ВЕЛИКОТЪРНОВСКИ УНИВЕРСИТЕТ
“СВ. СВ. КИРИЛ И МЕТОДИЙ”

Факултет “Математика и информатика”
Катедра „Компютърни системи и технологии”

Владимир Йорданов Йотов

МОДЕЛИ, БАЗИРАНИ НА ЙЕРАРХИЧНИ КОМПОЗИЦИИ ОТ ПРОСТРАНСТВА, ЗА УПРАВЛЕНИЕ НА СОФТУЕРНИ ВЕРСИИ

ДИСЕРТАЦИЯ

за присъждане на образователна и научна степен „**Доктор**”
Професионално направление: 4.6 “Информатика и компютърни науки”

Научен ръководител: **доц. д-р Христо Тужаров**

Велико Търново
2013

Съдържание

Увод

Управлението на версията на софтуерните продукти заема важно място в областта на софтуерното инженерство [22, 71, 74 – стр. 746, 90]. Въпреки наличието на разработени модели, научно-приложната област предоставя възможности за търсене на решения за постигане на по-висока ефективност на работния процес. Модерните гъвкави методологии предлагат един по-свободен начин на развитие на софтуерните продукти. Те предполагат използването на специалисти от много високо ниво, които познават разработвания продукт в детайли. Създаването и задържането на такива специалисти представлява предизвикателство пред ръководството на всяка една компания. Настоящата дисертация може да се разглежда като опит да се предостави възможност за снижаване на рисковете от използването на такива кадри и въвеждането на инструментариум за автоматизация при използването на гъвкавите методологии.

Обект на изследване в дисертацията са моделите и методите в управлението на версии чрез използването на йерархично композирани работни пространства за постигане на по-ефективен подход на нейното управление, ускоряване анализа на влиянието на промените над системата, усъвършенстване политиката на управление на знания в компаниите и инструмент за обсъждане финансовите аспекти на проектите.

Методологията на изследването включва следните подходи: евристичен анализ на предизвикателствата, стоящи пред съществуващите модели в научно-приложната област; сравнителен анализ на използваните модели и методи и определяне на нови идеи; търсене, изследване и ефективно развитие на модели и методи за управление на версия и повишаване ефективността на процеса на създаване и поддържане на софтуерните продукти.

Дисертацията се състои от увод, три глави, заключение, използвана литература, едно приложение и прототип.

В Първа глава е направен обзор на моделите в областта на управлението на версиите. Разгледани са мястото, целите и задачите на управлението на версии в рамките на разработването и поддържането на софтуерни продукти. Направен е обзор на съществуващите модели на версионизираните обекти и на начина на тяхното съхраняване в репозиторито с версии. Отделно е направен обзор на темата за съвместната работа на сътрудниците, където е наблегнато на работните пространства като средство за осъществяването на кооперираност. За постигане на пълнота в обзора са сравнени методите за проследимост на промените. Главата завършва с определяне на изводите, формиране на целта и задачите на дисертацията, които следва да бъдат решени във втора и трета глава.

Във Втора глава са представени теоретичните модели за управление на версия в среда с йерархично композирани работни пространства. Моделите са допълнени с авторска методологична рамка за тяхното ефективно използване. Във формулираните в края на главата изводи са посочени предимствата на разработените модели.

Трета глава съдържа аналитично обоснован избор на средства за реализиране на програмен прототип на система, реализираща теоретичните модели. Представени са описания на авторска алгоритмична реализация на по-важните моменти от прототипа. В главата е направена теоретично-експериментална сравнителна симулация на разработка на програмен продукт с и без използване на разработения прототип при гъвкава методология за разработване. Направените изводи в края на главата разкриват перспективите от използването на разработените модели.

В заключението е направено обобщение получените резултати. Формулирани са основните резултати в рамката на дисертацията. Посочени са някои актуални задачи, които могат да бъдат естествено продължение на настоящото изследване.

Разработката и апробацията на резултатите са извършени самостоятелно, като регулярно са представяни в катедра „Компютърни технологии” на Великотърновски университет „Св. св. Кирил и Методий”.

Получените резултати са публикувани в пет доклада, два от които в чуждестранни научни конференции.

Целта на настоящата дисертация може да се определи като: Изследване и създаване на модели за управление на софтуерни версии версии в среда, базирана на йерархично композирани работни пространства, които да послужат за създаването на прототип на система за управление на версии.

За успешното реализиране на така формулираната цел са поставени следните задачи:

1. Да се създаде модел на версионизиран обект, осигуряващ максимална гъвкавост при определяне степента на гранулираност на данните в съчетание с простота и универсалност.
2. Да се създаде модел на среда с йерархично композирани работни пространства, както и да се определят правилата за управление на версия на обекти в тази среда.
3. Да се адаптира метод за проследимост на промени, базиран на събития, за среда с модел на йерархично композирани работни пространства.
4. Да се определи терминологията в областта на версионизирането с използването на йерархично композирани работни пространства.
5. Да се създаде методологична рамка за създаване на софтуерни продукти в среда с йерархично композирани работни пространства.

6. Да се увеличи степента на автоматизация на дейностите при създаване на софтуерни продукти, в следствие на използване на разработените модели.

Глава първа
Управление на версиите при създаването на софтуерни системи

Глава има за цел да представи различни аспекти в областта на управлението на версиите, които пряко или косвено са обект на настоящото научно-приложно изследване. Управлението на версиите заема важно място в управлението на жизнения цикъл на софтуерните продукти (SDLC - software development life-cycle).

Първия параграф на главата разглежда мястото, целите и задачите на управлението на версии в рамките на разработването на софтуер. В следващият параграф ще бъдат представени различните модели за управление и контрол на версиите. Моделите на версионизиран обект представляват важен момент, като на техния обзор и анализ е посветен третият параграф на главата. Четвъртият параграф е посветен на подходите за съхраняване на версии. Петият параграф разглежда темата за съвместната работа и работните пространства като основен елемент за осъществяването й. Методите за проследимост на промените заемат основно място в осигуряването на отчетност на работата, а също така и във фазата на поддържане на продукта. Именно затова на тях е посветен шестият параграф от главата.

Главата завършва с резюмиране на изводите. Тези изводи показват актуалните предизвикателства в областта на управлението на версиите. Това мотивира определянето целта и задачите на дисертацията.

1.1. Място на управлението на версии

1.1.1. Модели на създаване на софтуер

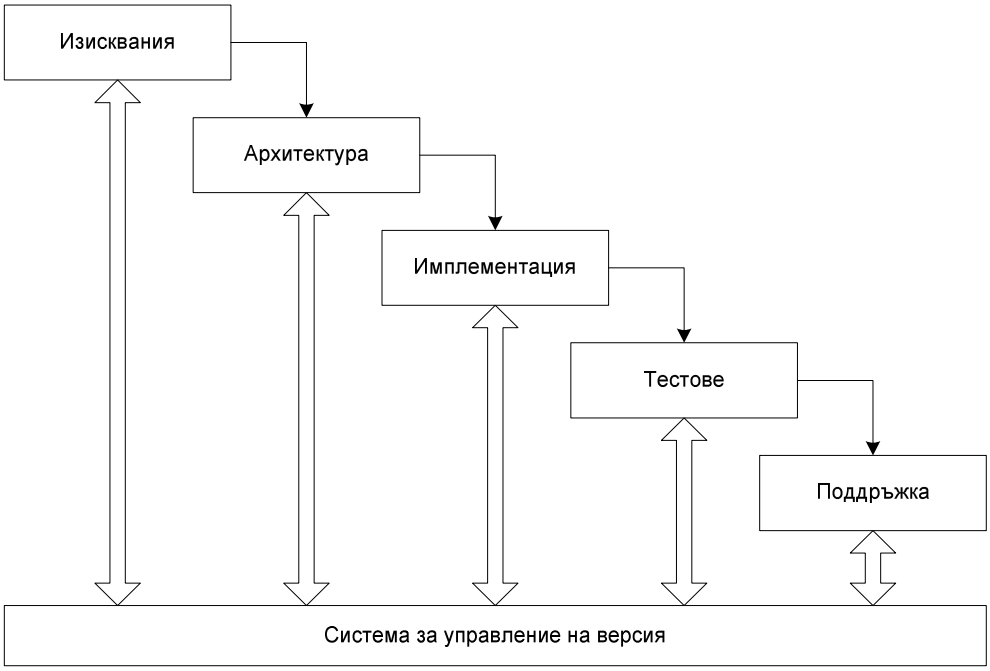
Създаването на софтуерни системи се определя като отрасъл на нематериалното производство с голяма степен на интелектуалоемкост.

