни пространства 60](#_Toc312755931)

[2.2.2. Модел на видимост на версионизирани обекти в среда с йерархично композиране на работни пространства 60](#_Toc312755932)

[2.3. Транзакции над версионизиран обекти 62](#_Toc312755933)

[2.3.1. Транзакции над версионизиран обект в рамките на едно работно пространство 62](#_Toc312755934)

[2.3.2. Транзакции над версионизиран обект между две работни пространства 63](#_Toc312755935)

[2.3.3. Транзакции над съставни обекти 66](#_Toc312755936)

[2.3.4. Класификация на транзакциите над версионизирани обекти 69](#_Toc312755937)

[2.3.5. Жизнен цикъл на версионизиран обект 71](#_Toc312755938)

[2.4. Проследимост на промените в среда с йерархична композиция на работни пространства 72](#_Toc312755939)

[2.4.1. Работни единици и работни пространства 74](#_Toc312755940)

[2.4.2. Модели на данните на система за управление на версията чрез йерархични пространства 75](#_Toc312755941)

[2.5. Изводи 76](#_Toc312755942)

[Глава трета Изследване приложимостта на моделите 78](#_Toc312755943)

[3.1. Възможности за реализиране на моделите 78](#_Toc312755944)

[3.2. Избор на технологии и за реализация 82](#_Toc312755945)

[3.3. Разработка на прототип на система за управление на версии 83](#_Toc312755946)

[3.3.1. Избор на софтуерен инструментариум. 83](#_Toc312755947)

[3.3.2. Архитектурен модел 83](#_Toc312755948)

[3.3.3. Модел на класовете 83](#_Toc312755949)

[3.3.4. Навигационен модел 83](#_Toc312755950)

[3.4. Сравнителен анализ преимущестата на прототипа 83](#_Toc312755951)

[3.5. Изводи 83](#_Toc312755952)

[Заключение 84](#_Toc312755953)

[Литература 85](#_Toc312755954)

# УВОД

Промяната стои в основата на развитието на нашата цивилизация, тя заема главно място в нашето ежедневие. Нейното управление е от голямо значение за успешното осъществяване на всеки проект, в частност и на софтуерните проекти. Особеното при софтуерните проекти е факта, че техните създатели ги реализират с цел да се автоматизира и ускори дадена сфера от човешката дейност. Ето затова именно в областта на създаването на софтуер възникват и първите системи предоставящи автоматизация на процеса по контрол и управление на версията (версионизиране) на изходния код на програмните продукти.

Управлението на версия (версионизирането) представлява процес, предоставящ инфраструктура за достъп, проследимост, контрол и управление на променяеми обекти. Тази инфраструктура заема ключово място при управлението на проекти, както и промяната на процеси, с цел тяхното подобряване.

Първоначално фокусът на тези системи е бил насочен към управление на версията на отделните програмни единици – файловете с изходен код. На по-късен етап възниква необходимостта и се създават системи за управление на версии от по-високо ниво, които предоставят възможност да се управлява крайната версия на продукта и различните му конфигурации в продукция, отговарящи на конкретни нужди на клиента. Тези системи са известни като системи за управление на конфигурации. Белладжио и Миллиган в [G319] определят тези системи като дисциплина от софтуерното инжинерство, включваща средства и методи за управление, отчитане и проследяване на измененията на програмните продукти и техните компоненти.

Софтуерното инженерство (СИ) е дял от информатиката свързан с осигуряване на методичен, инженерен подход при създаването на софтуерни системи […]. Предметът на софтуерното инженерство е тясно свързан с жизненият цикъл на софтуерните системи и включва *управление на изискванията*, *техники и инструменти за проектиране*, *верифициране*, *тестване и осигуряване на качество*, *среди за програмиране*, *разпространение, поддръжка и подобрение на системите*, *управление на проекти*, *метрики на софтуерните системи*, *проектиране и архитектура*, *взаимопроникване и преизползване на софтуерните системи* [ACM]. Системите за управление на версии представляват инфраструктурни системи в процеса на създаване, доставяне, поддръжка и подобряване на системите [...].

Основен **обект на изследване** в дисертацията представлява изследване, развитие и адаптиране на моделите за контрол и управление на версии на софтуерни продукти в среда за разработване базирана на йерархична композиция от работни пространства. терминология

Задачите на СИ в това направление са свързани с надеждното съхранение на артефактите съпътстващи жизненият цикъл на създаваната система.

**Методология на изследването**

При настоящото изследване са използвани ...

Дисертационният труд се състои от увод, три глави, заключение и използвана литература.

В първа глава е направен обзор на

Втора глава представя

Трета глава съдържа изследване за приложимост на предложените във втора глава модели за управление на версии.

В заключението са обобщени получените резултати.

Разработката и апробацията на резултатите са извършени ...

Получените резултати са публикувани в ... доклада в периода след 200.... От тях два броя са в чужбина.

Изказвам благодарност на ...

# Глава първа Управление на версията при създаването на софтуерни системи

Настоящата секция има за цел да представи различни аспекти в областта на управлението на версиите, които косвено са обект на настоящото научно-приложно изследване, тъй като те заемат важно място в съвременните системи за управление на версии, в частност при управлението на жизненият цикъл на софтуерните продукти (SDLC - software development life-cycle). Тук сe разглеждат подходите за съхраняване на състоянията на версионизираните обекти, направен е анализ на предимствата и недостатъците на различните подходи, както и опит за формално сравнение на ефективността им.

Вторият важен момент в обхвата на секцията разглежда моделите за осигуряване на конкурентен достъп до версионизираните обекти, в който е наблегнато на техни особености от практическа гледна точка. Това се обосновава от факта, че много малка част от софтуерните проекти се реализират от един разработчик.

Друг елемент, разглеждан в настоящата глава е пряко свързан с жизненият цикъл на софтуерните продукти, в частност разклоняването и сливането на техни версии. Методите за интеграция на две версии на даден обект имат за цел да се представят разработените подходи.

Не на последно място по важност е направен обзор на методите за осигуряване на проследимост на промените.

## Място на управлението на версии

### Методологии създаване на софтуер

Създаването на софтуерни системи се определя като отрасъл на нематериалното произвоство с голяма степен на интелектуало-емкост. Исторически по естествен начин се е достигнало до факта, че производителите на софтуерни продукти са основните потребители на системите за управление версии. Това предполага висока степен на преизползване на вече създадените системи, модули, компоненти при изграждането на новите системи. Едновременно с преизползването на съществуващите елементи, протича и друг много често срещан процес – процеса на тяхното модифициране с цел да удовлетворят промяната на първоначалните изисквания и/или появата на нови изисквания.

За създаването на софтуерни системи са разработени множество формални методологии – каскадна, спирална (итеративна), в-образна, гъвкави методологии и други [G296]. Многообразието от методологии се обуславя от различните предимства, недостатъци и характеристики, които съдържа всяка една от тях притежава. Допълнително може да се изтъкне че някои автори в своите изследвания правят опит да определят съответствие между големината на проекта, степенна на риск и целесъобразността от използването на една или друга методология [G83]. Въпреки това основните фази, през които по някакъв начин преминава всеки един софтуерен продукт, може да се сведат до следните до следните:

* Анализ и определяне обхвата, правилата, ограниченията и изискванията към системата;
* Определяне на системната архитектура, подходите и технологиите за изграждане на нова или промяна на съществуваща система;
* Създаване на версия на продукта, отговаряща на определен набор изисквания.
* Осъществяване на качествен контрол на системата спрямо изискванията, използваните стандарти и практиките за изграждане на системи;
* Въвеждане в експлоатация на системата в продукционна среда;
* Поддръжка, усъвършенстване и отстраняване на грешки.



Фиг. 1 Управление на версията при каскаден модел



Фиг. 2 Управление на версията при итеративен модел

Разпространението на софтуерните продукти заема важно място в жизненият цикъл на един продукт. Много малко са продуктите, които са поръчани и се използват само от един единствен клиент. Това довежда до необходимостта да се произвеждат такива продукти, които имат унифицирани функционални характеристики, отговарящи на изискванията на повечето клиенти. Въпреки това съществуват допълнителни компоненти, които изрично се поръчват от отделни клиенти, и които не са необходими за работата при останалите клиенти. Това довежда до необходимостта за управление на конфигурациите при доставянето до крайният клиент.

Масовите стандартни софтуерни продукти се разпространяват с лиценз и конфигурация „каквито са”. За тяхното разгръщане съответно се използват стандартизирани разгръщащи програми – инсталатори. Съвременна тенденция е вграждането на средства за автоматична актуализация в програмните продукти. От друга страна сложните и комплексни системи предполагат използването на ръчна процедура на разгръщане на конфигурацията на системата, особено когато нейното разгръщане зависи от инсталирането и конфигурирането на други системи – пощенски сървъри, СУБД и др. Тук следва да се отбележи, че под ръчна процедура се има в предвид ръчното стартиране на инсталационни, конфигурационни и актуализиращи скриптове от специалист.

Терминът **управление на конфигурация** следва да се разбира като синоним на терминът **управление на версия** [G383, 03:26] в рамките на настоящата дисертация.

От историческа гледна точка, преди появата на системите за контрол и управление на версии достъпът до определена версия на програмен продукт се е осъществявал чрез използване на ръчни процедури от областта на библиотекознанието. Те се характеризират с ниска скорост на достъп, тромавост [G319]. Поява на SCCS (Source Code Control System) през 1972 води до автоматизиране и ускорение процеса на версионизиране [G332].

### Управление на документни

Системите за управление на документи, представляват отделен клон на развитие на механизмите за управление на версии. Те предоставят класически механизми за версионизиране, използвани при в системите за управление на изходният код на софтуерните системи. При тяхното управление се акцентира на управлението на мета-атрибутите на документите, като се фокусира над следните аспекти [да сложа малко източници, G349]:

* Управление на мета-атрибути, което включва стандартни данни за документите, като дата на създаване, автор, дата на последно изменение, ключови думи и други. Тези данни се използват за подпомагане намирането на нужният документ в системата.
* Интеграция на документа с други системи, които извличат и/или модифицират отделни негови части.
* Индексиране, което се използва за определяне уникалността на версията на документа. Индексирането се използва също така и при изграждане на индексни топологии [източници].
* Съхранение на документите, като място, политики за съхранение, и други.
* Извличане на документи от репозиторито за съхранение. Някои от системите за управление на документи предоставят възможност да се определи форматът в който да се извлече нужният документ.
* Разпространението и публикуването на документите, което е свързано с механизми по извличането на документа във формат, възпрепятстващ неговата последваща модификация.
* Управление на сигурността и правата на достъп, нужна при работа с документи със сложна структура.
* Поддръжка на работен процес при работа с електронен документооборот, като промяна и контрол на статусите на документа и други.
* Управление на паралелната и съвместна работа по документите.

Най-популярните направления при версионизирането на документи може да се определят както следва:

* Уики системите [G342, G343, G344] предназначени за изграждане на уеб базирано съдържание с цел постигане на кооперираност, управление на знание и други.
* Електронни системи за документо-оборот базиран на правила. Като пример за система от тази група може да се посочи системата за докумето-оборот на платформата Microsoft Sharepoint [L5].
* Системите за управление на съдържание (Content Management Systems) могат да се определят като системи за управление на версията на уеб сайтовете. Интересен момент при тях е, че те предоставят механизми за управление на бъдещата версия на съдържанието на сайт. По този начин може да се извършва автоматично времево планиране активността на отделните страници. Най-популярните системи в това направление са Joomla! [L4], Wordpress[…], Drupal[…].
* Хипертекстови документи. Отдалечената и паралелна работа по хипертекстови документи е повлиано от създаването на протокола WebDav през 1995 – 1996 години от WebDav работната група. Протоколът представлява стандартно разширение на протокола HTTP, като се добавят следните допълнителни възможности [L6]:
  + Управление на мета-данни, съдържащи данни за автора, дата на създаване и др.
  + Управление на областите с имена, позволяваща отделните документи да са съставени от други документи в дървовидна структура, подобно на файловата структура.
  + Превенция на презаписа, състояща се в това само един потребител на системата в даден момент да има възможност да редактира даден документ.
  + Управление на версията на документа, позволяваща по-късно извличане на мажорни версии (издания) на документа.

### Цели, задачи и предизвикателства пред системите за контрол и управление на версии

Целта на системите за управление и контрол на версията е да предостави инструмент на разработчиците на програмни продукти да автоматизират процесите на версионизиране. В [G222] авторът, базирайки се на стандарта IEEE 729-1983, определя функционалният обхват на системите за управление на конфигурации, които могат да се обобщи в следните групи:

* Идентификационна схема, която отразява структурата на програмният продукт, неговите компоненти, както и техните типове.
* Контрол на изданията на продукта, неговите промени. Целта е да се осигури консистентност на компонентите на подукта.
* Актуално състояние, включващо записване и отчитане на статуса на компонентите и исканията за промяна.
* Одит и валидиране на завършеността на продукта и съвместимостта между отделните компоненти.
* Производството, включващо оптималното управление на конструирането на продукта.
* Управление на процеса на осигуряване изпълнението на организационните процедури, политики и модела на жизнен цикъл.
* Съвместна разработка, състояща се в контролиране на работата и взаимодействията между отделните разработчици на продукта.

В своето изследване Дарт [G222] допълнително развива класификацията на задачите, които трябва да изпълняват системите за управление на версиите. В представеният модел се разглеждат осем основни функционални направления, като определя две от тях за водещи. На Фиг. 3 графично е представена класификацията, където под формата на квадрати са илюстрирани основните функционални направления:

* Управление на компонентите, което включва изисквания към функции свързани с идентификацията, класификацията, съхраняването и осигуряването на достъп до отделните компоненти, които изграждат продукта.
* Управление архитектурата на продукта.
* Управление конструирането на продукта.
* Одит, позволяващ да се проследи развитието на продукта, както и прогресът по изграждането му.
* Отчетност, предоставяща възможност за генериране на статистически и други справки свързани с работата и дейностите по създаването на продукта.
* Управление на процеса по създаване на продукта. Тук голямо значение заема и управлението на измененията, анализ на влиянието над продукта, както и тяхната проследимост.
* Управление на екипа, членовете участващи в създаването и поддръжката на продукта, като се прилагат различни профили според ролята на отделният участник в процеса.



Фиг. 3 Класификация на функционалните изисквания към система за управление на измененията[G222] [G12]

Дарт определя като водещи изискванията свързани с работата в екип и тези свързани с поддържката процеса на разработване на продукта.

В групата за поддържане на екипа са включени изискванията за осигуряване автономната работа на членовете на екипа чрез използването на работни пространства. Тук се разглеждат изискванията за разрешаване на конфликтните точки при самото версионизиране, а също така и изисквания относно социалното обединяване на екипа.

Потребителите на системата, които са част от екипа, при създаването на нови продукти, реализират неговото изграждане, структура и дефинирането на компонентите му.