Исторически по естествен начин е формулиран факта, че производителите на софтуерни продукти са основните потребители на системите за управление на 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 – процесът на тяхното модифициране с цел да удовлетворят промяната на първоначалните изисквания и/или появата на нови изисквания.

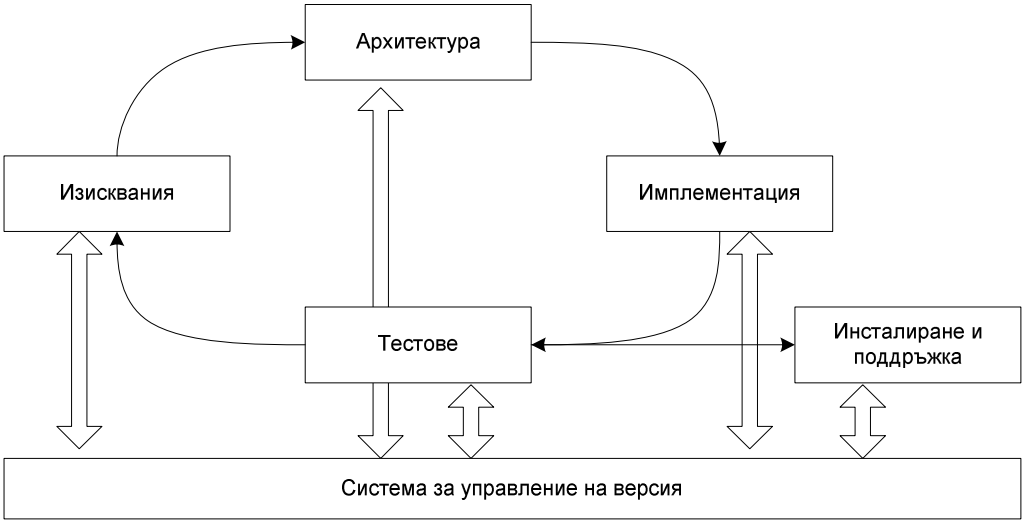
За създаването на софтуерни системи са разработени множество формални методологии – каскадна [79], спирална [21] (итеративна), в-образна, Модел на бърза разработка (Rapid Application Development - RAD) [83] и други [5]. Многообразието от методологии се обуславя от различните предимства, недостатъци и характеристики, които съдържа всяка една от тях. Допълнително може да се изтъкне, че някои автори в своите изследвания правят опит да определят съответствието между големината на проекта, степента на риск и целесъобразността от използването на една или друга методология [4]. Въпреки това основните фази, през които по някакъв начин преминава всеки един софтуерен продукт, могат да се сведат 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 съществуващата система.
- Създаване на версия на продукта, отговаряща на определен набор изисквания.
- Осъществяване на качествен контрол на системата за съответствие на 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 жизнения 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 останалите клиенти. Това води до необходимостта за управление на конфигурациите при доставянето до крайния 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 за 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 предвид ръчното стартиране от специалист на инсталационни, конфигурационни и актуализиращи скриптове от специалист.

От хронологическа 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 и тромавост [2]. Появата на SCCS (Source Code Control System) през 1972 г. води до автоматизиране и ускорение на процеса на версионизиране [80].

В рамките на предлаганата дисертация терминът **управление на конфигурация** следва да се разбира като синоним на термина **управление на версия** [67 – 03:26].

Управление на документи

Системите за управление на документи представляват отделен клон на развитие на механизмите за управление на версии. Те предоставят

класически механизми за версионизиране, използвани в системите за управление на изходния код на софтуерните системи. При тяхното управление се акцентира на управлението на метаатрибутите на документите, като се фокусира над следните аспекти [18, 106]:

- Управление на метаатрибути, 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 нужния документ в системата.
- Интеграция на документа с други системи, които могат да 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.
- Съхранение на документите: място, политики и др.
- 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 нужния 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, нужни при работа с документи със сложна структура.
- Поддръжка на работен процес при работа с електронен документооборот като например промяна и контрол на статусите на документа.
- Управление на паралелната и съвместната работа по документите.

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

- Уики системите [24, 56, 63], предназначени за изграждане на уеб базирано съдържание с цел постигане на кооперираност, управление на знание и др.
- Електронни системи за документооборот, базиран на правила. Като пример за система от тази група може да се посочи системата за докуметооборот на платформата Microsoft Sharepoint [62].
- 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! [50], Wordpress[109], Drupal[34].
- Хипертекстови документи. Отдалечената и паралелна работа по хипертекстови документи е повлияна 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 [99]:
  - Управление на метаданни, 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.

- Ограничаване работата над документите, 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.

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

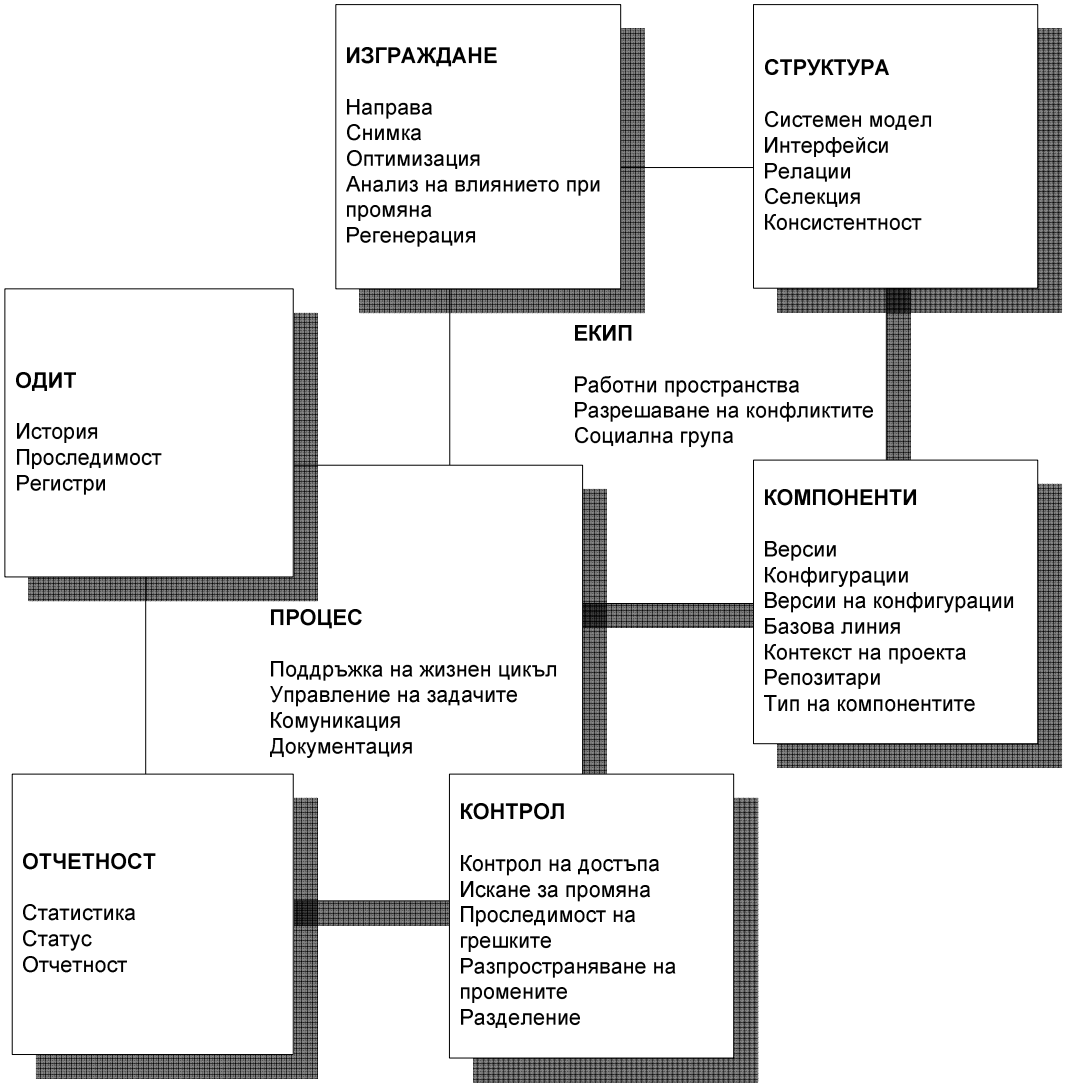
Целта на системите за управление и контрол на версията е да предостави инструмент на разработчиците на програмни продукти да автоматизират процесите на версионизиране. Много автори [23, 32, 35], базирайки се на стандарта IEEE 729-1983, определя функционалния обхват на системите за управление на конфигурации, който може да се обобщят в следните групи:

- Идентификационна схема, която отразява структурата на програмния продукт, неговите компоненти, както и техните типове.
- Контрол на изданията на продукта и 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.
- Производство, 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.

Допълнително се развива класификацията на задачите [23, 32, 35], които трябва да изпълняват системите за управление на версиите. В представения модел се разглеждат осем основни функционални направления, като две от тях се определят 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 прогреса 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 има и управлението на измененията, анализът 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 отделния участник в процеса.

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



Фиг. 3 Класификация на функционалните изисквания към система за управление на измененията[23, 32, 35]

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 и оптимизация. Промяната в първоначалните изисквания е често

срещано явление в практиката [53], затова системата за управление на версията трябва да предоставя функционалност, подпомагаща осъществяването на анализ на обхвата на влиянието от промяна в изискванията.

От гледна точка на структурата на крайния 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 крайния продукт.

От гледна точка на компонентите, изискването към версионизиращата система е да поддържа различни версии на различните компоненти на продукта. Тя трябва да подпомага конфигурацията на продукта за отвелните клиенти, които често имат специфични изисквания. Не на последно място, системата трябва да поддържа версии на клиентските 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, процеса и на дейностите, които са извършени 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.

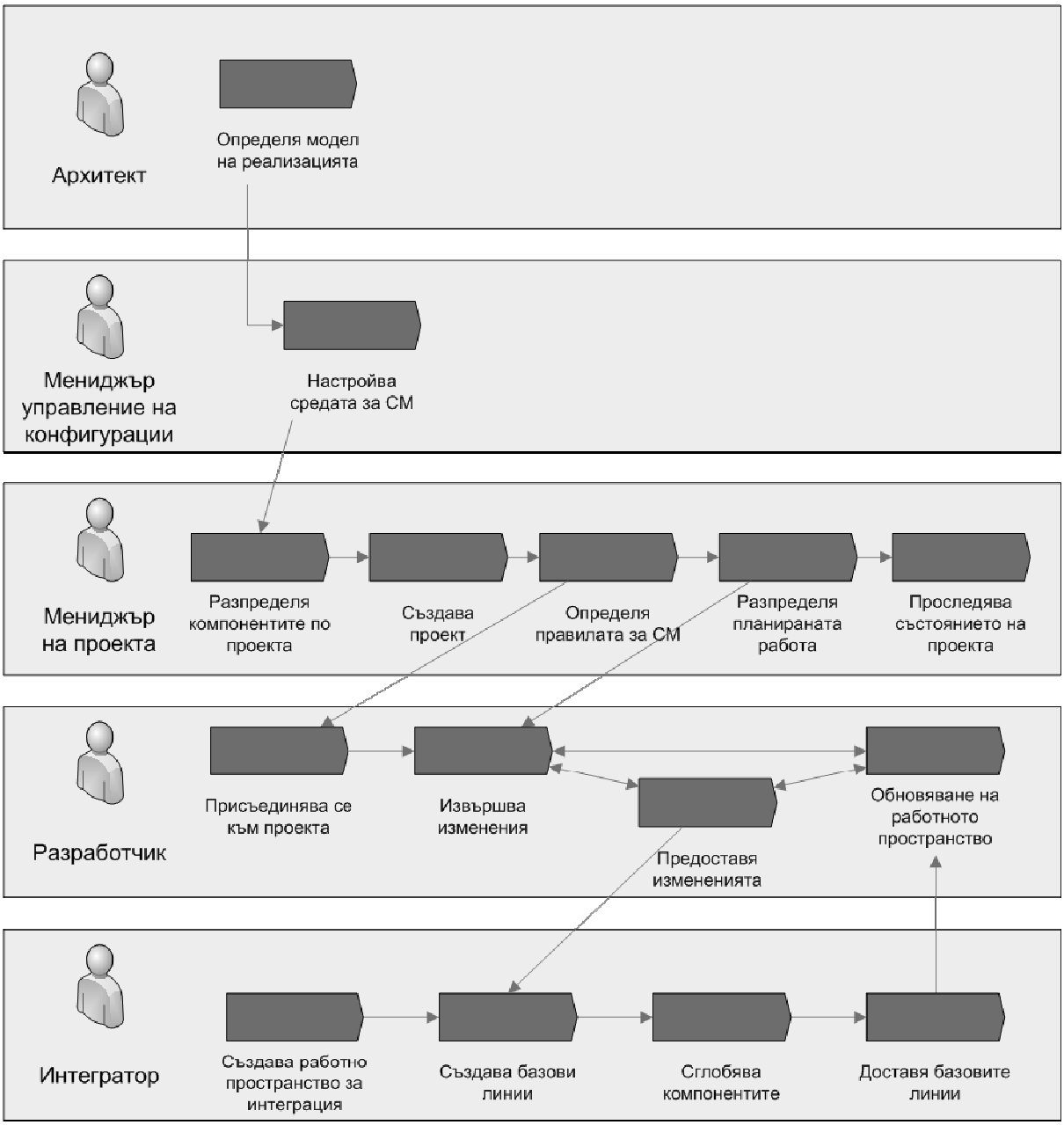
Акцентът при контрола е насочен към контрола на достъп, управление исканията за промяна, проследимостта на грешките.

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.

1.1.3. Управление на версиите при създаването на софтуер

В зората на компютърната ера управлението на версията на софтуерните продукти се реализира под формата на ръчен процес с участието на библиотекар. По-късно започват разработки към автоматизиране на този процес, като през 70-те години на XX век широко разпространение получават системите SCCS и RCS. В по-късен етап те се превръщат в основа за поредица допълнителни инструменти, които работят, използвайки двете системи, и предоставят допълнителен функционал на разработчиците на софтуер. Този процес не остава изолиран, паралелно се появяват други проекти и системи за управление на версии не само на софтуерни продукти, но и на CAD/CAM продукти, както и на документи.