Групата изисквания за изграждането на продукт включват неговата направа, оптимизация. Промяната в първоначалните изисквания е често срещано явление в практиката [G356], затова системата за управление на версията трябва да предоставя функционалност подпомагаща осъществяването на анализ обхвата на влиянието от промяна в изискванията.

От гледна точка на структурата на крайният продукт, се определят изисквания за използване на системен подход от системата за контрол на версията, който да подпомага поддръжката на интерфейси, релации между отделните същности на крайният продукт.

Изискванията към компонентите на един софтуерен продукт, версионизиращата система трябва да поддържа различни версии на различните компоненти, да подпомага конфигурацията на крайният продукт за различните крайни клиенти, имащи специфични изисквания, както и да поддържа версии на тези конфигурации.

Втората главна група изисквания - тези към процеса на разработка на продукта, включват изисквания относно поддръжката на определен жизнен цикъл, предоставянето на механизъм за управление на отделните задачи, комуникацията в рамките на проекта, документирането на продукта. Като области, допълващи процеса, авторът на изследването определя изискванията към одит, отчетност и контрол.

За целите на одита на създаването на софтуерни продукти, системата трябва да предоставя възможности за проследимост историята на продукта, процесът и на дейностите, които са извършените от членовете на екипите.

От гледна точка на отчетността – трябва да се поддържа статистика, статус и отчетност на задачите, които се явяват неотделима част от процеса на създаване на продукта.

Акцентът при контролът е насочен към контролът на достъп, управление исканията за промяна, проследимостта на грешките. Разпространението на промените, сред всички участници в процеса на създаване на продукта, и разделението на сложните задачи на по-малки завършват съвкупността от изисквания към една система за управление на версия.

### Управление на версиите при създаването на софтуер

Исторически...

В своята книга Беллаждио и Миллиган [G319] представят процеса за унифицирано управление на промените (Unified Change Management). Те определят следните участници в процеса на създаване на софтуерни системи: архитект, мениджър управление на конфигурации, мениджър на проект, разработчик, интегратор. На Фиг. 4 е представен процеса на унифицираното управление на промените. Прави впечатление, че в представеният модел липсва фазата на управление на качеството над крайният продукт.

Други слабости на модела... да помисля



Фиг. 4 Унифицирано управление на промените [G319]

## Модели на управление и контрол на версиите

В настоящата секция са представени основните модели, осигуряващи версионизирането на продуктите. На по-ниско ниво различните модели могат да се реализират чрез различните по-горе модели

### Модел на извличане/записване

Моделът на извличане и записване е широко използван в системите UNIX SCCS [G277] и RCS [G148], като от последната система произлизат най-широко използваните CVS/Subversion [G50, G54, G101]. При подхода на извличане/записване се използват две независими среди – репозитарна среда и среда за разработване на софтуер. В репозитарната система са имплементирани механизми за версионизиране на файлови артефакти.

Потребителите на системи използващи този подход нямат директен достъп до репозитарната система. Те първо трябва да извлекат конкретна версия на даден файл и след като го променят имат възможност отново да го запишат в репозиторито. Като недостатък на този подход някои автори [G275] посочват факта, че репозитарната система не управлява работната област и разчита на файловата система да осигури правата на достъп до файловете. От друга страна често инструментите работещи над същата тази файлова система не са съобразени с принципите на версионизиране и избора на желаната версия.

При системите за управление и контрол на версията използващи подхода на извличане/записване потребителите, разработчиците, използват т. нар. работни области, представляващи различни файлови директории. Може да се отбележи, че поддръжката на потребителите в техните работни области е твърде ограничена. Понятието работно пространство се свежда до файлова директория, в която потребителят съхранява извлечените файлове и от която той, след като ги е модифицирал, ги записва в репозиторито.

### Композиционен модел

Композиционният подход представлява естествен еволюционен наследник на подхода извличане/записване. Този подход се базира на употребата на следните основни елементи [G275]:

* модел на системата;
* правила на избор на версия.

В моделът на системата се дефинира структурата на системните компоненти, които ще подлежат на контрол на версия. Чрез правилата на избор на версия се дефинира за конкретната конфигурация за всеки компонент, коя версия да бъде включена. Именно от този подход изследователите започвт да използват термина Системи за управление на конфигурации (SCM). Той може да се разглежда като версионизиране от по-високо ниво, т.е. версионизиране на конфигурация на крайният продукт.



Фиг. 5 Избор на версия на компонент при конфигурация [G275]

Феилер [G275] определя два подхода за конструиране на правила за избор:

* Чрез търсене на път в граф на версии с етикети.
* Чрез предикати над мета-данни на версионизираните обекти.

### Модел на версионизиране чрез използване на дълги транзакции

Транзакцията в рамките на базите от данни представлява поредица от действия/промени над данни, които при определени условия могат се отменят. Основни характеристики на транзакциите са:

* Те са изолирани при конкурентен достъп до данните от други транзакции, потребители или системи.
* Те осигуряват механизми за запазване целостта на данните, при отпадане на система или некоректно поведение на системата.

Това което отличава това дългите транзакции от транзакции в стандартните транзакции е факта, че точно и ясно не е определено кога тя ще приключи. Също така не е известно дали тя ще приключи изобщо. Като особеност на дългата транзакция може да изтъкне факта, че тя притежава свойството да запазва промените (persistence). В рамките на версионизирането, дългата транзакция представлява поредица от действия, чийто краен резултат представлява нова версия на продукта. Дългите транзакции поддържат механизъм за едновременна работа на няколко потребителя, при който се използва механизма на вложени транзакции [G275]. Пространството в което се изпълнява дадена дълга транзакция е известно и като работно пространство. Дългите транзакции и механизма на влагане на транзакции, може да се разглежда като предшественик на работните пространства, които са по-подробно разгледани в глава 4.

### Версионизиране чрез набор от промени

В този модел продуктите се описават като базова линия и набори от промени, определящи версиите на компонентите. Този модел на версионизиране силно използва подхода на съхраняване на промените в репозитори, разгледан в 1.2.1. Особеност на набора от промени е, че този подход се характеризира с намалени изисквание към използваната памет.

Друга характеристика на този подход е необходимостта да се следва определена последователност при прилагане на промените върху обектите.



Фиг. 6 Набор от промени при построяване на конфигурация [G275]

Ориентиран е на логическата промяна на компонентите[G275]

Примери – поправки (patches). Ограничения и зависимости между отделните поправки (patches).

...

### Обектно-ориентиран модел за контрол на версии

В своето изследване [G141], Нгуйен представя модел на система за управление на версии, която се базира на обектно-ориентиран модел на артефактите. Основните елементи на модела са проектът, който от своя страна е съставен от компоненти. Компонентите биват два вида – композирани и атомни компоненти. Композираните компоненти могат да бъдат съставени от други композирани компоненти и/или от атомни компоненти. Атомните компоненти съдържат структурни единици, която представлява най-малката единица подложена на контрол на версия. На Фиг. 7 графично е представен обектно-ориентиран модел, използван в системата Molhado.

Като ключова особеност на своят модел, авторите изтъкват неговата логическа и обектно-ориентираност. Това позволява да се разшири версионизирането на обектите на логическо ниво, което не налага някаква конкретна форма на физическо представяне на данните. Така разработчиците могат да се фокусират над моделирането на софтуерният продукт в съответствие с конкретна парадигма, а не над конкретното файлово представяне на данните.

ГКато недостатък на представеният модел може да се отбележи наличието на два вида компоненти – композиран и атомен. Това прави по-сложен и по-труден за реализиране. В частност това изисква реализиране на функции, които да транформират атомните компоненти в композиране и други такива, които да трансформират композиран компонент в атомен.



Фиг. 7 Обектно-ориентиран системен модел в системата Molhado [G141]

## Модели на версионизиран обект

Секцията има за цел да представи различните модели на версионизирани обект. За всеки модел са представени неговите силни и слаби страни, които служат да се направят изводи определящи научно-приложните приноси в областта.

Преди да се направи обзор на моделита на версионизиран обект, следва да определим самото понятие. Водещите автори в областта на управлението и контрола на версиите [G42, G8] декомпозират версионизираните обекти на две съставни части – състояния на обекта (версии) и граф на версиите. Графът на версиите представлява граф, при който върховете представляват отделните версии на обекта, а ребрата съответстват на логическата последователност на създаване на версиите.

### Файлово базиран модел

При файловият модел версионизираните обекти се съхраняват под формата на файлове в рамките на файлова система. Този модел е широко използван от популярните системи за управление и контрол на версиите, такива като RCS [G365, G366], CVS [G50, G54], Subversion [G101], ClearCase [G319]. Предпоставка за това е лесното реализиране, приемливата скорост на работа и факта, че потребителите на тези системи имат добри навици при работа с файлове. Въпреки това много изследователи […,G141, G360] определят този модел като несъвършен, като се изтъкват следните недостатъци:

* Файлът като единица за съхраняване на данните е твърде обемиста и не гарантира, че данните са структуризирани.
* Съществува реална опасност от получаване на несъответствие между модели, логическа реализация и версионизиране на файлове [G141], тъй като файлово ориентираният модел не предоставя механизъм за дефиниране и управление на свързаност между различните артефакти – изисквания, модели, код, тестови сценарии. Това води до трудности при управлението на средни и големи по обхват и сложност проекти […]. От друга страна, използването на допълнителни файлове, за описване на свързаността между файловете, води до увеличаване ръчната работа и съответно на риска от получаване на грешки в крайният продукт.

Системите базирани на файловият модел до голяма степен разчитат на възможностите, предоставени от файловата система при осигуряване на различните нива на достъп до обектите [да видя дали го нямам по-горе]. Следва да се отбележи че възможностите, които предоставя файловите системи не винаги могат да отговарят на конкретните нужди на проекта.

### Обектно-ориентиран модел

В системата Molhado [G141] е използван обектно-ориентиран подход за версионизиране на обектите (артефакти). При него обектите се асоциират със възли (от англ. node), а те от своя страна се разбиват на атрибути. Получената матрица авторите я наричат матрица от слотове. Тя се допълва от трето измерение – това на версиите.



Фиг. Модел на данните за версионизиране на артефактите в системата Molhado [G141]

Като слабости на разгледаният модел може да се изтъкне неговата фиксираност на модела на съхраняваните обекти. Това се допълва от факта, че при промяна на самият модел на версионизираните обекти, е нужно да се използват допълнителни инструменти за версионизиране на модела. При промяна на модела на версионизираните обекти възниква необходимостта от използванети на допълнителни инструменти, които да осигурят трансформацията на обектите от единият модел към другият и обратно. [да потърся примери за такива случаи]

### Съхраняване на промени

В практиката са известни следните няколко подхода за съхраняване на промените […]:

* Съхраняване на състояние
* Съхраняване на разлики (промени)
* Съхраняване на транзакции

При подхода за съхраняване на „състояние” (англ. snapshot) в хранилището всяка версия на обектите се съхранява под формата на самостоятелно копие. При тази стратегия се постига устойчивост и независимост на отделната версия на даден обект спрямо другите негови други версии. Това се изразява във факта, че евентуалната недостъпност на отделна версия не оказва влияние над достъпа до другите версии на артефакта. Друга характеристика при този подход е неговата висока скорост на обработка на заявките, което е неразривно свързано с повишени изисквания към обема на паметта на хранилището, които зависят от големината на направените промени, а от броя на промените. Тук следва да отбележим, че големина е пряко свързана със степента на гранулираност на артефакта. При извличане на съществуваща или при съхраняване на нова версия този подход ни дава константна сложност на алгоритмите при този подход.



Фиг. Пример за извличане на версия при подход за съхраняване на „състояния”

При разглеждането съхраняването на промени различните автори […] използват понятието делта между две версии, която отразява разликите между две версии на даден обект. В [G319, стр. 60] са представини три вида реализации на подхода на съхравяване на промени: прав подход; обратен подход; вграден подход.

##### Прав подход за съхраняване на „промени”

При този подход се базира на съхраняване именно на тези делти в репозиторито. При заявка за извличане на определена версия на даден обект, системата следва да извлече началното състояние на обекта и всички делти, направени преди исканата версия. На базата на тези данни системата следва да изчисли заявената версия. В следствие на това могат да се обобщят следните недостатъци на подхода:

* Подхода е неустойчив поради факта, че версията се изчислява на база предишна версия и приложен върху нея делти
* Бързината на системата е намалена, което се обуславя от факта, че при извличането на определена версия на определен версионизиран обект системата трябва да извлече начална версия и набор от делти, от които да изчисли исканата версия. От Фиг. 2 фигурата ориентировъчно може да се определи сложността на алгоритъма по извличане на версия на обект, при използване на подход за съхраняване на „промени”, като линейна спрямо номера на версията, която е нужно да се извлече:

f(x) = O(n) {за обръщения към репозиторито} + O(n-1) {за изчисляване на версия} = O(n), където n е номера на версията на обекта

* При записване на нова версия е нужно да се изчисли предходната версия, след това се изчислява делтата между двете версии, която накрая се съхранява в репозиторито.
* Подходът е трудно приложим за версионизиране на нетипизирани или слабо-типизирани обекти (данни), понеже сложността на изчисляването на делтите между версиите би довело до претоварване на системата, дори и при малки разлики между версиите [G291].



Фиг. Пример за извличане на версия при прав подход за съхраняване на „промени”

Описаният по-горе негатив при силно типизираните версионизирани обекти се превръща в предимство – при стратегията “набор от промени”, изискванията към обема на външната памет стават значително по-малки. Така например в [G348] авторите в резултат на експеримент показват, че при използване на подхода за съхраняване на „състояние” изисква до 30 пъти повече дисково пространство в сравнение с използване на подход за съхраняване на „промяна”.

##### Обратен подход за съхраняване на „промени”

Поради слабостите на правият подход е създаден обратен подход да съхраняване на „промени”. При него акцента е в посока да се съхранява в монолитна форма само последната версия на обекта и на делти за разлики с предишни версии. При този подход дава подобрение в производителността, когато се работи предимно с последните версии на продукта.

Слабостта на тази реализация е при работа с по-стари или разклонени версии. Така в репозиторито започват да се съхраняват няколко последни версии – по една за всяко разклонение на версиите.



Фиг. Пример за извличане на версия при обратен подход за съхраняване на „промени”

Обратният подход за съхраняване непромени се използва за първи път в системата за управление на изходен код RCS [G365, G366]. На по-късен етап с цел подобряване производителността на системата, употребата на подходът сведена само до главната линия на разработка [G319 – page 60]

##### Вграден (inline) подход за съхраняване на „промени”

При вграденият подход за съхраняване на „промени” не се съхраняват никакви състояния на обектите, а само делти във формата на специална нотация, обезпечавайки възстановяване на произволна версия за постоянно време. Подходът е заложен в основата на системите CSSC и ClearCase. [G319 – page 60]. За повишаване скоростта на работа на системата, вграденият подход бива допълнен от механизми за кеширане на сглобените версии, които са най-често използвани.

### Интеграция между разклонени версии

В жизненият цикъл на софтуерните продукти е често срещано явление продукта да се адаптира според конкретните нужди на даден клиент. Това е предпоставка за разклоняване на версия от базовата линия с версии на продукта. След натрупване на определено количество близки адаптации, се появява нужда от улесняване тяхната поддръжка. В следствие на процес на тяхното рефакториране те се генарилизират в общ компонент за системата. Именно при процесът на обединяване на версиите от две паралелни линии е поставена задачата за интеграция на версии, които заемат основно място в системите за контрол и управление на версиите. При сливането на две версии на даден артефакт е поставена задачата да се запазят автоматично, до колкото е възможно, неговите характеристики реализирани в двете версии. При невъзможност това да се извърши по автоматичен начин (т. нар. конфликт между версиите), процесът на сливане се извършва ръчно. В практиката са разработени различни методи за автоматично сливане на версии, като може да се отбележат следните направления в областта [G32]:

* Сливане чрез на символен анализ на текст,
* Семантичнен анализ на текст, чрез конструиране на граф на зависимостите за всеки един обект. [G357, G359]
* Съхраняване на история на командите, променящи състоянието на обекта.[G357]

[G360] two-way merge, three-way merge

Текстовото сливане [G273, G148], предполага търсене и намиране на максимални по дължина общи под-низове в съдържанието на двете версии, както и определяне на техните разлики – делти. Според вида на делтите, системата може да предприеме автоматични действия за интеграция на версиите или да предостави процеса по сливане на версии в ръцете на потребителя.



Фиг. Сливане на делти на две версии [G109]

В своето изследване Вестфехтел [G109] представя алгоритъм за сливане на артефакти, базиран на синтактичен анализ на структурирани документи. При сливането, той определя и следва следните правила за сливане на два артефакта:

* Всяка промяна трябва да идва или от първият или от вторият входен артефакт;
* Изходната версия d, не трябва да съдържа дублираща се информация;
* Изходната версия d, не трябва да съдържа конфликтни точки, т.е. елементи, който са премахнат в един от двата входни артефакта.

Интеграция чрез семантичен анализ на текст [G274] [G32] – граф на зависимостите.

...

Сливане, базирано на съхранена история на командите по време на редактирането

[G360 и G361, G177] – да ги прегледам

…

### Съставност и гранулираност на версионизирани обекти

В тази секция се разглежда гранулираността и съставността на версионизираните обекти, както и практическите нужди, довели до тяхната поява. Под съставен версионизиран обект ще се разбира такъв обект, който е съставен от други версионизирани обекти.

<http://en.wikipedia.org/wiki/Object_composition>

#### Съставни обекти

Информационните системи и програмните езици исторически се развиват в посока за оптимизиране на работният процес. Като представител на основните похвати за постигане на това може да се изтъкне увеличаването степента на преизползване и унифицираност на програмния код и изграждащите софтуера обекти. Така в практиката като стандартен метод на изграждане на системите е един основен обект да включва в себе си други обекти или да използва други обекти през референция. Като добри примери в това направление могат да се изтъкнат стандартните библиотеки, които са вече неразделна част от всеки един програмен език.

Допълнително в подходите за версионизиране от тип извличане/записване към управление на конфигурация [G275] се появява нуждата от определянето на композицията на обекти също като версионизиран обект [G334]. В [G275] конфигурацията е определена като същност състояща се от две части – системен модел и правила за избор на версията. Показаният по-горе на модел за избор на версия на компонент при конфигурация [G275] ...

В най-новите изследвания […Наков за Java7] се идентифизира нуждата от зависимост на продукта от минимална версия на библиотека или компонент…

## Съвместна работа и работни пространства

В секцията са разгледани в исторически план развитието на моделите на работните пространств. Представени са техните предимства и недостатъци, което служи като основа за направата на изводи и детайлното определяне на задачата по създаване на модел на йерархично композируеми работни пространства .  В [G171] авторите предлагат концепцията, че управлявлението на работните пространства следва да се разглежда като по-високо ниво на абстракция отколкото операциите на извличането и записването (checkout/checkin) на примитиви (версии) и работна директория.

### Модели за осигуряване на конкурентен/паралелен достъп

Настоящата секция е фокусирана да представи различните общоприети практики за осигуряване на паралелен и конкурентен достъп до версионизирани ресурси. Едновременно с това е направен опит да се съпоставят представят силните и слабите страни при различните модели.

#### Модел за конкурентен достъп „Заключване – Модифициране – Отключване”

При файлово базираните хранилища, както беше споменато по-горе, нивото на гранулация на версионизираните обекти е сравнително голямо. При този тип репозиторита часто се използва решението „Заключване – Модифициране – Отключване”. [G101]



Фиг. Подход "заключи, модифицирай, отключи"[G101]

При този подход са известни следните недостатъци:

* Заключването може да предизвика административни проблеми – т. нар. мъртва хватка, породена от едновременен опит от двама потребители за екслузивен достъп до едни и същи обекти;
* Заключването може да се окаже предпоставка за ненужна сериализация – проблем от организационно естество. Често един потребител има необходимост да актуализира началото на един файл, докато друг – краят на същият файл.
* Заключването може да създаде фалшиво чувство за сигурност. [G101]

Често срещано положение при поддръжката на софтуерните продукти е, че за отстраняването на даден проблем или програмна грешка е нужно да се извършат малки по размер промени на много места. Това довежда до прекомерно увеличаване броят на заключените файлове, което от своя страна е предпоставка за понижаване ефективността на работа. Според [G101] съществуват различни практики, за решаването на тези недостатъци, но те могат да се разглеждат по скоро като заобикаляне същността на проблема, отколкото като цялостен подход за неговото разрешаване

#### Оптимистично и песимистично заключване при достъп до споделен ресурс

За осъществяването на конкурентен достъп до файловете се използва както оптимистичен, така и песимистичен подход за заключване на файловете.

При оптимистичният заключване няколко потребители могат да отворят за редактиране даден файл. При записването на файла системата известява останалите потребители, че версията в хранилището е била актуализирана. При песимистичното заключване системата предоставя възможност само един потребител да работи с файла с право на запис. Останалите потребители имат достъп до ресурса само в режим на четене. Едва след отключването на файла, останалите потребители имат възможност да извършват манипулации над него.

#### Модел за конкурентен достъп „Копиране – Модифициране – Сливане”

Еволюционно като наследник на горе описаният подход за версионизиране представлява може да се посочи подхода „Копиране – Модифициране – Сливане”. При него ективно се използват алгоритми за сливане на версионизирани обекти, които възникват при паралелната им промяна. Проблемът за сливане на две версии е представен по-подробно в точка 1.3.4.



Фиг. Подход “копирай - модифицирай – слей” [G101]

На Фиг. 14 графично е представен подхода „Копирай-модифицирай-слей” за версионизиран обект. Както се вижда от фигурата, слабо място на модела е опитът за запис и получаването на грешка от тип “out-of-date”. При така създалата се ситуация, потребителят на системата е отговорен за сливането на своята версия с тази, която се намира в репозиторито.

### Същност на работното пространство

В средите за разработка на софтуер под термина работно пространство се разбира изолирано място за разработка (работа), което отговаря на определени функционални критерии. В [G12, стр.406] авторите казват, че „... модерното работно пространство е създадено "зад гърба на науката”, за да изпълни задача, определена от потребителя...”. Работното пространство се определя като системен елемент, който предоставя следните три основни функции:

1. Пясъчна кутия – безопасно място, където потребителите имат свободата да променят и създават обекти.
2. Позволява да се построи на определена версия/ конфигурация на софтуерната система.
3. Изолираност на промените, тестовете, и другите съпътстващи дейности без това да влияе на останалите участници в създаването на софтуерния продукт.

Общоприето разбиране за работното пространство [G12, G50, G276] е, че то представлява файлова директория. От друга страна в [G3] то се определя като инструмент предоставящ механизми за коопериране и интегриране на отделните части на проекта. Предизвикателствата към тази интерпретация на работните пространства са свързани с осигуряването на политика на достъп до версионизираните обекти. Като най-популярен подход можем да изтъкне метода на извличане/запис [G51], който често е допълнен от механизми за заключване на артефактите.

Като следствие от горното може да се определят следните основни направления на развитие при работните пространства:

1. Постигане на изолираност, с цел да се предотврати неволното или нежеланото взаимодействие при паралелна работа над компонентите на софтуерният продукт.
2. Място за коопериране, където екипите постигат синхрон и интегрираност на крайният продукт. [G313]

[G104] view = workspace (ClearCase), да видя и [G319] какво пише по въпроса.

### Файлово базирани работни пространства.

При файлово-ориентираните системи за управление на версията [G50, G101] под работно пространство се разбира файловата директория в която се извличат от репозиторито версиите на файловете. На Фиг. 15 е представен модел на работно пространство при файлово-ориентираните системи за управление на версията.



Фиг. Модел на работно пространство при файлово-ориентирани системи за управление на версия

Като предимства на представеният модел може да се посочат неговата простота за реализация, бързодействие и лекота за работа. Въпреки това съществуват редица недостатъци, основните сред които са:

* Правото на достъп до обектите се гарантира от файловата система. Много малко са файловите системи, които са способни да предостави нужното ниво на сигурност, които се изисква при работа с версионизирани обекти.
* Файловете, като версионизиран обект са с твърде едра гранулираност и както бе споменато по-горе, те не предоставят възможност за дефиниране на релации и проследимост между отделните обекти (файлове).

При сравняване с предишна версия на даден файл, потребителят е нужно да се обърне към хранилището за извличане на предишната версия на файла. При условие, че изходната за файла версия вече е била актуализирана в репозиторито, се получава усложняване задачата по намиране и извличане на истинската предишна версия. Именно това затрудняване, следствие на паралелна и конкурентна работа води до най-големи разходи на ресурси (човеко-часове), и като следствие до не достатъчно ефективна работа на потребителите на системите за контрол на версия.

### Модел на работни пространства със съхранена версия

В своето изследване Фейлер [G275] представя модел на работно пространство, който е съставен от два вида елемента: съхранена версия и работна версия - Фиг. 16. При извличане на версия от репозиторито, тя се представя в работното пространство като съхранена версия. Съхранената версия не може да бъдат променяна в рамките на работното пространство, нейната роля е да осигурят рамка над която се извършват нужните промени.Всички промени в работното пространство се отразяват само в работната версия на продукта. При записване на промените в репозиторито се записва именно работната версия. Наличието на съхранена и работна версия в работното пространство позволява да се постигне изолираност на процеса на работа в случай на промяна на изходната версия в репозиторито. Освен това се постига автономност на работата – за стравнение спрямо изходната версия не е нужно обръщение към репозиторито.



Фиг. Модел на работно пространство със съхранена версия [G275]

Въпреки че моделът на работно пространство със съхранена версия до известна степен решава въпроса с определянето на разликите между две версии и автономността на процеса на разработка, това създава друго проблемно положение. Пример за паралелно записване и несъответствие на съхранената версия и тази в репозиторито. Също така при паралелна работа над една и съща изходна версия има опасност от изгубване на промени при записване в репозиторито. На Фиг. 17 е представена примерна ситуация за описаната слабост на модела. Тук програмистът Иван работещ върху отстраняване на дефект в компонент А, създава негова нова версия и записва конфигурацията в репозиторито. Програмистът Петър, работещ над задача за подобряване на компонент В, извършва запис в репозиторито на своята конфигурация, в която е включена дефектна версия на компонент А.



Фиг. Примерна ситуация за конфликт между версии на един компонент

### Модел на разпределени хранилища

Разпределените хранилища представляват модерен подход за осигуряване на паралелна работа над огромни продукти. Като най-известният пример за продукт използващ разпределени хранилища може да се спомене ядрото на Линукс […]. Най-популярните системи за разпределено управление на версията са GIT […] и Mercury […]. Подходите използвани при създаването на тези системи са предприети с цел намаляване (даже ликвидиране) необходимостта от поддържане на централно хранилище. На машината на всеки потребител се инсталира отделна система за контрол на версията, със собствено хранилище. Репозиторито на тези системи се синхронизира с главно репозитори, което също така се явява инсталация на тази система. Този подход до голяма степен ускорява аналитическата работа с репозиторито, което се намира локално. Въпреки това тези системи наследяват недостатъците от използване файловите версионизирани обекти.

... Диаграма...

### Модели на йерархично композиране на работни пространства

В опитите си да намерят баланс между кооперативността и изолираността, различни изследователи до концепцията да се подредят работните пространства във формата йерархично-дървовидна структура, т.е. да се композират йерархично. Така например в [G170] е търсен начин да разрешат недостатъците при конкурентен достъп до мета-данните на общо работното пространство. В изследването изоставят механизма на заключване и се представят опростена политика на достъп, с прилагането на двуслойна йерархия от работни пространства. Тя се състои от групово работно пространство (корен) и частни работни пространства (листа). Авторите предполагат, че по този начин ще се постигне намаляване колизиите при опити за достъп до едни и същи ресурси по време на извличане и записване в груповото работно пространство [G279].

В [G284] авторите предлагат използването на многослойна йерархия от работни пространства. Като механизъм за достъп до обектите те избрат файлово-ориентиран подход на извличане и записване от родителско работно пространство. Във [G12, G285] представят виртуални работните пространства, наричани още изгледи (англ. view), като способ за осигуряване на достъп с право на запис до избрани версионизираните обекти и достъп с право на четене до другите версионизирани обекти. Като причина за появата на виртуалните пространства се изтъква увеличението бързодействието на системата, чрез намаляване на използваните системни ресурси. [G313].

Като заключение може да се посочат следните предимства при използването на йерархично композиране на работните пространства:

1. Драстично се намалява броя на колизиите при конкурентен достъп до версионизираните обекти.
2. Постигане на естествена изолираност от промени извън работното пространство, в частност от промени, извършвани от съседни работни пространства.
3. Подобряване общото бързодействие на разработване в следствие от реализирането на по-ефективен работен процес

## Методи за проследимост на промените

### Проследимостта на промените

Лошата практика да се предоставят непроменяем документ с изисквания много често предизвиква огромни непредвидени разходи за даден проект. Един такъв документ не може да обхване всички аспекти, които трябва да се поддържат от крайният продукт. Промените в изискванията и обхвата на проекта много често се променят без да са били отразени в документа с изисквания. Затова този документ би следвало да е жив документ, който да се развива паралелно, като неизменна част от софтуерният продукт. Авторите на [G306] препоръчват да се използва техниката за проследимост, като една добра практика при управление на промените.

Проследимостта на версията ни позволява да направим оценка на реалната цена на продукта. Предложеният модел на проследимост на промените претендира за принос в сферата на оценяването на сложността на задачите. Така например в [G72] авторите правят опит за такова оценяване, базирано на проследяване на подобни промени в системата, които са били извършвани по-рано.