В своята книга Беллаждио [2] представя процес за унифицирано управление на промените (Unified Change Management). Той определя следните участници в процеса на създаване на софтуерни системи: архитект, мениджър управление на конфигурации, мениджър на проект, разработчик, интегратор. На Фиг. 4 е представена схема на процеса за унифицирано управление на промените. Прави впечатление, че в представения модел липсва фазата на управление на качеството над крайния продукт, което е задължително при създаването на съвременните системи.



Фиг. 4 Унифицирано управление на промените [2]

1.2. Модели на управление и контрол на версиите

В настоящия параграф са представени основните модели, осигуряващи версионизирането на продуктите. Направен е анализ на моделите от по-ниско ниво на версионизиране с цел определяне потребностите и предизвикателствата към версионизиращите системи.

1.2.1. Модел на извличане / записване

Моделът на извличане и записване е широко използван в системите UNIX SCCS [78] и RCS [96], като от последната система произлизат най-широко използваните CVS/Subversion [28, 64, 75]. 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 [37] 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.

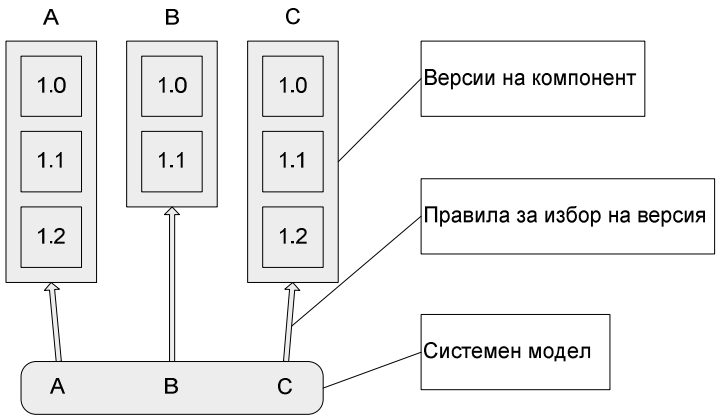
При системите за управление и контрол на версията, използващи подход на извличане/записване, потребителите и разработчиците, използват т. нар. работни области, представляващи различни файлови директории. Може да се отбележи, че поддръжката на потребителите в техните работни области е твърде ограничена. 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.

1.2.2. Композиционен модел

Композиционният подход представлява естествен еволюционен наследник на подхода извличане/записване. Този подход се базира на употребата на следните основни елементи [37]:

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

В модела 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 крайния продукт.



Фиг. 5 Избор на версия на компонент при конфигурация [37]

Феилер [37] определя два подхода за конструиране на правила за избор:

- Чрез търсене на път в граф на версии с етикети.
- Чрез предикати над метаданни на версионизираните обекти.

1.2.3. 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.

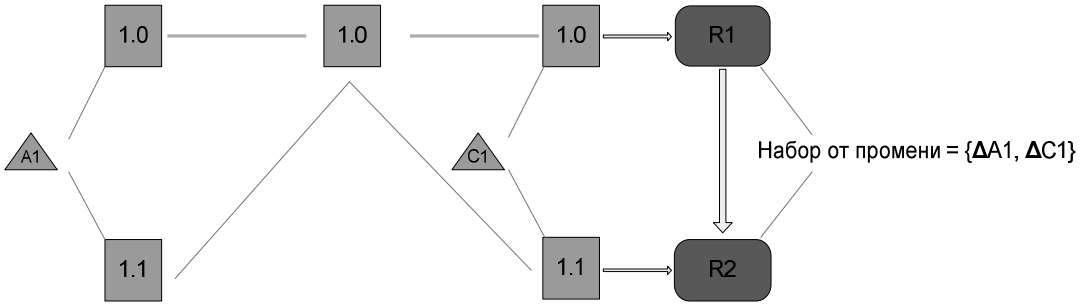
Това, което отличава дългите транзакции от стандартните транзакции е фактът, че не е определено кога тя ще приключи. Също така не е известно дали тя ще приключи изобщо. Като особеност на дългата транзакция може да изтъкне факта, че тя притежава свойството да запазва промените (persistence). В рамките 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 механизмът на вложени транзакции [37]. Пространството, в което се изпълнява дадена дълга транзакция, е известно и като работно пространство. Дългите транзакции и механизмът на влагане на транзакции могат да се разглеждат като предшественик на работните пространства.

1.2.4. 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 използва подхода на съхраняване на промените в репозитори, разгледан в пункт 1.2.1. Особеност на набора от промени е, че този подход се характеризира с намалени изисквания към използваната памет.

Друга характеристика на този подход е необходимостта да се следва определена последователност при прилагане на промените върху обектите, която е ориентирана на логическата промяна на компонентите [37].

Като пример от практическа гледна точка може да се изтъкнат поправките (patches), които често имат изискване да бъде инсталирана определена предишна (или набор от предишни) поправки върху определена версия на продукта.



Фиг. 6 Набор от промени при построяване на конфигурация [37]

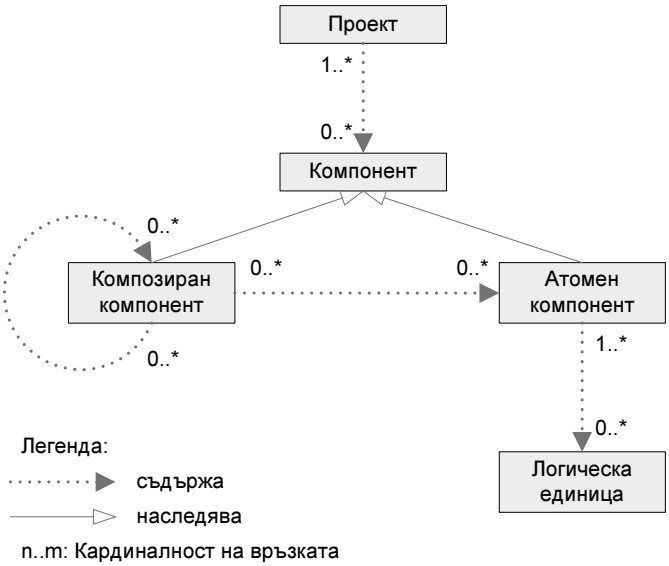
1.2.5. Версионизиране при обектно-ориентиран модел

В своето изследване Нгуйен [68] 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 в неговия модел са разделени на 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 своя модел авторите изтъкват неговата логическа и обектна-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 софтуерния продукт в съответствие с конкретна парадигма, а не над конкретното файлово представяне на данните.

Като недостатък на представения модел може да се отбележи наличието на два вида компоненти – композиран и атомен. Това го прави по-сложен и по-труден за реализиране. В частност се изисква реализиране на допълнителни 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 [68]

1.3. Модели на версионизиран обект

Параграфът има за цел да представи различните модели на версионизирани обект [102]. 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 [29, 74] 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.