За ефективното управление на софтуерните проекти е необходимо ефективно да се управляват промените, както и да се използват ефективни подходи за оценяване на необходимите ресурси и време за реализация при промяна в изискванията или при определянето на дефекти в софтуерния продукт. Основен инструмент за осигуряването на това е проследимостта на промените, който освен горните две предпоставки има голямо влияние и в области като [G312, G311, G339]:

* Сертифициране – Проследимостта на промените подпомага процесът на сертифициране на програмните продукти, в частта пълнота на реализирането (имплементирането) на изискванията.
* Анализ на ефекта от промяната - Без използването на проследимостта практиката показва, че е твърде вероятно да се пропусне елемент от системата, който е повлиян от промяна на дадено изискване.
* Поддръжката на програмният продукт се улеснява, особено при използването на процес на системно подобряване и отстраняване на грешки.
* Улесняване проследяване състоянието на проекта е следствие от използването на политика за прилагане на практики за проследяване на промени.
* Ускоряване процесът на преработване и оптимизиране на определени част от системата.
* Преизползването на елементи изграждащи системата в други системи се обуславя от наличието на връзки между дизайн, код и тестове.
* Намаляване влиянието на рисковете за проекта, в следствие от подобрените възможности за следене състоянието на проекта.
* При проверка качеството на софтуерните продукти, връзките за проследяване предоставят информация, коя част от системата, кое изискване реализира. По този начин се подобрява определянето дали това изискване е удовлетворено.

Когато се говори за проследимост на промените спрямо дадено изискване, трябва да се има в предвид, наличието на четири типа проследяващи връзки [G339]:

* Напред от изискванията – връзките от този тип определят работата или действията, които следва да бъдат направени.



Фиг. Типове прослевяващи връзки [G339]

* Назад към изискванията – тези връзки представят проверките, удостоверяващи че имплементацията на продукта съответства на изискванията. Анализът на тези връзки предоставя механизъм за предотвратяване имплементирането на функционалност на златната чиния (gold-plating).
* Напред към изискванията отразява връзките продиктувани от промяна в бизнес нуждите, техническите презумпции, които следва да са отразени в пакета от изисквания към продукта.
* Назад от изискванията [G310, G339] осигурява възможност да се определи дали клиетските нужди са технически реализуеми, както и да се определи до каква степен техническото решение налага ограничаване или модификация на нуждите на клиента.

### Работни единици

Хелминг в своят труд [G311] въвежда понятието „работна единица” (англ. WorkItems) като дефиниция за работата, която следва да се извърши. Пак там авторите представят следната класификация на работните единици:

* Единици за действие (англ. ActionItems). Те представляват определена съвкупност от дейности, която произтича както от изискванията към крайният продукт, така и от методологията, която се следва в процеса на създаване на продукта.
* Въпросите (англ. Issues) представляват работни единици, резултатът от които е да изяснят, допълнят или коригират работни единици от другите видове.
* Репорт за дефект *(англ. BugReports)*. Репортът за дефект представлява искане за промяна във функционалността на системата, с цел достигане на пълно съотвествие на определени общоприети стандарти и изисквания към продукта.
* Работен пакет (англ. WorkPackage). Работният пакет може да се разглежда като пакет, в който се съдържа съвкупност от работни единици от горните три вида. Освен това, той може да съдържа и други работни пакети в себе си. Чрез него се осигурява различните нива на абстракция на заданията към различните участници в процеса.

### Управление на изискванията и исканията за промени

Управлението на изискванията заема важно място в съвременните процеси за създаването на софтуер. Всеки софтуерен продукт се създава с цел да удовлетвори някакъв набор от изисквания. При големи проекти, където участниците в проекта (stakeholder) са достатъчно много и притежават специализация (компетенции) в различни сфери като: оперативна поддръжка на системите, маркетинг, финанси, контрол, отношения с клиенти, планиране, логистика и др. Това довежда до генерирането на голямо количество от изисквания, които по може да се променят, отменят, както и да се заменят с нови такива.

Системите за управление на изискванията предоставят възможности за проследяване на техният жизнен цикъл, чрез използване на методологии от версионизирането. Самите изисквания и искания за промени са добър пример за работни единици.

### Методи за проследяване на промени

Едно често срещано явление при създаването на софтуер е лошата практика документа с бизнес спецификация или този със системна спецификация да се предоставя в завършен и непроменяем вид. Това неминуемо представлява голям риск за успеха на пректа, тъй като може да причини огромни разходи. Един такъв документ много трудно може да покрие всички функционални аспекти, на които трябва да отговаря крайният продукт. Изискванията и обхвата на проекта често биват променяни, без това да се отрази в документа със спецификация, което на един по-късен етап води до трудности с определянето дали софтуерният продукт отговаря на нуждите на потребителя. Затова документа с изискванията би следвало да е един жив документ, който да се развива паралелно със софтуерният продукт. За улесняване на дейностите по промяна на изискванията и техният обхват, авторите на [G306] препоръчват използването на техниката проследимост на изискванията, като добра практика в процеса на управление на измененията.

Доброто управление на измененията предполага използването на добра проследимост на измененията между всички артефакти, които са част от процеса на създаване на софтуерният продукт – изисквания, системен дизайн, тестове за установяване на качеството на продукта, изходният код и документацията. Активното използването на проследимостта между отделните артефакти ни дава следните предимства [G312, G311, G339]:

* Улеснен анализ и оценка на влиянието на дадена промяна, както над целият продукт, така и над всеки отделен негов компонент;
* Намаляване на себестойността на поддръжката на продукта;
* Подобрена оценка за степента на качество на продукта;
* Редуциране и оптимизиране работата по преработване на системата и други.

Клеланд-Хуанг [G308] представя класификация за методите за проследяване на промени за нефукционални изисквания. Тя отбелязва, че тези методи намират приложимост също така и при проследяването на функционалните изисквания. По-долу са представени отделните методи, техните особености, предимства и недостатъци.

#### Матрици

Матриците са най-масово използваният метод за статично представяне на отношения между изискванията и другите компоненти при създаването на софтуер. Матриците се отличават с високо ниво на ръчни процеси при тяхното създаване, поддържане и анализ степента на влияние на дадена промяна в даден компонент над останалите компоненти от системата. Те също така не предоставят добро ниво на разширяемост и в течение на времето е възможно данните в матриците да не съответстват на действителността. Тяхното широко разпространения се обяснява от най-голямото им предимство - тяхната простота.

#### Ключови думи и онтология

Чрез използването на ключови думи и онтология се предоставя възможност да се проследяват връзките между изискванията и системната архитектура. Като предимство на този метод може да се изтъкне липсата на необходимост от поддържане на централизирана матрица, както и разпределянето на проследяващите връзки в самите артефакти на системата. Главното предизвикателство при този метод е подготвянето и резервирането на набор от ключови думи, които ще се използват през целият жизнен цикъл на продукта. Решаването на тази задача се явява и основна слабост на метода в посока разширяемост и дълговременна поддръжка на връзките.

#### Аспектно изграждане (Aspect Weaving)

Метода на аспектно изграждане е приложим при използване на аспектно ориентираното подход на изграждане (Aspect Oriented Programming - AOP) на системите, който се базира на пресичане на практиката подобни по своята същност функционалности да се намират в различни части на системата. В AOП подобните задачи формират аспекти, които се капсулират в една същност, която чрез специални техники и правила бива вградена в кода на програмата. По този начин AOП осигурява връзката между аспектите и имплементацията. За съжаление Аспектното изграждане не предоставя добри възможности за оценяване обхвата и влиянието на промяна над съществуващ продукт.

#### Извличане на информация

Този метод се е появил в следстви на опитите на изследователите да автоматизират дейносттите по изграждане и поддържане на матриците за проследимост. Техните изследвания [G322, G323] са насочени именно в посока извличане на нужната информация от ... Основен акцент на методът е обработката подобности, базирана на честотата на срещане и описанието на термините в заявката и обработваните документи. Методът се отличава с автоматично генериране на проследяващите връзки между обектите. Въпреки това той не гарантира 100% прецизност на полученият резултат.

Първоначално определянето на обхвата и влиянието на допълнителните промени не са били предмет на метода. Това предполага използването на ръчни методи за тяхното определяне. Но на по-късен етап изследователи [G72] правят разработки в посока автоматизиране на анализа при използване на метода.

#### Проследимост базирана на сценарии

Целта на този метод е да се изгради карта от свързвания (мапинг) между изискванията и останалите артефакти на системата – архитектура, код, тестови сценарии. Като недостатък на подхода следва да се отбележи необходимостта от ръчно създаване и поддържане на връзките на проследимост, както и че сценариите не могат да обхванат в пълнота функционалността на системата.

#### Проследимост базирана на събития (Event-based Traceability)

Проследимостта базирана на събития представлява метод, при който се използва архитектурата публикуване-записване при дефиниране на връзките за проследимост. Методът се отличава с факта, че предоставя възможност за автоматизирано оценяване на влиянието и обхвата на промените, както и автоматизираност при мониторинга на прогреса на проекта и при оценяването на архитектурните модели.

#### Процесно-ориентиран метод

Методът получава практическо приложение при процесно-ориентираните инжинерни среди […]. Тези среди включват следните три области – моделиране, прокарване и производителност. Моделирането определя процесът и задачите по проследимост. Прокарването реализира и контролира процесът по реализиране на продукта, както съответните проследяващи връзки. Производителността на методът се отличава с трудоемкост при създаването и поддържането на проследяващите връзки. Той из...

## Изводи

От направеният анализ на предметната област предизвикателствата по-долу следва да се разглеждат като цели и задачи на настоящият научно-приложен труд. От направеният обзор може да се обобщят следните направления за подобряване в областта на управлението и контрола на версии:

Осигуряване на гъвкавост при дефиниране на степеннта на гранулираност на обектите и тяхната конфигурираност.

Осигуряване на гъвкав модел…

Осигуряване на простота, универсалност и разширяемост на модела.

Недостатъчно детайлните разработки в сферата на йерархично композираните работни пространства са предпоставка за търсене на научно-практическа новост именно в това направление.

Съчетавайки в единен механизъм двете основни направления за употреба на работните пространства, изолираност и интеграция, са също обект на научно изследване.

Като подход за конкурентен достъп ще се спрем на модел на ексклузивен достъп до определено работно пространство само от един потребител едновременно.

При изграждането на прототипа, е удачно да се използва подход за съхраняване състояния на версионизираните обекти, тъй като подхода е прост за реализация и предполага по-висока скорост системата.

Процес включващ всички звена при създаването на софтуер – анализ, архитектура, разработка, качествен контрол, документация....

# Глава втора Модели за управление на версии в среда с йерархична композиция на работни пространства

В настоящата глава са представени теоретичните разработки и модели, чиято разработка е пряко свързана с успешното решаване на задачите, определени в първа глава на настоящият научно-приложен труд. В секциите съставящи главата представени следните модели: модел на версионизиран обект, модел на йерархично композирани работни пространства, модел на видимост на версионизирани обекти в среда на йерархично композирани работни пространства, класификация на версионизиращи транзакции, модел на жизнен цикъл на версионизиран обект и адаптация на метод за полуавтоматична проследимост на промените в среда на йерархично композирани работни пространства.

## Модел на версионизиран обект

В настоящата секцията има за цел да представи модел на версионизиран обект. Моделът който се предоставя възможност за гъвкаво комбиниране между версионизирани обекти, като се изграждат композиции от обекти. При построяването и представянето на модел на версионизиран обект е използван това на същност-отношение (Entity-Relationship). Той ни предоставя добър механизъм за гъвкаво и свободно реализиране на поставените задачи.

Както беше споменато в обзорната глава, водещите автори в областта на управлението и контрола на версиите [G42, G8] определят версионизираните обекти, като съставен от две части – състояния на обекта (версии) и граф на версиите. Под граф на версиите се разбира такъв граф, чиито върховете представляват отделните състояния (версии) на обекта, а ребрата съответстват на логическата последователност на създаване на версиите (Фиг. 19). В настоящото научно-приложно изследване термините „версионизиран примитив”, „версия на обект” и „състояние на версионизиран обект” ще бъде използван като синоними.

Този модел не отговаря на изискванията на поставените обзорната глава задачи и така изграденият моделът следва да се подобри и развие.



Фиг. Концептуален модел на версионизиран обект

Първата нова характеристика, която следва да притежава един модел на версионизиран обект е той да предоставя възможност да се определи нивото на детайлизираност, т.е. на гранулираност, която самият модел следва да поддържа. Така се определя необходимостта за включването на механизъм за построяване на композиции от обекти, като основна част на модела. Като формална дефиниция на термина „съставен обект” може да се приеме следната:

**Дефиниция**: Съставен обект ще наричаме обект, който е съставен от други обекти (версии на обекти) посредством композиции.

**Дефиниция**: Композиция представлява същност определяща връзката между супер-обект и под-обект. Един съставен обект може да бъде супер-обект на една или повече композиции, т.е. да е съставен от един или повече под-обекти.

Включването на композиранете обекти и съответно на под-обектите в предметната област води до необходимостта да се изследва (преоцени) как това се отразява на процесът на версионизиране на обектите. Така на Фиг. 20 е представен пример за промени в композицията на обектите, като със стрелки са показани промените в съставността (състава) на обектите между отделните версии на крайният продукт. Детайлно изследване на версионизирането на съставни обекти е представено в следващата подсекция на главата.



Фиг. Пример за промяни на съставността на обектите

При свързването на версиите на един версионизиран обект е необходимо да се използва релационна връзка от тип външен ключ (англ. foreign key). От тук следва, че самата същност **версионизиран обект** е необходимо да притежава само и единствено уникален и непроменяем номер, който е удачно да се използва и като първичен ключ за същността.

Като основни елементи на същността **версионизиран примитив** може да се определят следните:

* Номер на версионизиран обект, с който дадената версия е свързан;
* Номер на версия – пореден номер, който определя по уникален начин версията в рамките на обекта.
* Наименование на обекта. Определяйки наименованието на ниво примитив, потребителят получава възможност да проследява отделните версии даже и при преименоване на обектите. Така полученият модел става по-пълноценен, елиминерайки недостатъка свързан с преименоването на обектите (файловете) при системи като CVS и SVN […].
* Съдържание на обекта, включващи данните в съответната версията на обекта.

Версионизираният примитив се определя еднозначно посредством уникалната двойка **номер на версионизиран обект** и **номер на версия**. Въпреки възможността да се използва уникалната двойка като съставен първичен ключ, добрата практика в проектирането на ER модели […] препоръчват всяка отделна същност да притежава свой собствен, не съставен ключ. Т.е. в разглежданата същност за първичен ключ ще се използва допълнително поле – глобален номер на версията.