1.3.1. Съхраняване като файлове

При файловия 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 [71, 72], CVS [64, 65], Subversion [28], ClearCase [2]. Предпоставка за това е лесното реализиране, приемливата скорост на работа и фактът, че потребителите на тези системи имат добри навици при работа с файлове. Въпреки това някои изследователи [54, 55, 68, 82] определят този модел като несъвършен, като се изтъкват следните недостатъци:

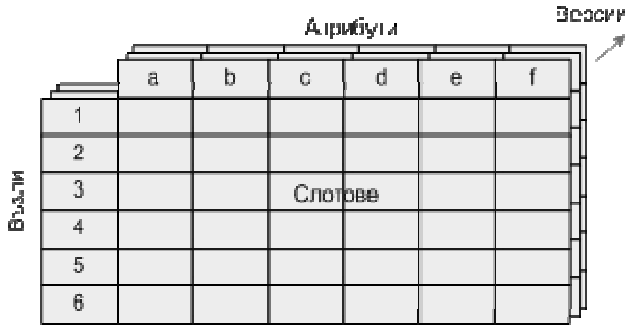
- Файлът като единица за съхраняване на данните е твърде обемиста и не гарантира, че данните са структуризирани.
- Съществува реална опасност от получаване на несъответствие между модели, логическа реализация и версионизиране на файлове [68], тъй като файлово ориентираният модел не предоставя механизъм 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 служебни файлове за описване на свързаността между основните файлове води до увеличаване ръчната работа и съответно на риска от получаване на грешки в крайния продукт.
- Файловото ниво на версионизиране не предоставят оптимален вариант при работа с графови модели, където се изисква версионизиране на ниво елемент на граф [44]. Това е особено важно при определяне на конфликтните точки при сравнение на две версии.

Системите, базирани на файловия модел, до голяма степен разчитат на възможностите, предоставени от файловата система при осигуряване на различните нива на достъп до обектите. Следва да се отбележи, че възможностите, които предоставят файловите системи, не винаги могат да

покрият конкретните нужди, стоящи пред една система за управление на промените.

1.3.2. Съхраняване при обектен модел на данните

В системата Molhado [68] е използван обектно-ориентиран подход за версионизиране на обектите. При него обектите се асоциират с 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. 8 Модел на данните за версионизиране на артефактите в системата Molhado [68]

Като слабости на разгледания модел може да се изтъкне неговата фиксираност от гледна точка структурата на съхраняваните обекти. Това се допълва от факта, че при промяна на самия модел на версионизираните обекти е нужно да се използват допълнителни инструменти за версионизиране на модела, т.е. не би било възможно прилагането на разработване, движено от модели (model driven development). При промяна на модела на версионизираните обекти възниква необходимост от използването на допълнителни инструменти, които да осигурят трансформацията на обектите от единия модел към другия и обратно. Последното съответно усложнява прилагането на механизми за проследяване на промените.

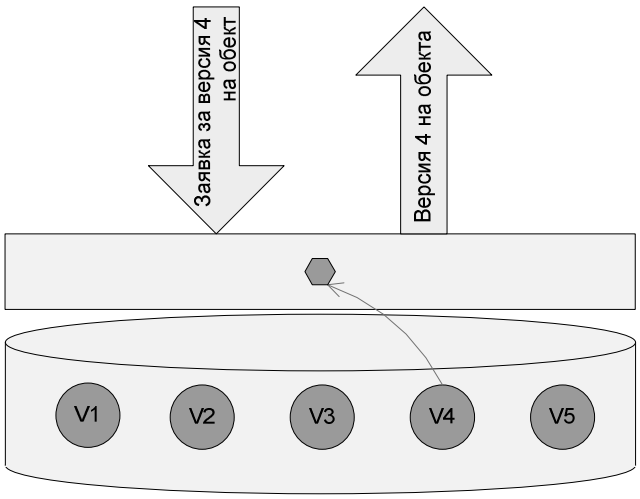
1.4. Подходи за съхраняване на версии

Съхраняването на версиите на обектите е важен момент в областта на управлението им и има голямо влияние над производителността на системата. Най-широко използваните подходи за съхраняване на версии са [66]:

- Съхраняване на състояние.
- Съхраняване на разлики (промени).
- Съхраняване на транзакции.

При подхода за съхраняване на „състояние" (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. Това е неразривно свързано с повишени изисквания към обема на използваната памет на хранилището. В този случай необходимият обем зависи не от големината, а от броя на направените промените. Тук следва да отбележим, че големината е пряко свързана със степента на гранулираността на артефакта. При извличане на съществуваща или при съхраняване на нова версия този подход ни дава константна сложност на алгоритмите.

При разглеждане на съхраняването на промени различните автори [29, 35, 36, 57, 76, 111] използват понятието делта между две версии, което отразява разликите между две версии на даден обект. В [2, стр. 60] са представини три вида реализации на подхода на съхравяване на промени: прав подход; обратен подход; вграден подход.



Фиг. 9 Пример за извличане на версия при подход за съхраняване на „състояния”

1.4.1. Прав подход за съхраняване на „промени”

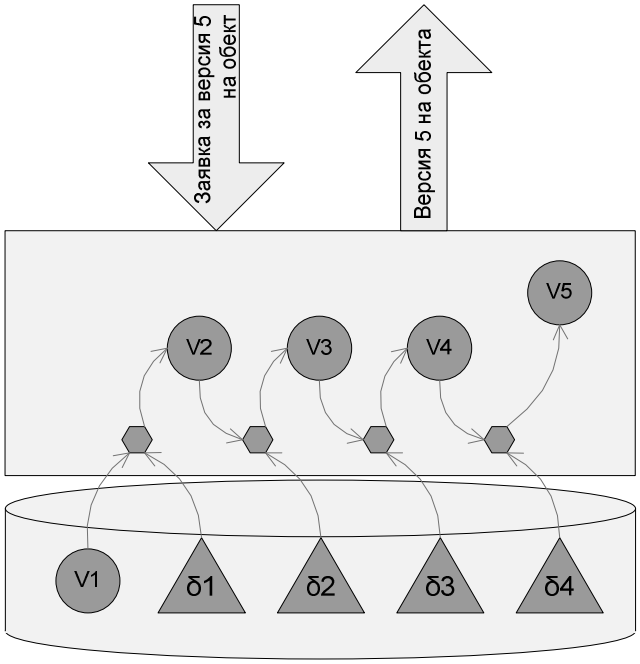
Подходът се базира на съхраняване на делтите в репозиторито. При заявка за извличане на определена версия на даден обект, системата следва да извлече началното състояние на обекта и всички делти, направени преди исканата версия. На базата на тези данни системата следва да изчисли заявената версия. В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

- Подходът е неустойчив поради факта, че версията се изчислява на база предишна версия и приложени 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. 10 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 номерът 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.

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 [110] 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".

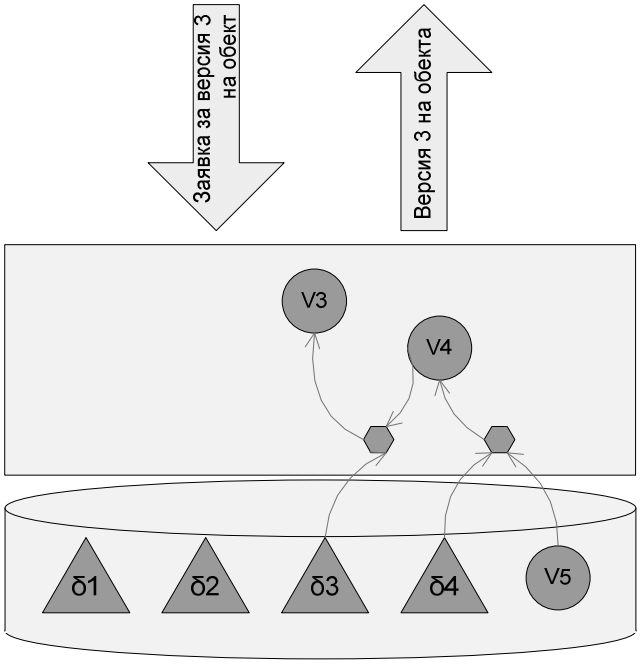


Фиг. 10 Пример за извличане на версия при прав подход за съхраняване на „промени"

1.4.2. Обратен подход за съхраняване на „промени”

Поради слабостите на правия подход е създаден обратен подход за съхраняване на „промени”. При него акцентът е в посока да се съхранява в монолитна форма само последната версия на обекта и на делти за разлики с предишни версии. 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. 11 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 [71, 72]. На по-късен етап с цел подобряване производителността на системата, употребата на подхода е сведена само до главната линия на разработка [2 – стр. 60].