За нуждите на версионизиране на съставни обектите, следва да се дефинира допълнителна същност - „Композиция на версионизирани примитиви” (накратко композиция), която е еднозначно свързва версията на супер обекта с версиите на неговите под-обекти. Като атрибути на композицията може да се определят иследните:

* Глобален номер на супер-обекта;
* Глобален номер на под-обекта.

Въпреки че комбинацията от двата атрибута е уникална и еднозначно определят една нейна инстанция, за разглежданата същност е необходимо да се използва отделен атрибут за първичен ключ – номер на композиция.

За нуждите на отчетността и проследимостта на промените, така създаденият модела следва да се разшири с цел да да поддържа граф на верссите. В ER моделите […] е прието графовата структура да се моделира от две същности – същност на възлите и същност на дъгите. От определението на версионизиран обект, може да се заключи, че върховете на графа на версиите са реализирани от същността версия на обект. Липсващото звено може да се реализира от нова допълнителна същност, която ще се грижи за маркиране на дъгите на грава - дъга на граф на версия. Новата същността е необходимо да поддържа следните атрибути:

* Номер на дъгата – първичен ключ за същността
* Глобален номер на изходната версия
* Глобален номер на целевата версия
* Потребител, извършил промяната
* Дата и час на промяната
* Допълнителни данни относно промяната

На Фиг. 22 е представена ER диаграмата на построеният модел на версионизиран обект.



Фиг. ER модел на версионизиран обект

### Версионизиране на съставен версионизиран обект

Настоящата подсекция има за цел да представи особеностите при управлението на версия на съставни обекти от първи ред. Базирайки се на тях, ще се определи процесът на версионизиране на съставни обекти от ред N. Определението по-долу ни дава значението на термина ред на съставен обект.

**Определение:** Съставен обект от ред **0** ще наричаме такъв обект, за който няма асоциирани под-обекти, т.е. фактически той не се явява съставен обект. Съставен обект от ред **N** ще наричаме такъв обект, за който най-големият ред на асоцииран под-обект е равен на **N-1**.

Важно е да се вземе под внимание факта, че в дефиницияна на съставен обект (по-горе) не са налагани никакви ограничения над под-обектите, от което може да се изведе следствието:

**Следствие:** Един под-обект сам по себе си може да се явява съставен обект от други обекти, като по този начин да се създаде суперпозиция от съставни обекти.

Една от основните задачи, която стои пред настоящият научно-приложен труд е да не усложняваме без необходимост тук създадените модели. Изхождайки от това, както и на факта от липсваща практическа необходимост, при построяването на суперпозиция от съставни обекти следва да въведем следното ограничаващи правила (ограничения):

**Правило:** В дадена суперпозиция от съставни обекти, обект може да присъства най-много един път.

**Правило:** Един обект може да присъства най-много в една суперпозиция от обекти



Фиг. Дърво от обекти

При промяна в съставността между два обекта, то следва да разглеждаме версиите на обектите като различни (Фиг. 23). Така например нека разгледаме един стол (супер-обект) с подлакътници (под-обекти). Когато отделим подлакътниците от стола, ние получаваме нова версия на стола – стол без подлакътници. Трябва да отбележим, че конкретният под-обект не си променя версията, т.е. в нашият случай подлакътниците си остават подлакътници. Така се получава единствено промяна в композициите на супер-обекта. Подобно е положението и при „сглобяването” на композиран обект, къде то имаме промяна на версията само на обекта, който става супер-обект. Така например от един стол без подлакътници, при добавяне на подлакътници, ние получаваме нова версия на стола, без да има промяна във версията на подлакътниците.



Фиг. Промяна в композицията на обекти, чрез промяна на версия

Друга особеност при съставните обекти, която непременно следва да разгледаме е, че при промяна на под-обект (т.е. промяна на неговата версия), се получава индиректна променя в съставния обекта (Фиг. 24). Така например при промяна цвета на тапицерията на един стол от червен към син, ние на практика освен новата версия на по-обекта получаваме нова версия на целият стол – стол със синя тапицерия. Като частни случаи на промяна на под-обект може да се приеме асоциирането на обект като под-обект, както и премахването на асоциация с под-обект и неговата (на под-обекта) трансформация като нормален обект.



Фиг. Индиректна промяна версията на съставен-обект при промяна на съставящ-обект

При обратното положение – при промяна версията на супер-обекта – не означава че имаме промяна във версията на съставящите го под-обекти. Така че ако имаме стол с три крака и червена тапицерия, то добавянето на четвърти крак към стола не води до променя версията (цвета) на под-обекта тапицерия (Фиг. 26) .



Фиг. Промяната на супер-обекта, не влияе на версията на под-обекта

Като следствие от последните две правила, може да се изведе следствието:

**Следствие:** При промяна на версията на даден под-обект за даден супер-обект, това не влиае на версиите на другите под-обекти, съставящи супер-обекта (Фиг. 27).



Фиг. При промяна във версията на един под-обект, не се променя версията на съседните под-обекти

The composed object model and the visibility principles from the previous section lead us to the principle of composed object visibility: For a sub-object version is a part of a super-object version composition, the sub-object version is not visible is a workspace where the super-object version is not visible.

### Примерен модел на същността „клас”

В настоящата подсекция демонстрираме възможностите на ER модела на версионизиран обект, като се акцентира на подобренията в сравнение с файлово модел за версионизиране [G141, G261]. В резултат на извършеният анализ на недостатъците на файловият модел, в [G334] сме представили модел на версионизиран обект, който предоставя възможности за свободно определяне на нивото на гранулираност на версионизираният обект. Това ни предоставя възможност за значително подобряване на версионизирането, както в областта на обектно-ориентираното програмиране, така и в областта на model-driven development. Въпреки насочеността на настоящият научен труд към създаването на сойтуерни продукти, тук представеният модел претенира за универсалност и приложимост и в други области на човешката дейност, които се нуждаят от контрол на версията над дадени артефакти. На Фиг. 28 сме представили диаграма на композиция на версионизирани обекти, съставящи същността клас.

Така ние определяме вътрешна и външна структура на класа, където външната структура е определяща за неговото взаимодействие с външният свят – други класове, които са необходими за нормалната работа на нашият клас, или на които нашият клас влияе. Вътрешната структура от своя страна определя вътрешното поведение и взаимодействие на класа. Като базови елементи на класа можем да отбележим полетата и методите.

При изграждането на модела сме идентифицирали мета-атрибутите на класа. Един клас се идентифицира уникално чрез своето име и пакет, в който той се намира. Като типа на класа сме посочили абстрактен, интерфейс, нормален клас, финален клас и вътрешен клас, като за вътрешният клас имаме допълнителен атрибут – външен клас. Някои езици за програмиране, като C++ позволяват да се наследява повече от един клас [G294], докато други, подобно на Java [G298], реализират полиморфизма чрез използването на интерфейси. Това ни довежда до идентифицирането на следните два мета-атрибути на същността клас: *списък с наследените класове* и *списък с имплементираните интерфейси*. Принципът на преизползване на кода в съвременните програмни езици се реализира чрез използването на външни класове (библиотеки), като те се декларират за употреба в даденият клас, това е последният мета-атрибут, който идентифицираме на ниво клас – списък с декларираните за употреба класове.

Полетата и методите на класа съдържат следните общи мета-атрибути: име, тип, тип на капсулираност, тип на достъпа. Типът на достъпа определя начинът за достъп до съответният елемент, той може да бъде през инстанция или през самият клас. Типа на капсулираност определя нивото на достъп до съответният елемент. Най-често използваните типове на капсулираност в обектно-ориентираните езици са:

* частен – елемента е достъпен само в рамките на класа;
* защитен – елемента е достъпен само в рамките на класа и неговите наследници;
* пакетен – елемента е достъпен само за класове от пакета на даденият клас;
* публичен – елемента е достъпен за всички класове.



Фиг. Примерен модел на композиране на версионизирани обекти за същността клас

За същността метод можем да посочим следните мета-атрибути: тип на върнатият резултат, списък с параметрите, тяло на метода, тип на презаписване. Списъкът с параметри съдържа параметри, всеки от които се идентифицира уникално по своята позиция в списъка и от типа на параметъра. Използвайки типа на презаписване, ние определяме, дали методът е нормален, финален (т.е. не може да бъде презаписван в наследниците на класа) или абстрактен (т.е. трябва да бъде презаписан в наследниците на класа).

В новите обектно-ориентирани езици, като C#, ние забелязваме следващата стъпка в еволюционно развитие на същността поле. Новото тук е появата на специализирани методи, познати като извличащ метод (getters) и записващ метод (setters). Те ни позволяват да дефинираме специфично поведение на класа при опит за достъп до полетата му. Факта, че тези методи споделят общ интерфейс ни позволява да ги причислим към същността поле [G334].

Така построеният модел използва в пълна степен предимствата на фината гранулираност на версионизираните обекти, което се явява основен недостатък при файлвови-базираните версионизирани обекти...

## Йерархично композирани работни пространства. Модел на видимост на версионизирани обекти

### Модел на йерархично композирани работни пространства

… продукт – издание – главно работно пространство и йерархия от работни пространства.

ограничение, един обект може да присъства само с една версия в дадено работно пространство

### Модел на видимост на версионизирани обекти в среда с йерархично композиране на работни пространства

Настоящето научно изследване се базира на използването на набор от пространства, подредени в йерархична композиция (дървовидна). Както във всяка йерархична структура, така и тук ще се разглеждат отношения родител-наследник, като се прави акцент на видимостта на версионизираните обекти, определена от представени принципи на видимост.



Фиг. Примерна йерархична композиция на пространства

Под видима версия на версионизиран обект ще се разбира версията или липсата на версия за обекта, с която потребителят (разработчикът) работи в рамките на определено пространство, което от своя страна се нарича локално работно пространство. Под локална версия на версионизиран обект за дадено работно пространство, ще се разбираме такава негова версия, която е асоциирана с работното пространство.

Принципи на видимост:

1. Ако даден версионизиран обект има версия в рамките на дадено пространство (локална версия), то в това пространство се вижда само тази версия на обекта, въпреки наличието на други версии в родителските пространства.
2. Локалната версия на обект от дадено работно пространство се вижда рекурсивно във всички под-пространства, освен ако няма дефинирана друга локална версия в тях.

От изложените принципи можем да изведем следствията:

1. В всяко работно пространство, където обектите нямат локална версия, са представени с тяхна версия намираща се в най-близкото родителско работно пространство.
2. Ако за дадено работно пространство обекта няма версия в нито едно родителско работно пространство, то той не се вижда в първоначално избраното работно пространство.

На Фиг. 30 нагледно са представени двата принципа на видимост на версии в една примерна йерархична конфигурация от работни пространства. Със стрелки сме показали посоките на разпространение на видимост на отделните версии на обекта.



Фиг. Разпределение на версиите на версионизиран обект съгласно принципите на видимост

## Транзакции над версионизиран обекти

### Транзакции над версионизиран обект в рамките на едно работно пространство

В тази секция е направен опит да се представят транзакциите над версионизирани обектяи в рамките на едно работно пространство. По-долу са разгледани следните транзакции: създаване на версионизиран обект; актуализиране на не локален версионизиран обект; създаване на маркер на състояние (state-mark) на локален версионизиран обект, маркер на състояние изтрит обект и отказ от маркер на състояние.

Създаването е първата транзакция за всеки един версионизиран обект. След изпълнението на транзакцията, обектът притежава първоначална (нулева) версия, в която той е „празен”, т.е. не съдържа никаква информация.

Създаването на маркер на състояние представлява транзакция, при която се създава нова версия на даден версионизиран обект. Тази транзакция може да се разглежда, като основа на механизъм за създаване на сигурни точки (safe-point).

Като обратна транзакция за създаване на състояние може да се квалифицира тази по отказ от маркер на състояние. Чрез нея в представеният модел последното състояние се освобождава, а текуща локална версия на обекта става версията предхождаща отказаната.

Създаването на дълги редици от последователни и неразклонени версии, особено от един и същи потребител в рамките на едно и също работно пространство, ни навежда до идентифицирането на транзакция по обединяване на последователни версии, с цел икономия на памет и последваща по-бърз и по-лесен анализ на извършената работа.

Актуализирането на не-локален версионизиран обект, т.е. на обект, който няма локална версия в текущото работно пространство, може да се определи като най-важната функционалност на текущата секция. Тази транзакция не е напълно ограничена само до едно работно пространство, тъй като тя е съставена от следните стъпки:

* Извличане на предишната видима за работното пространство версия на обекта;
* Създаване на локална версия на обекта в текущото работно пространство;
* Създаване на релация на версиите (дъга в графа на версиите), в която предишната видима версия се явява версия-първоизточник за новата локална версия на обекта.

Изтриването на даден обект е възможно чрез транзакция за създаване на т.нар. маркер за изтрит обект. Този маркер има за цел да „скрие” обекта в работното пространство и той да стане невидим в текущото работно пространство, както и за неговите под-пространства. Следва да се отбележи, че всички описани в тази секция транзакции над обекта вече не могат да се извършват, с изключение на транзакцията по отказ от маркер на състояние.

### Транзакции над версионизиран обект между две работни пространства

Транзакциите между две работни пространства може да се разделят на две групи – публикуване на версия на обект и отказ от локална версия. Преди да ги разгледаме следва да въведем термините производна и паралелна (непроизводна) версия на обект.

**Дефиниция:** Нека разгледаме един версионизиран обект и две негови версии X и Y. Ако съществува път в графа на весии на обекта от версия Х до версия Y, то версия Y се явява производна версия на версия Х, а версия на Х – предшестваща версия Y.

**Дефиниция:** Нека разгледаме един версионизиран обект и две негови версии X и Y. Ако не съществува път в графа на версиите за обекта от версия Х до версия Y, то двете версии се явяват паралелни или непроизводни.

**Дефиниция:** Под публикуване на версия на обект, ще се разбира поредицата от действия, необходими за привеждане локалната версия на обекта от текущото работно прострастнво в локална версия в родителското работно пространство.

Простото публикуване на версия е при ситуация, когато в родетивското работно пространство не съществува локална версия на публикуваният обект – Фиг. 31.



Фиг. Просто публикуване

Следващата по сложност транзакция е тази за актуализиращо публикуване (Фиг. 32). Характерно при нея е, че тя е възможна, когато бъдат едновременно изпълнени следните две условия:

* в родителското работно пространство съществува локална версия на обекта, който се публикува;
* версията на обекта, който се публикува, се явява производна на версията му в родителското работно пространство.

При актуализиращото публикуване не се извършва сливане между двете версии понеже производната версия представлява еволюционно продължение на предшестващата версия. Следва да се отбележи,че след публикуването на новата версия, тя става видима във всички работни под-пространства на родителското, където няма локална версия на разглужданият обект



Фиг. Актуализиращо публикуване

Когато версията обекта, която се публикува в родителското работно пространство се явява паралелна спрямо намиращата се там локална версия (Фиг. 33), тогава следва двете версии да се слеят. Като резултат на сливането се получава нова версия на обекта. Настоящото научно изследване няма за цел да представи някакъв нов метод за сливане на версии на обект, затова тук може да бъде използван както ръчен подход за сливане, така и алгоритмичен подход, подобен на споменатият по-рано алгоритъм на Вестфехтел [G109], two-way merge или three-way merge [G360].



Фиг. Публикуване със сливане

Транзакцията по отказ от локална версия явява обратна на транзакциите по публикуване на версия. Тя включва само една стъпка: премахване на локалната версия на обекта от работното пространство. При премахването сработват механизмите от Следствие 1 от модела на видимост на версионизираните обекти (2.2.2). Важно е да се отбележи, че ако в нито едно от родителските работни пространства не съществува версия на избраният обект, то той става недостъпен за последваща употреба, т.е. той се изтрива. Това полоение следва да се отчита, когато транзакцията се извършва в главното работно пространство на изданието на продукта.

### Транзакции над съставни обекти

В настоящата подсекция има за цел да определят принципите и особеностите при изпълнение на транзакции над съставен обект особено след проманя в неговата композиция.

Нека разгледаме ситуацията, когато имаме локална версия на обекта В в родителското работно пространство и негова видима версия в текущото работно пространство. В текущото работно пространство се създава под-обект А за обекта В (Фиг. 34). При публикуването версията на под-обекта А е възможно и тя не води до някаква промяна във версията на обект В в родителското работно пространство. Въпреки това при последващо публикуване версията на обекта В заедно с неговите композиции, в родителското работно пространство ще доведе до автоматично обновяване (в рамките на работното пространство) на композиционната схема на обектите (Фиг. 33 – зелената пунктирана стрелка). Това е продиктувано от факта, че информацията относно организацията на съставният обект следва да се разглежда като неделима част от него.

При публикуване на новата версия на съставният обект (B,v3) води до изискването това да се извърши в комплект с версията на новосъздаденият под-обект (Фиг. 33 – стрелките с №2).



Фиг. Новосъздаен под-обект към супер-обект

Нека се разгледа ситуацията, когато имаме локална версия на обекта В в родителското работно пространство, която е видима в текущото работно пространство (Фиг. 35). В текущото работно пространство се извършва промяна в в под-обекта А, която води до промяна на обекта В. Именно създаването на нова локална версия на под-обекта води до автоматичното създаване на нова локална версия на съставният обект. Следва да се отбележи, че самостоятелното публикуване на новата версия на под-обекта в родителското работно пространство би следвало да не се допуска. Това ограничение следва от факта, че наличието на нова версия на под-обекта, предполага наличието на нова версия на супер-обекта (Фиг. 35 – червената стрелка с №1), а също така и от ограничението, че един обект може да присъства само с една версия в дадено работно пространство. Като извод в разглежданата ситуация може да се определи следното:

**Правило (П-2.3.3-1):** Публикуването на версия на локален съставен обект следва да се извършва в комплект с всички локални версии на неговите под-обекти, които имат различна версия в родителското работно пространство (Фиг. 35 – зелената и жълтата стрелки с №2).

Локалните версии на под-обекти в родителското работно пространство могат да бъдат както производни, така и паралелни. В тези случаи е необходимо да се изпълнят съответните транзакции, кояито бяха разгледни в предишната под-секция.



Фиг. Индиректна променена версия на супер-обект, породена от нова версия на под-обект

Нека се разгледа ситуацията, когато имаме локална версия на обекта В в родителското работно пространство, която е видима в текущото работно пространство (Фиг. 36). От композицията на съставен обект се изключва под-обект. За да се отрази тази промяна в родителското работно пространство, е достижимо единствено чрез публикуване на съставният обект. Тази публикация води само до отразяване на промяната в композицията в родителското работно пространство, без да се променя версията на под-обект, който в новата версия на супер-обекта, вече не съставна част от него.



Фиг. Липса на промяна във версията на обект А, т.е. няма необходимост от неговото публикуване

Нека имаме видим съставен обект А с под-обект В, като обекта А и под-обекта В са локални версии в родителското пространство. От композицията на даден съставен обект А премахвахнем под-обекта В. След това потребителят модифицира обекта В, т.е. създава нова негова версия (Фиг. 36). При така създадената ситуация публикуването новата версия на под-обекта В би довела до следната конфликтна ситуация: Версия v1 на обект В изисква в работното пространство обекта А да присъства (видимо или локално) с версия v1.



Фиг. Публикуване на бивш под-обект не е възможно, преди публикуване новата версия на бившия супер-обект

Този факт може да се разглежда като предпоставка за формулирането на правилото:

**Правило (П-2.3.3.-2):** Публикуването на версия на обект, който притежава предишна версия, явяваща се под-обект на съставен обект в родителското работно пространство на текущото работно пространство, следва да се извършва едновременно с публикуването на локалната версия на съответният съставен обект.

### Класификация на транзакциите над версионизирани обекти

При работата си в среда с йерархично композирани работни пространства, потребителите извършват модификации над обектите в тях. С цел систематизирането и класифицирането на тези действия в текущата под-секция класификация на транзакциите над версионизирани обекти в среда от йерархично композирани работни пространства. В таблицата по-долу у представена класификацията.

|  |  |  |
| --- | --- | --- |
| **Групиране** | **Тип обекти** | **Транзакции** |
| Транзакции в рамките на едно работно пространство | Прости обекти | Създаване на версионизиран обект |
| Маркиране на версия |
| Актуализация на не локален версионизиран обект |
| Маркиране на изтрито състояние |
| Съставни обекти | Включване на обект в композицията на съставен обект. |
| Автоматично регистриране на индиректна нова версия на съставен-обект, породена от нова версия на под-обект |
| Изваждане на под-обект от композицията на съставен обект |
| Транзакции между две работни пространства | Прости обекти | Просто публикуване |
| Актуализиращо публикуване |
| Публикуване със сливане |
| Отказ от локална версия |
| Съставни обекти | Публикуване на съставен обект |
| Публикуване на обект, който е изваден от композицията на съставен обект. |

Таблица Класификация на транзакциите над версионизиран обект в среда с йерархична композиция на простраства

Двете основни групи транзакции, които са идентифицирани са транзакциите в рамките на едно работно пространство и тези, които изискват за своето изпълнение да се включат две работни пространства. Ако първата група транзакции е свързана с класическите задачи от управление на версии като: създаването на обекти, техните промени, както и изтриването им (маркирането като изтрити), то втората група транзакции са свързани именно с организиране на съвместната работа на участниците в проекта.

### Жизнен цикъл на версионизиран обект

Под жизнен цикъл ние ще разбираме последователността от събития и състояния на даден обект от неговото съзване, до неговото изтриване - изтегляне от употреба. За описване на жизнен цикъл на версионизиран обект, са използвани представените в 2.3.4 транзакции. На фигура Фиг. 37 е представен модел на жизненият цикъл на версионизиран обект, чрез използванета на диаграма на поредица от събития (event-driven chain). Моделът включва следващите етапи:

* Създаване на версионизиран обект;
* Създаване на версия на обект;
* Сливане на версии на обект;
* Маркиране на обект, като изтрит;
* Отказ от локална версия на обект, при условие, че съществува негова локална версия в някое от родителските работни пространства;
* Отказ от локална версия на обект, при условие, че в нито едно родителско работно пространство той няма локална версия.



Фиг. 37 Жизнен цикъл на версионизиран обект

Следва да се изтъкне факта, че транзакциите

В представената диаграма можем да изтъкнем следните особености:

* Жизненият цикъл на всеки версионизиран обект започва с неговото създаване.
* Като крайна транзакции на версионизиран обект може да се отбележат „Маркиране на изтриване”. Тези връзки са отбелязани с червени стрелки на диаграмата
* Транзакцията „Отказ от локална версия” играе роля на крайна за жизненият цикъл на обекта, ако във всички родителски работни пространства този обект няма версии. Тези връзки са отбелязани с оранжево на диаграмата.
* Транзакцията „Маркиране на изтриване” е възможно да не е крайна за жизненият цикъл на обекта, при условие че в родителските работни пространства съществува версия на обекта и се приложи транзакцията „Отказ от локална верния”. Особеното между двете споменати транзакции е, че ако в родителските работни пространства не съществува версия на обекта, то прилагането на „Отказ от локална версия” не е възможно (безсмислено). Поради тази причина на диаграмата връзката между двете е отбелязано с пунктерана стрелка.
* Трите транзакции свързани с публикуването на обектите в родителското работно пространство, отбелязани със син цвят на диаграмата, могат да се изпълняват последователно и в зависимост от конкретната версия на обектите между родителското и дъщерното работно пространство автоматично се прилага една от трите представени конфигурации.

## Проследимост на промените в среда с йерархична композиция на работни пространства

Една от задачите, пред които са поставени съвременните методологии и практики по управление на промените разгледани в глава 1, е задачата за анализирането на промените. Настоящата секция е фокусирана на създаването на модели подпомагащи анализа и проследяване на промените. Основните причините, които предизвикват промяна в един софтуерен продукт са две: промяна в изискванията към системата и откриване на неправилно или дефектно функциониране на системата, изикващо да се извърши корекция. Тези източници на промени в нашият модел ще ги наричаме *причини*. За извъшването на пълноценен анализ, причините следва да бъдат асоциирани с извършените промени над обектите, които могат да се дефинират като *следствие*.



Фиг. Проследяваща връзка между причината и следствието

От направеният анализ в глава 1 на методите за проследимост на промени, може да се определи като метод притежаващ най-висок потенциал за развитие този базиран на събития. В текущата секция ще бъде направен опит за адаптиране и интегриране на метода в разглежданата среда с йерархична композиция на работни пространства. Моделирането на метода за проследяване на промените може да се разглежда като допълнително звено, свързващо моделът на работните пространства с този на версинизиран обект, като се запази възможността за гъвкаво дефиниране нивото на гранулираност на обектите, с които се работи.

### Работни единици и работни пространства

В настоящата подсекция е представно основно понятие в процеса на създаване на софтуер – работна единица. Тук под работната единица ще се използва разширение на определението, което дава Хелминг в своят труд [G311] – Работна единица е работата, която следва да се извърши.

**Определение:** Работна единица се нарича съвкупността от дейности (работата), определена от дадена причина, която следва да се извърши.

**Определение:** Следствието от причината ще се нарича набора от промени над обекти, в следствие от изпълнението на съответната работна единица.

Диаграма – нива на терминологията

Следва да се отбележи, че формулираната работна единица е резултат от работата по определяне на задачите, извършена от мениджъра. Работните единици освен са средство за определяне и разпределение на задачите.

Метода на проследимост базиран на събития включва два етапа:

* Настройване на средата за генериране на проследяващи събития.
* Прихващане на събития и създаване на проследяващи връзки.

В представеният подход това е включва следните две подстъпки:

1. Определяне на версионизираният обект като работна единица.
2. Подготвяне на работното пространсто за автоматично генериране на проследяващи връзки, чрез активиране или деактивиране на необходимите работни единици към работното пространство.

След като е извърването на втората подстъпка, представеният модел е способен да автоматично прихваща събитията на промяна на обектите

After second step our model is able to catch the events of versioned object change and automatically to create traceability links with initiators all attached to the workspace work items. We have to count that some work items could be a composed object. On attaching of composed workitem we will assume that the team member wants to perform changes caused also by all sub-work items no matter the composition depth.



Фиг. …

### Модели на данните на система за управление на версията чрез йерархични пространства

Моделът, описан в настоящата подсекция има за цел да представи един еволюционен подход при изграждане на модел... Първоначално представеният модел в [...] и разгледан в 0предоставя възможност за ... В по-късното ни изследване [...] ние развиваме модела до ниво, което ни предоставя възможност да се реализира адапта – проследимост на промените. Основната задача е да се запази до голяма степен възможностите на вече създаденият модел. Във връзка с това ние добавяме една допълнителна същност, както и две релации – същността „Работна единица”, и релациите „Асоциирани работни единици” и „Връзка на проследимост” [G364].

Същността „Работна единица”...

Релацията „Асоциирани работни единици” определя кои от работните единици се гледат като предпоставка към

The model present in [5] is extended for supporting event-based traceability method described in current paper. Main goals of that extension are to keep all current system capabilities and to guard further system extension feature of system data model. Therefore only one entity and to relationships are added– work item; attached work item; traceability link



Фиг. ER модел за събитийно-базирана проследимост на промени

New entity provides support for workitem without changing the infrastructure if versioned object entity. New relations follow self explained naming convention

...

## Изводи

Анализирайки построеният модел на версионизиран обект може да се изтъкнат следните предимства в сравнение с файлово базирания модел, както и другите модели разгледани в първа глава:

* Простота на модела. Това предполага универсалност и пълнота на модела, която се изразява в неговата приложимост при различни практически задачи.
* Адаптивност на модела спрямо конкретиката на задачите, които следва да се решават. Следващата подсекция демонстрира от практически тази страна на модела.
* Разширяемост на модела, която сме демонстрирали в ...
* Свобода при определяне степента на гранулираността на обектите, с които се работи.
* Свобода при композирането на обектите, което би следвало да реши проблемите, стоящи при разработването движимо от модели [G360]

В допълнение можем да отбележим, че фината гранулираност на обектите намалява вероятността за един и същ версионизиран обект да се получи конфликт между неговите версии, понеже намалява вероятността той да бъде едновременно променян от двама или повече потребителя. Това позволява да прилагане на по-ефективни алгоритми при сливане на версии, които също така и по-рядко ще има нужда да се прилагат.

В настоящата глава разгледахме ER модел на версионизиран обект, както и модел на йерархично композирани работни пространства. В допълнение към тези модели бяха представени модел на жизнен цикъл на версионизиран обект, в който централно място заемат класифицирани транзакции. В допълнение към това особено място е отделено на особеностите при версиониране на композирани обекти.

Накрая, но не по важност е представена адаптация на метод за проследяване на промени, базиран на събития. Адаптацията на този метод използва в голяма степен композираността на пространствата и на обектите, като механизъм за разбиване на големи задачи на по-малки и тяхното решаване. Проследяването позволява в пълнота да се обхване целият процес не само в софтуерното производство, но и в други отрасли, при които честата промяна на артефактите се счита за нормален работен процес.

...

# Глава трета Изследване приложимостта на моделите

Средите за разработка на софтуерни системи постоянно се развиват, като предоставят все повече инструменти и библиотеки за автоматизиране и ускоряване процеса на разработка. Основните направления, които може да се споменат са: автоматично дописване на код, възможности за постъпково изпълнение, анализ производителността на изходният код, средства за интерактивно моделиране.

Във втора глава са представени модели за версионизиране на обекти в среда с йерархично композиране на работни пространств. Целта е те да преминат експериментална пробация за увеличаване на ефективността на работният процес при създаването на софтуернти продукти, чрез увеличаване степеннта на неговата автоматизация.