1.4.3. Вграден (inline) подход за съхраняване на „промени”

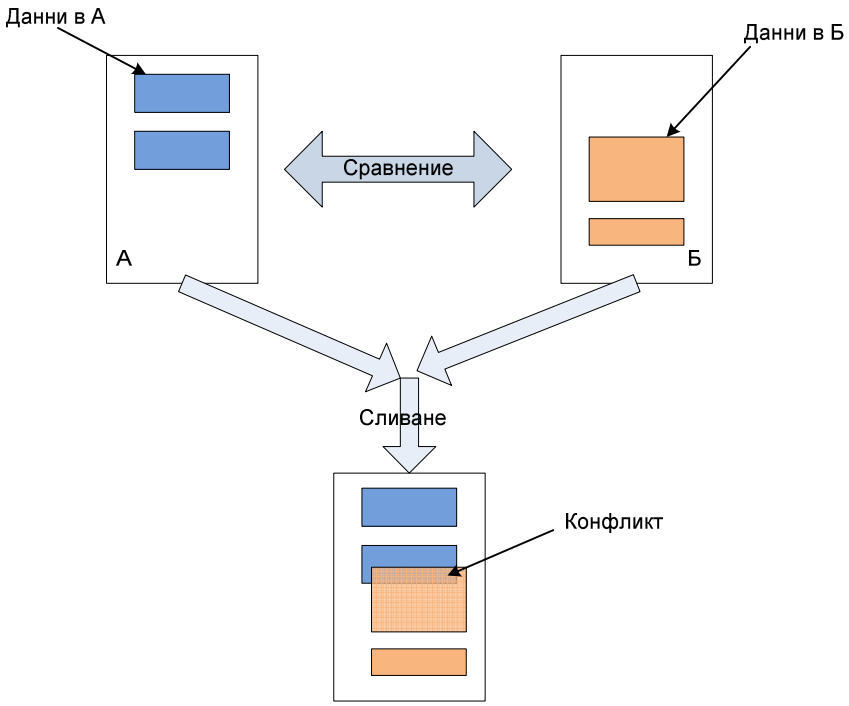
При вградения подход за съхраняване на „промени” не се съхраняват никакви състояния на обектите, а само делти във формата на

специална нотация, обезпечавайки възстановяване на произволна версия за постоянно време. Подходът е заложен в основата на системите CSSC и ClearCase. [2 – стр. 60]. За повишаване скоростта на работа на системата вграденият подход бива допълнен от механизми за кеширане на сглобените версии, които са най-често използвани.

1.4.4. Сливане на паралелни версии

В жизнения цикъл на софтуерните продукти е често срещано явление продуктът 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 на тяхната поддръжка. В следствие на процеса на тяхното рефакториране те биват консолидирани в общ компонент за системата. Именно при процеса 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, доколкото 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 предполага това да се извърши ръчно. Когато се говори за методи за сливане, е задължително да се спомена сливането с двойно сравнение (two-way merging) и сливането с тройно сравнение (three-way merging) [31].

1.4.4.1. Сливане с двойно сравнение

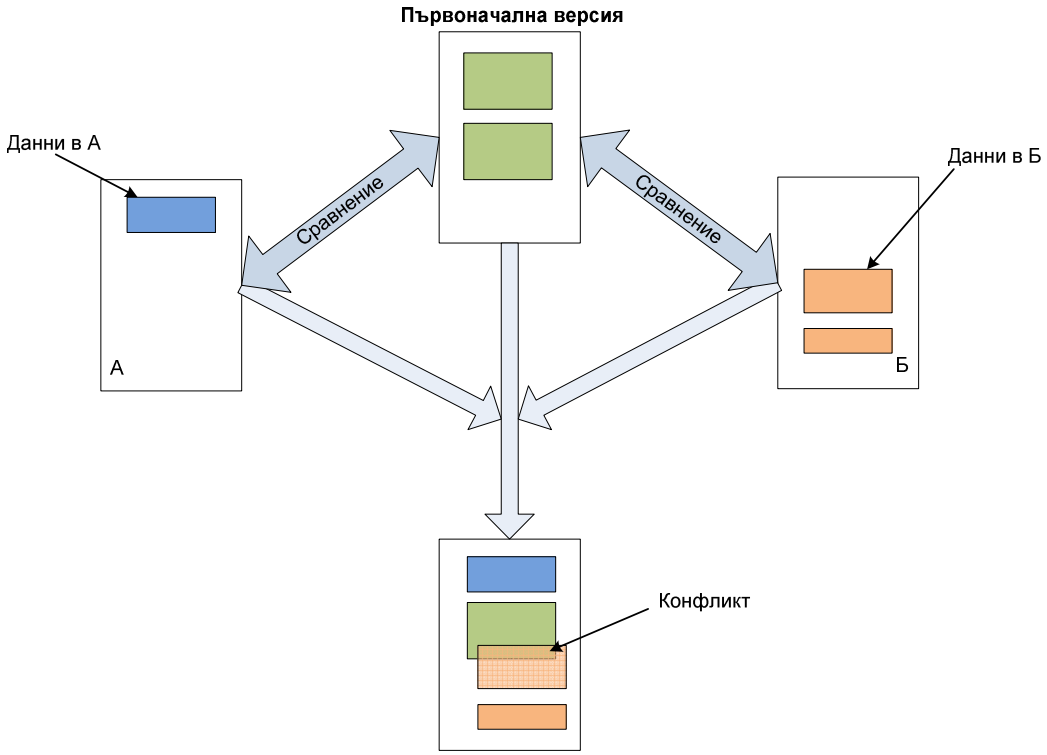


Фиг. 12 Сливане с двойно сравнение [31]

Сливането с двойно сравнение е по-просто. То се използва най-често при сливане на паралелна версия с основния клон на разработване. При него се използва метод „най-добро познаване” (best-guess) за изчисляване на резултатната версия. Този метод на сравнение се характеризира с ниска степен на качество на резултатната версия, т.е. в нея вероятността от наличие грешки поради неправилно сливане е много висока.

1.4.4.2. Сливане с тройно сравнение

Сливането с тройно сравнение при анализа на промените се използва обща първоначална версия за двете паралелни версии, които следват да бъдат слети. Получената резултатна версия е с по-високо качество в сравнение с резултатната версия от използването на сливането с двойно сравнение. Това, разбира се, става с цената на по-бавен алгоритъм [31].



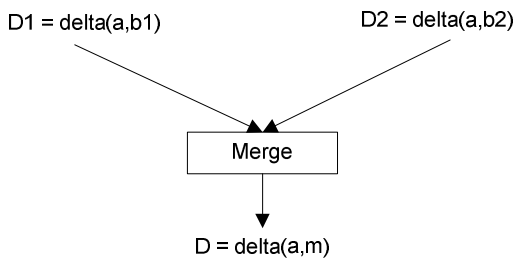
Фиг. 13 Сливане с тройно сравнение [31]

Представените методи на сливане се базират на сравнение на две версии, при сравняването на които се използват алгоритми за анализ. В практиката са разработени различни алгоритми за анализ и сравняване на версии, като те може обобщят в следните направления [57]:

- Сливане чрез на символен анализ на текст.
- Семантичнен анализ на текст (данни), чрез конструиране на граф на зависимостите за всеки един обект [44, 66].
- Сравнение историята на командите (транзакциите), променящи състоянието на обекта [66].

1.4.4.3. Синтактично сливане

Текстовото сливане [33, 96] 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. 14 Сливане на делти на две версии [101]

В своето изследване Вестфехтел [101] 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 първия или от втория входен артефакт.
- Изходната версия d, не трябва да съдържа дублираща се информация.
- Изходната версия d не трябва да съдържа конфликтни точки, т.е. елементи, които са премахнати в един от двата входни артефакта.

1.4.4.4. Семантично сливане на версии

Хорвиц и др. [44] представят своят HPR (Horwitz, Prins, and Reps) алгоритъм за анализ на изходен код на програми на езици, неподдържащи процедури (функции). Следва да се отбележи, че типът на алгоритамът може да се класифицира като такъв на сливане с тройно сравнение. В HPR алгоритъмът за направата на семантичен анализ на кода се използват представяния, подобни на графите на зависимостите в програмата (program dependence graph - PDG). Алгоритъмът е използван като основа за разработката на [20], където авторите представят нов алгоритъм за интегриране на версии на програми, написани на езици с възможност за извикване на процедури (функции). Авторите изтъкват следните

характеристики на представения семантично-базирания инструмент за сливане:

1. Използват се знания за програмния език, с цел да определи дали съществуват несъвместими моменти между двете версии на програмата.
2. Инструментът предоставя гаранции относно поведението на програмата след нейното сливане – на базовата версия и на нейните два варианта.

Предложеният алгоритъмът на Бинкли [20] включва следните стъпки:

1. Определяне на местата на промяна в програмата.
2. Определяне на общите моменти в трите версии на програмата.
3. Формиране на граф на сливанията.
4. Проверка за смущения, където се проверява за наличието на два типа смущения – дали изопачаване на промяната в следствие от прилагане на графа на сливанията; дали графът на сливанията съответства на някоя програма.

1.4.4.5. Транзакционно-ориентирано сливане

В своето изследване Шмидт и съавтори [82] представят модел за сливане на версии на елементи, базирано на история на командите. Авторите определят поредицата от команди на промяна, извършени по време на редактирането, като транзакция на промяната. Представеният алгоритъм е класифициран като сливане с тройно сравнение. Шмидт посочва, че характерна особеност на представения от него модел е, че за успешното сравнение на два елемента е необходимо да се проследи историята на промените до техен общ елемент. В [54, 55] се предлага подход за определяне на конфликтните точки между две версии, чрез анализ стъпките на промяна в транзакциите, променящи обектите.

1.4.5. Големина на промяната - съставност и гранулираност на версионизирани обекти

В този параграф се разглежда гранулираността и съставността на версионизираните обекти, както и практическите нужди, довели до тяхната поява. Под съставен или композиран обект ще се разбира такъв обект, който е съставен от други обекти, където композицията е способ за комбиниране на прости обекти, с цел да се представят по-сложни структури от данни.

1.4.5.1. Съставни обекти

Информационните системи и програмните езици исторически се развиват в посока за оптимизиране на работния 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 [37] се появява нуждата от определянето на композицията на обекти също като версионизиран обект. В [37] конфигурацията е определена като същност, състояща се от две части – системен модел и правила за избор на версията. Показаният по-горе на Фиг. 5 модел за избор на версия на компонент при конфигурация [37] демонстрира организирането на конфигурация.

1.5. Съвместна работа и работни пространства

В параграфа са разгледани в исторически план развитието на моделите на работните пространства. 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 [38] 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.

1.5.1. Модели за осигуряване на конкурентен / паралелен достъп

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

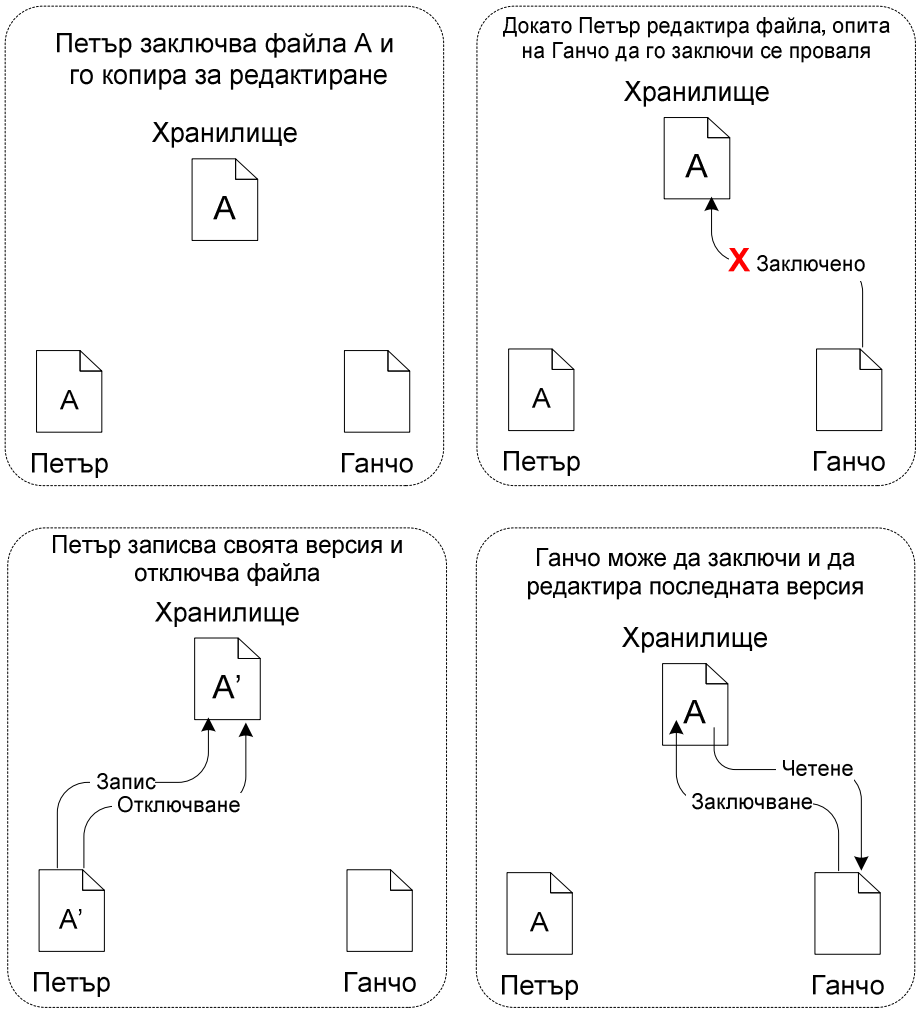
1.5.1.1. Модел за конкурентен достъп „Заключване – Модифициране – Отключване”

При файлово базираните хранилища, както беше споменато по-горе, нивото на гранулация на версионизираните обекти е сравнително голямо. При този тип репозиторита често се използва решението „Заключване – Модифициране – Отключване” [28].

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 ексклузивен 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 – края на същият файл.

- Заключването може да създаде фалшиво чувство за сигурност [28].