## Възможности за реализиране на моделите

При реализацията на софтуерен продукт с инфраструктурно значение следва да се извършва при използването на технологии и инструменти, позволяващи лесно разширяване, поддръжка и интеграция с други системи от същото инфраструктурно ниво. При изборът на платформа и среда [G367, G368, G372, G369, G382, G381, G384] за разработка, бяха взети в предвид следните критерии:

* Познаване на езикът – това до голяма степен скоростта на разработка на прототипа
* Възможност за бързо и лесно използване на съвременни технологии
* Скорост на разработка и удобство на интерфейса.
* Познаването и възможности на библиотеки до голяма степен е субективен критерий, който има съществено значение за успешното разработване на прототипа.
* Възможности за разгръщане и демонстрация на прототипа.

При определяне на технология за разработване бяха разгледани PHP, JavaEE и C# DotNet. В таблица 2 са представени оценки по отделните критерии, като рейтингът на всяка една технология представлява сума от отделните оценки.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Познаване на езика** | **Възможност за използване на съвременни технологии** | **Скорост на разработка** | **Библиотеки** | **Демонстрация** | **Рейтинг** |
| **JavaЕЕ** | 6 | 6 | 5 | 5 | 5 | 27 |
| **C# .Net** | 3 | 6 | 6 | 5 | 3 | 23 |
| **PHP** | 6 | 4 | 4 | 4 | 6 | 24 |

таблица Рейтинг на платформите

В резултат на направеният анализ бе избрана JavaEE, като платформа с най-висок рейтинг, и потенциално най-високи възможности за бързо разработване на прототип.



Фиг. Архитектура на J2EE платформата

Технологиите позволяващи да се ускори работата в J2EE платформата може да се разделят на две основни - на такива, които предоставят механизъм за управление на потребителският интерфейс и бизнес логиката, и на технологии за достъп до данните.

При анализирането на J2EE технологиите за управление на потребиталският интерфейс и бизнес логика е направена оценка съгласно следните основни критерии:

* Удобство за използване на технолокията
* Скорост на разработване, която се постига в резултат от прилагането на технологията
* Гъвкавост при последващи промени и модификации – този критерий заема особено място при разработването на нови системи, при които степента на неопределеност е висока [да намеря источник за това твърдение].
* Познаване на технологията – субективен критерий, който има тежест най-вече при началният етап на разработването.

В своето изследване Барцис [G385] определя JSF технологията като по-гъвкава от

Като технология на представяне и контролери са разгледани и анализирани [G368, G385, G180, G373, G379] следните: Servlet/JSP, Struts, JSF, Spring, като резултатът от направеният анализ е представен в таблица 3:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Удобство** | **Скорост** | **Гъвкавост** | **Познаване** | **Рейтинг** |
| Servlet/JSP | 3 | 2 | 3 | 6 | 14 |
| Struts | 4 | 4 | 4 | 2 | 14 |
| JSF | 5 | 5 | 6 | 6 | 23 |
| Spring | 5 | 5 | 6 | 4 | 20 |

таблица Рейтинг на J2EE технологиите

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Удобство** | **Скорост на разработка** | **Скорост на решението** | **Гъвкавост** | **Рейтинг** |
| JDBC | 1 | 2 | 6 | 2 | 11 |
| Hibernate |  |  |  |  |  |
| OpenJPA |  |  |  |  |  |

[G180 – Hibernate]

Сравнение на Faces библиотеките и JSP J2EE.

- Spring

Малък увод на J2EE платформата, преход към WEB технологиите [G368 133стр. 141стр.]. Недостатъци/слабости на JSP и сървлет технологиите и необходимост от появяване на нови технологии- Struts [G373], Spring[…], Java Server Faces Face (JSF)[G379] технологията - сравнение.

Като най-популярни JSF библиотеки могат да се определят RichFaces на JBoss [G370] [G375]-има за Struts&Spring; jBoss RichFaces; ICEfaces; MyFaces [<http://www.jsfcentral.com/articles/geiler-04-04.html>],

Сравнение на ОРМ/JPA библиотеките [G371, G374]. EclipseLink; [JBoss Hibernate](http://en.wikipedia.org/wiki/Hibernate_(Java)); OpenJPA [G374]

Вторият основен компонент при разработването на прототипа представлява платформата за бази данни. При изборът на подходяща база бяха определени следните критерии:

* Опит със съответната база
* Лекота на разработка
* Функционални възможности на базата
* Производителност
* Интеграция с избраната платформа за разработка (Java & JPA)
* Възможност за използване при демонстрация на продукта

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Опит** | **Скорост на разработка** | **Функционални възможности** | **Производителност** | **Интеграция** | **Демонстрация** | **Рейтинг** |
| Oracle | 5 | 5 | 6 | 6 | 6 | 4 | 32 |
| Ms SQL | 4 | 6 | 5 | 5 | 5 | 4 | 29 |
| MySQL | 6 | 4 | 4 | 5 | 4 | 6 | 29 |

таблица Рейтинг на СУБД

[G376] – препоръчват Oracle

[G377] – Сред трите разглеждани платформи, Oracle предлага най-добре възможности

[G380] – Comparison of relational database management systems

[G378] – Stored Procedures, and Code Generation in MySQL, MsSQL и Oracle

* Функционални възможности на платформата
* Възможност за интеграция с библиотеките за
* Цена
* Възможности за разгръщане и демонстрация на прототипа.
* Помощ и поддръжка

## Избор на технологии и за реализация

...

## Разработка на прототип на система за управление на версии

### Избор на софтуерен инструментариум и определяне процесът на разработка.

При разработката на прототип на система за управление на версия с йерархично композирани работни пространства бе избрана средата Eclipse, която предоставя добри интрументи за бърза разработка. Средата Eclipse предоставя и много добра интеграция с други системи, които се използват при разработването на софтуерни продукти. При разработване на прототипа е използвана безплатната система за управление на версията Subversion на code.google.com. За разгръщането на приложението са използвани Ant […] сценарии, които са авноматично изпълнявани от Hudson[…]. Сценариите указват и мястото за разгръщане на Tomcat сървър. След разгръщането на прототипа, той може да бъде използван от клиентските машини, чрез уеб интерфейс. На Фиг. 42 е представена схема на процесът от разработването, през разгръщането, до използването на системата от крайните потребители.



Фиг. Процес на разработка, разгръщане и използване на прототипа

### Архитектурен модел

При разработването на прототип на системата е използван модифицирана MVC архитектура, като са определени следните фукционални компоненти:

* Слой на базата данни



Фиг. Архитектура на система-прототип Versia

* Слой на достъпа до базата данни, който е представен от библиотеката OpenJPA. В оригиналната MVC архитектура този слой липсва, а тук той е въведен поради функционалните характеристики на Java запазващата (persistent) библиотека OpenJPA. Това се изразява във факта, че тя позволява бърз и лесен механизъм за промяна на базата от данни, използвана в по-ниският слой – слоят на базата от данни, като това се извършва чрез промяна в конфигурационният файл на библиотеката.   
  Следва да се отбележи важна характеристика на библиотеката OpenJPA – тя позволява да се автоматично създаване и актуализиране на съответствието на модела на предметната област и схемата на базата от данни. Така при добавяне на ново свойство за даден обект, библиотеката в автоматичен режим може да създаде в съответната таблица нова колона (без допълнителната намеса на администратор). Този факт до голяма степен автоматизира процеса на разгръщане на системата и води до минимизиране на риска от появяване на грешки в прозукционната среда.
* Слой на предметната област – представлява реализация на предметната област чрез използването POJO (Plain Old Java Object – Обикновен Стар Java Обект). Необходимостта от използването на POJO е продиктувано от факта, че всички Java запазващи (persistent) библиотеки изискват предметната област да е реализирана именно чрез такива обекти.
* Слоей на бизнес логиката. Реализацията на бизнес логиката е чрез използването на Java Beans компоненти (наричани по-долу компоненти на бизнес логика)
* Слой на представянето реализиран под формата JSF страници, в които се използват специализирани тагове (tags) от библиотеката RichFaces. Именно чрез тези специализирани тагове се осъществява интеграцията с компонентите за бизнес логика. Те също така предоставят възможност за използване на богати компоненти, като календар, избор на елементи, таблици и др.
* Слой на клиентския интерфейс може да се обособи като отделен слой, използващ готовият инструментариум на библиотеката RichFaces. Към този слой може се отнесе конфигурацията на преходите между екраните, както и конфигурацията на компоненти реализиращи бизнес логиката. Не на последно място именно тук се намира поддръжката на Aсинхронните JavaScript и XML (AJAX) заявки.

### Модел на класовете

Поради наличието на голям брой класове в рамките на прототипа, тяхното представяне е реализирано на няколко етапа – класове реализиращи предметната област...

Класовете за представяне на предметната област са реализирани в пакета … Техният модел до голяма степен повтарят модела представен в 2.1, 2.2 и 2.4.



Фиг. Клас диаграма на предметната област

...

### Навигационен модел

...

## Сравнителен анализ преимущестата на прототипа

За целите на оценяването преимуществата на разработеният прототип е предприето експериментално решаване на задача (проект) от една страна с използването на стандартни средства, а от друга – със средствата на прототипа. При експерименталното решаване на задачата е използвана еднаква методология, за разработване на софтуени продукти, с цел да се постигне чистота на експеримента.

### Постановка на задачата на експеримента

В рамките на експеримента е заложено:

* Определяне на изискванията и разработка на продукт
* Намирането и отстраняване на дефект
* Промяна в едно от първоначалните изистквания и неговата реализация
* Създаване на ново изискване, непряко свързано със съществуващо такова.

### Решаване на задачата с използване на съществуващите системи и подходи

...

### Решаване на задачата при използването средствата и подхода на прототипа

...

## Изводи

...

# Заключение

В настоящата дисертация представихме моделите за контрол и управление на софтуерна версия.

Теоритичните разработки, представени в Глава II, бяха насочени в три основни направления – модел на версионизиран обект, модел на среда за версионизиране, базирана на йерархична композиция от работни пространства, и модел за полу-автоматично съзвадане на проследяващи връзки. Предложените модели предоставят възможност за допълнително автоматизиране процеса на разработка на софтуерни продукти, и ... фокусиране на замесените страни върху задачите, над които работят, а не на процеса на отчитане на свършената работа.

# Литература

[G327] A Giude to the Project Management Body of Knowledge (Version 4.0), Project Management Incorporated, Inc., Pennsylvania, USA, 2008, 496 p.

[G326] Arveson P., The Deming Cycle, http://www.balancedscorecard.org/TheDemingCycle/tabid/112/Default.aspx, accessed 24.02.2010

[G377] Arvin, Troels,Comparison of different SQL implementations, 2011, http://troels.arvin.dk/db/rdbms/

[G312] Asun ion, H. U. 2008. Towards practical software traceability. In Companion of the 30th international Conference on Software Engineering (Leipzig, Germany, May 10 - 18, 2008). ICSE Companion '08. ACM, New York, NY, 1023-1026. DOI= http://doi.acm.org/10.1145/1370175.1370228

[G385] Barcis, Roland, JavaServer Faces (JSF) vs Struts, 2004, http://websphere.sys-con.com/node/46516 (accessed 18.12.2011)

[G337] Bhardwaj P., You can’t improve IT, if you are not measuring IT, http://notjustitsm.wordpress.com/2009/11/25/you-cant-improve-it-if-you-are-not-measuring-it/ ,2009 (accessed 13.03.2010)

[G274] Binkley, D., Horwitz, S., and Reps, T. 1995. Program integration for languages with procedure calls. ACM Trans. Softw. Eng. Methodol. 4, 1 (Jan. 1995), 3-35. DOI= http://doi.acm.org/10.1145/201055.201056

[G221] Brown, A., Dart, S., Feiler, P., Wallnau, K.. The state of automated configuration management. Tech. Rep. CMU/SEI-ATR-91, Software Engineering Inst., Carnegie Mellon Univ., Pittsburgh, Pa., Sept 1991

[G344] Buffa, M. and Gandon, F. 2006. SweetWiki: semantic web enabled technologies in Wiki. In Proceedings of the 2006 international Symposium on Wikis (Odense, Denmark, August 21 - 23, 2006). WikiSym '06. ACM, New York, NY, 69-78. DOI= http://doi.acm.org/10.1145/1149453.1149469

[G250] ClearQuest. http://www-01.ibm.com/software/awdtools/ clearquest/

*[G308] Cleland-Huang, J. 2005. Toward improved traceability of non-functional requirements. In Proceedings of the 3rd international Workshop on Traceability in Emerging Forms of Software Engineering (Long Beach, California, November 08 - 08, 2005). TEFSE '05. ACM, New York, NY, 14-19. DOI= http://doi.acm.org/10.1145/1107656.1107660*

[G322] Cleland-Huang, J., Settimi, R., BenKhadra, O., Berezhanskaya, E., and Christina, S. 2005. Goal-centric traceability for managing non-functional requirements. In Proceedings of the 27th international Conference on Software Engineering (St. Louis, MO, USA, May 15 - 21, 2005). ICSE '05. ACM, New York, NY, 362-371. DOI= http://doi.acm.org/10.1145/1062455.1062525

[G101] Collins-Sussman B., Fitzpatrick, B. W., Pilato C. M., Version Control with Subversion, Compiled from r3305, 2008 (достъпена 16 Март 2009)

[G384] Converse, Tim, Joyce Park, PHP Bible, 2nd Edition, 2002, Wiley Publishing, ISBN 0-7645-4955-3 (1012 p)

[G8] Conradi, R. and Westfechtel, B. 1998. Version models for software configuration management. ACM Comput. Surv. 30, 2 (Jun. 1998), 232-282. DOI= http://doi.acm.org/10.1145/280277.280280

[G336] CSI Model, http://www.goworld.co.nz/Solutions/ITIL+Enablement/ Continual+Service+Improvement/CSI+Model.html, (accessed on 13.03.2010)

[G237] Dart, S. 2000 Configuration Management: the Missing Link in Web Engineering. Artech House, Inc.

[G273] Doray, Arnold, Beginning Apache Struts: From Novice to Professional, 2006, Apress, 536 pages, ISBN:9781590596043

[G249] eChangeMan, http://www.serena.com/

[G12] Estublier, J., Leblang, D., Hoek, A., Conradi, R., Clemm, G., Tichy, W., and Wiborg-Weber, D. 2005. Impact of software engineering research on the practice of software configuration management. ACM Trans. Softw. Eng. Methodol. 14, 4 (Oct. 2005), 383-430. DOI= http://doi.acm.org/10.1145/1101815.1101817

[G172] Estublier, J., Vega, G. 2007. Reconciling software configuration management and product data management. In Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (Dubrovnik, Croatia, September 03 - 07, 2007). ESEC-FSE '07. ACM, New York, NY, 265-274. DOI= http://doi.acm.org/10.1145/1287624.1287662

[G14] Estublier, J. 2000. Software configuration management: a roadmap. In Proceedings of the Conference on the Future of Software Engineering (Limerick, Ireland, June 04 - 11, 2000). ICSE '00. ACM Press, New York, NY, 279-289. DOI= http://doi.acm.org/10.1145/336512.336576

[G275] Feiler, Peter P., Configuration Management Models in Commercial Environments, 1991

[G382] Ferguson, Jeff, C# Bible, 2002, John Wiley & Sons, 2002, (808 pages), ISBN:9780764548345

[G381] Gilmore, W. Jason, Beginning PHP and MySQL: From Novice to Professional, Apress, 2010, (825 pages), ISBN:9781430231141

[G311] Helming, J., Koegel, M., and Naughton, H. 2009. Towards traceability from project management to system models. In Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering - Volume 00 (May 18 - 18, 2009). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 11-15. DOI= http://dx.doi.org/10.1109/TEFSE.2009.5069576

[G23] Hicks, D. L., Leggett, J. J., Nürnberg, P. J., and Schnase, J. L. 1998. A hypermedia version control framework. ACM Trans. Inf. Syst. 16, 2 (Apr. 1998), 127-160. DOI= http://doi.acm.org/10.1145/279339.279341

[G315] Hillyer, M., Managing Hierarchical Data in MySQL, 2005, http://dev.mysql.com/tech-resources/articles/hierarchical-data.html (accessed 02.05.2010)

[G310] Jarke, M. 1998. Requirements tracing. Commun. ACM 41, 12 (Dec. 1998), 32-36. DOI= http://doi.acm.org/10.1145/290133.290145

[G232] Johansson, C., The V-Model, 1999, http://www.bucanac.com/ documents/The\_V-Model.pdf (accessed on 19/02/2008)

[G102] Jones, M. T., Version control for Linux, http://www.ibm.com/ developerworks/linux/library/l-vercon/, 2006, (достъпена 13 Февруари 2009)

[G364] Jotov, Vl., Adaptation of Event-Based Traceability Method for Environment with Hierarchal Composed Workspaces, Internationsl Conference Automatics and Informatics'10, Sofia, 2010, pp. l-269-272

[G291] Jotov, Vl., An Investigation on the Approaches for Version Control Systems, ComSysTech’08, Gabrovo, 2008

[G334] Jotov, Vl., Towards a model of versioning domain, 5th CEE-SECR, 2009, Moscow, (под печат)

[G313] Jotov, Vl. Transaction over Versioned Objects in Hierarchical Workspace Environment, ECAI09, Pitesti, 2009

[G279] Katz, R. H. 1982. A database approach for managing VLSI design data. In Proceedings of the 19th Conference on Design Automation Annual ACM IEEE Design Automation Conference. IEEE Press, Piscataway, NJ, 274-282

[G32] Lee, B. G., Chang, K. H., and Narayanan, N. H. 1998. An integrated approach to version control management in computer supported collaborative writing. In Proceedings of the 36th Annual Southeast Regional Conference ACM-SE 36. ACM Press, New York, NY, 34-43. DOI= http://doi.acm.org/10.1145/275295.275302

[G370] Katz, Max, Practical RichFaces, ISBN-13 (paperback): 978-1-4302-1055-9, 2008, USA

[G356] Klimmer M.,The Mega Project Mandate, Transforming Government, 2008, pp. 25-32.

[G342] Krahn, R., Ingalls, D., Hirschfeld, R., Lincke, J., and Palacz, K. 2009. Lively Wiki a development environment for creating and sharing active web content. In Proceedings of the 5th international Symposium on Wikis and Open Collaboration (Orlando, Florida, October 25 - 27, 2009). WikiSym '09. ACM, New York, NY, 1-10. DOI= http://doi.acm.org/10.1145/1641309.1641324

[G149] Leblang, D., The CM challenge: Configuration management that works. Configuration Management, 2, 1994

[G323] Marcus, A. and Maletic, J. I. 2003. Recovering documentation-to-source-code traceability links using latent semantic indexing. In Proceedings of the 25th international Conference on Software Engineering (Portland, Oregon, May 03 - 10, 2003). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 125-135.

[L5] Microsoft Sharepoint, http://sharepoint.microsoft.com/ (accesed 20.06.2010)

[G343] Morris, J. C. 2007. DistriWiki:: a distributed peer-to-peer wiki network. In Proceedings of the 2007 international Symposium on Wikis (Montreal, Quebec, Canada, October 21 - 25, 2007). WikiSym '07. ACM, New York, NY, 69-74. DOI= http://doi.acm.org/10.1145/1296951.1296959

[G54] Morse, T., 1996, CVS, Linux J. 1996, 21es (Jan. 1996), 3.

[G383] Nakov, Sv., 05. Source Control Systems, 2011, http://kaltura.ludost.net/p/101/sp/10100/download/entry\_id/0\_vqc9ye7z/relocate/05.%20Source%20Control%20Systems%20(Source).mp4 (accessed 04.01.2012), 3:26

[G141] Nguyen, T. N., Munson, E. V., Boyland, J. T., and Thao, C. 2005. An infrastructure for development of object-oriented, multi-level configuration management services. In Proceedings of the 27th international Conference on Software Engineering (St. Louis, MO, USA, May 15 - 21, 2005). ICSE '05. ACM, New York, NY, 215-224. DOI= http://doi.acm.org/10.1145/1062455.1062504

[G139] Nguyen, T. N., Munson, E. V., and Thao, C. 2004. Fine-grained, structured configuration management for web projects. In Proceedings of the 13th international Conference on World Wide Web (New York, NY, USA, May 17 - 20, 2004). WWW '04. ACM, New York, NY, 433-442. DOI= http://doi.acm.org/10.1145/988672.988731

[G261] Nguyen, T. N., Munson, E. V., and Boyland, J. T. 2004. Object-oriented, structural software configuration management. In Companion To the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (Vancouver, BC, CANADA, October 24 - 28, 2004). OOPSLA '04. ACM, New York, NY, 35-36. DOI= http://doi.acm.org/10.1145/1028664.1028684

[G374] Nitin, K. L., Ananya S., Mahalakshmi K., and S. Sangeetha, iBATIS, Hibernate, and JPA: Which is right for you?,JavaWorld.com, 07/15/08

[G365] O’Donovan, Brian, RCS Handbook, 1992

[G366] Official RCS Homepage, http://www.cs.purdue.edu/homes/ trinkle/RCS/

[G252] Perforce. http://www.perforce.com/.

[G285] Posner, J., Block, J., CASEVision™/ClearCase Concepts Guide, 1994, *http://techpubs.sgi.com/library/dynaweb\_docs/0620/SGI\_Developer/* books/ClrC\_CG/sgi\_html/index.html

[G100/стр. 746] Pressman, R. S., Software Engineering: A Practitioner's Approach, 2005, McGraw-Hill Professional, 880 pages, page 746, ISBN 0072853182

[G50] Price, Derek R., CVS—concurrent versions system v1.11.22, http://ximbiot.com/cvs/manual/ cvs-1.11.22/cvs.html, 2006 (accessed 26.04.2009)

[G246] PVCS http://www.merant.com/

[L2] Crockfordm D., RFC 4627, The application/json Media Type for JavaScript Object Notation (JSON) , http://tools.ietf.org/html/rfc4627

[G277] Rochkind, M. J., The Source Code Control System. In IEEE Transactions on Software Engineering SE-1:4 (Dec. 1975), pages 364–370.

[G332] Ruparelia, N. B. 2010. The history of version control. SIGSOFT Softw. Eng. Notes 35, 1 (Jan. 2010), 5-9. DOI= http://doi.acm.org/10.1145/1668862.1668876

[G42] Slein, J. A., Vitali, F., Whitehead, E. J., and Durand, D. G. 1997. Requirements for distributed authoring and versioning on the World Wide Web. StandardView 5, 1 (Mar. 1997), 17-24. DOI= http://doi.acm.org/10.1145/253452.253474

[G170] Silva, M., Gedye, D., Katz, R., and Newton, R. 1989. Protection and versioning for OCT. In Proceedings of the 26th ACM/IEEE Conference on Design Automation (Las Vegas, Nevada, United States, June 25 - 28, 1989). DAC '89. ACM, New York, NY, 264-269. DOI= http://doi.acm.org/10.1145/74382.74427

[G247] StarTeam, http://www.borland.com/us/products/starteam/

[G284] Sun WorkShop TeamWare User's Guide, http://docs.sun.com/source/806-3573/TeamWareTOC.html

[G244] Telelogic, http://www.telelogic.com/products/synergy/

[L1] The 1998 ACM Computing Classification System, http://www.acm.org/about/class/1998

[G148] Tichy, W. F. 1985. RCS—a system for version control. Softw. Pract. Exper. 15, 7 (Jul. 1985), 637-654. DOI= http://dx.doi.org/10.1002/spe.4380150703

[G372] Jain, Pallavi et al. , J2EE Professional Projects, 2002, 888 pages, ISBN:9781931841221,

[L4] Joomla!, http://www.joomla.org/

[G291] Jotov, V. 2008. An investigation on the approaches for version control systems. CompSysTech '08, vol. 374. ACM, New York, NY, V.11-1. DOI= http://doi.acm.org/10.1145/1500879.1500959

[G361] Koegel, M., Helming, J., and Seyboth, S. 2009. Operation-based conflict detection and resolution. In Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models (May 17 - 17, 2009). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 43-48. DOI= http://dx.doi.org/10.1109/CVSM.2009.5071721

[G371] Linwood, Jeff, Dave Minter ,Pro Hibernate 3, 2008, ISBN:9781590595114, 264 pages

[G378] Terrence Ryan, MSSql vs MySQL vs Oracle, Stored Procedures, and Code Generation, 2007, http://www.numtopia.com/terry/blog/archives/2007/11/ mssql\_vs\_mysql\_vs\_oracle\_stored\_procedures\_and\_cod.cfm (12.12.2011)

[G383] Troelsen, Andrew, Pro C# 2010 and the .NET 4 Platform, Fourth Edition, Apress, 2010, (1753 pages), ISBN:9781430225492

[G248] *True Change,* http://www.truesoft.com/

[G376] Turvey, Steve, Duelling databases: Four apps tested, ZDNet.com.au , 2005,December 23rd, http://www.zdnet.com.au/duelling-databases-four-apps-tested-139226455.htm

[L6] WebDAV Resources, http://www.webdav.org/

[G245] WebIntegrity. http://www.mks.com/

[G72] Weiss, C., Premraj, R., Zimmermann, T., and Zeller, A. 2007. How Long Will It Take to Fix This Bug?. In Proceedings of the Fourth international Workshop on Mining Software Repositories (May 20 - 26, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 1.

[G109] Westfechtel, B. 1991. Structure-oriented merging of revisions of software documents. In Proceedings of the 3rd international Workshop on Software Configuration Management (Trondheim, Norway, June 12 - 14, 1991). P. H. Feiler, Ed. ACM, New York, NY, 68-79. DOI= http://doi.acm.org/10.1145/111062.111071

[G98] Whitehead, E. J. 2001. Design spaces for link and structure versioning. In Proceedings of the Twelfth ACM Conference on Hypertext and Hypermedia (Århus, none, Denmark, August 14 - 18, 2001). HYPERTEXT '01. ACM Press, New York, NY, 195-204. DOI= http://doi.acm.org/10.1145/504216.504265

[G339] Wiegers, Karl E., Software Requirements, Second Edition, Microsoft Press, 2003, 516 pages

[G235] Wikipedia contributors, "AllFusion Harvest Change Manager," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title= AllFusion\_Harvest\_Change\_Manager&oldid=193222994 (accessed February 14, 2009)

[G375] Wikipedia contributors, "Comparison of web application frameworks," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title= Comparison\_of\_web\_application\_frameworks&oldid=465118984 (accessed December 12, 2011).

[G380] Wikipedia contributors, "Comparison of relational database management systems", http://en.wikipedia.org/w/index.php?title= Comparison\_of\_relational\_database\_management\_systems&oldid=465424149, (accessed December 12, 2011)

[G349] Wikipedia contributors, “Document management system”, Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php? title=Document\_management\_system&oldid=358652743 (accessed April 2010)

[L7] Wikipedia contributors, "Hypermedia," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title=Hypermedia&oldid= 389812815 (accessed November 13, 2010).

[G379] Wikipedia contributors, "Java Server Daces", http://en.wikipwdia.org/ w/index.php?title=JavaServer\_Faces&oldid=457471575, (accessed December 12, 2011)

[G48] Wikipedia contributors, "List of revision control software," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php? title=List\_of\_revision\_control\_software&oldid=270041448 (достъпена 14 Февруари, 2009).

[G251] Wikipedia contributors, "Microsoft Visual SourceSafe," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php? title=Microsoft\_Visual\_SourceSafe&oldid=266174882 (accessed February 15, 2009).

[L3] Wikipedia contributors, "SOAP," Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title=SOAP&oldid= 360564396 (accessed May 8, 2010).

[G348] Wu, Q., Pu, C., and Irani, D. 2009. Cosmos: a Wiki data management system. In Proceedings of the 5th international Symposium on Wikis and Open Collaboration (Orlando, Florida, October 25 - 27, 2009). WikiSym '09. ACM, New York, NY, 1-2. DOI= http://doi.acm.org/10.1145/1641309.1641343

[G153] Zeller, A. and Snelting, G. 1997. Unified versioning through feature logic. ACM Trans. Softw. Eng. Methodol. 6, 4 (Oct. 1997), 398-441. DOI= http://doi.acm.org/10.1145/261640.261654

[L5] Андрейчин, Л., Л. Георгиев, Ст. Илчев, Н. Костов, Ив. Леков, Ст. Стойков, Цв. Тодоров, Д. Попов, Български тълковен речник, Наука и изкуство, София, 2008

[G319] Белладжио Дэвид, Том Миллиган, Разработка програмного обеспечения: управление изменениями, ДМК Пресс, 2009, 384 стр.

[G298] Екел Брус, Да мислим на JAVA, том 1, София, СофтПрес, 2001, 591 стр.

[G83] Коуберн Ал. Каждому проекту своя методология, 2005, http://www.citforum.ru/SE/project/meth\_per\_project/ (accessed on 16.03.2011)

[G367] Наков, Светлин, и колектив, Въведение в програмирането с Java, София, 2008, ISBN 978-954-400-055-4

[G369] Наков, Светлин, и колектив, Въведение в програмирането със C#, София, 2011, ISBN 978-954-400-527-6

[G368] Наков, Светлин, Интернет програмиране с Java, София, 2004, ISBN 954-775-305-3

[G206] Снедакер, С., Управление IT-проектом, или Как стать полноценным CIO, превод от англ., Москва, 2009, 616 стр.

[G294] Фейсон Тед, Borland C++ Обектно-ориентирано програмиране - Част І, София, Нисофт, 1994, 400 стр.