Фиг. 15 Подход "заключи, модифицирай, отключи"[28]

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 броя на заключените файлове, което от своя страна е предпоставка за понижаване ефективността на работа. Според [28] съществуват различни практики за решаване на тези недостатъци. Те обаче могат да се разглеждат по-скоро като заобикаляне същността на проблема, отколкото като цялостен подход за неговото разрешаване

1.5.1.2. Оптимистично и песимистично заключване при достъп до споделен ресурс

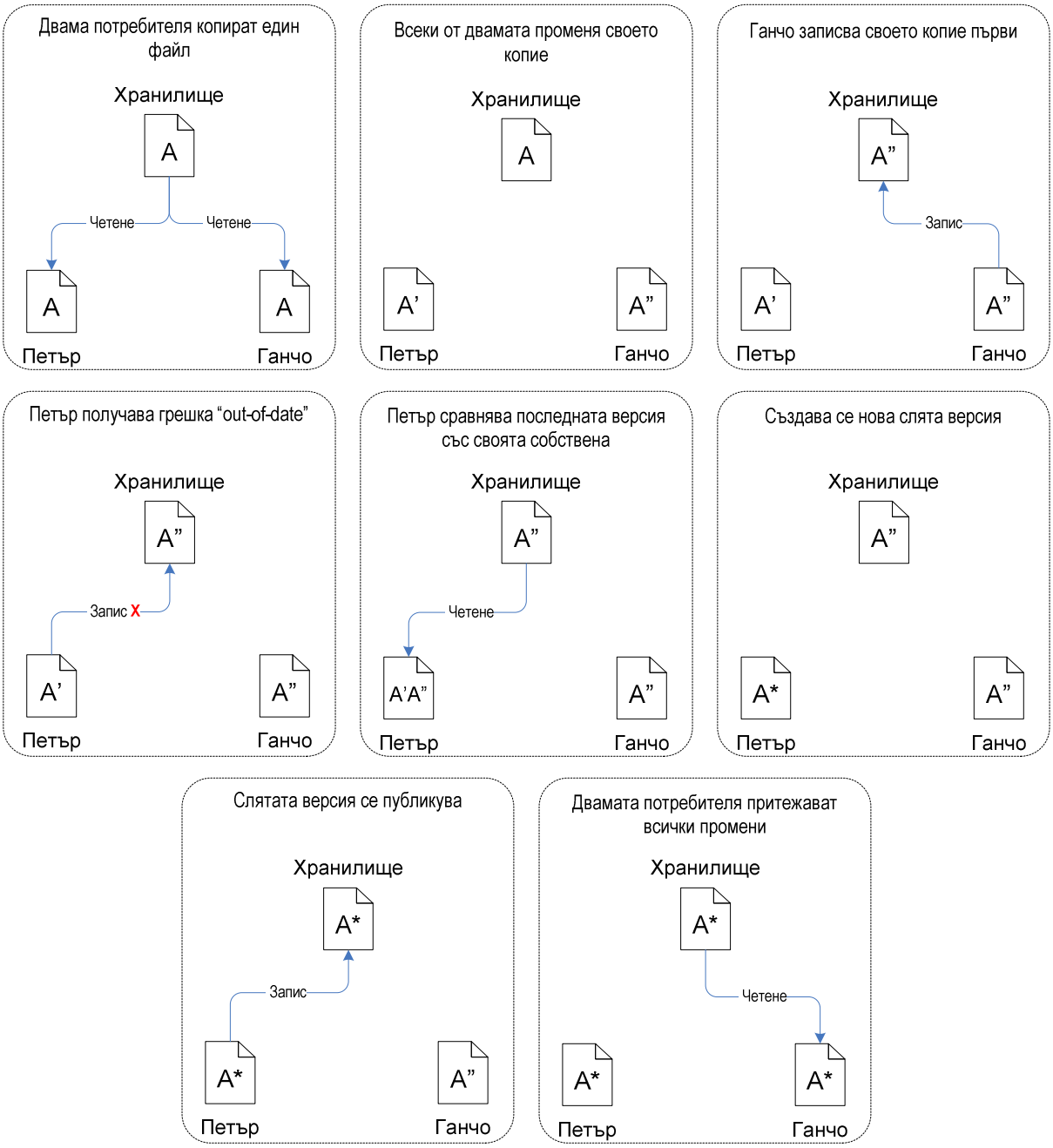
За осъществяването на конкурентен достъп до файловете се използва както оптимистичен, така и песимистичен подход за заключване на файловете.

При оптимистичното 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.

1.5.1.3. Модел за конкурентен достъп „Копиране – Модифициране – Сливане”

Еволюционно, като наследник на горе описания подход за версионизиране, може да се посочи подходът „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 1.4.4.

На Фиг. 16 графично е представен подходът „Копиране – модифициране – сливане” 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.



Фиг. 16 Подход "копиране - модифициране – сливане" [28]

1.5.2. 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 [35 – стр.406] авторите споменават, че **„... модерното работно пространство е създадено "зад гърба на науката", за да изпълни задача, определена от**

потребителя...”. Там 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 процеса на създаването на софтуерния продукт.

Общоприето разбиране за работното пространство [35, 75, 89] е, че то представлява файлова директория. От друга, страна Клемм и съавтори [27] го определят като инструмент, предоставящ механизми за кооперирание 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 може да се изтъкне този на извличане/записне [65]. Той често е допълнен от механизми за заключване на артефактите.

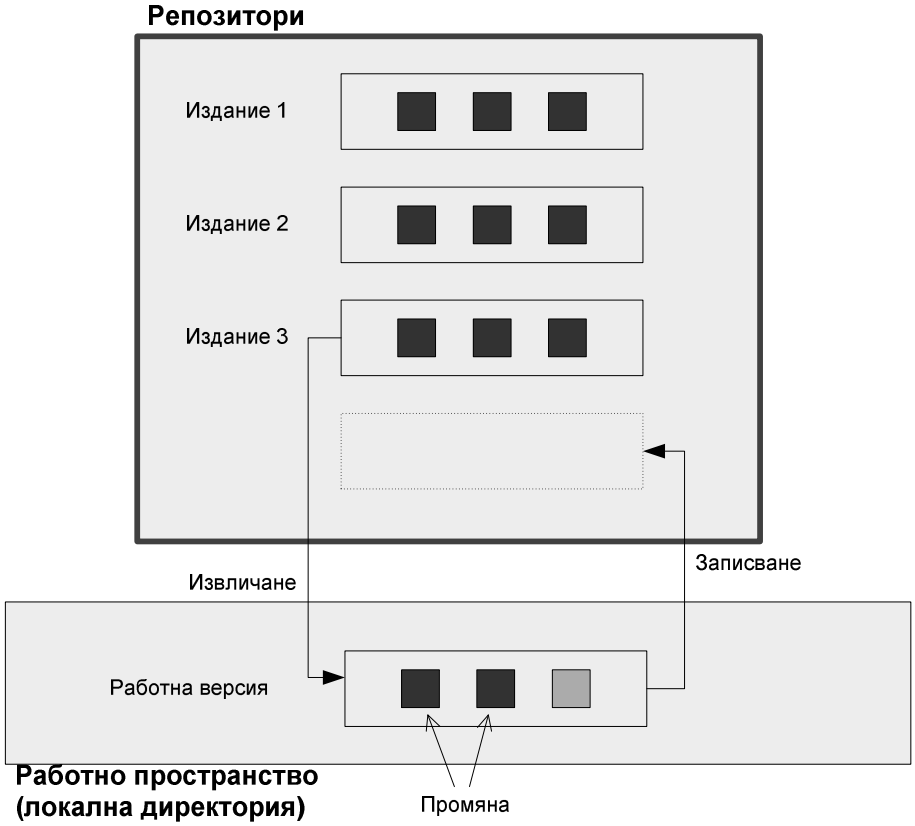
Като следствие от горното може да се определят следните основни направления на развитие при работните пространства:

1. Постигане на изолираност, с цел да се предотврати неволното или нежеланото взаимодействие при паралелна работа над компонентите на софтуерния продукт.
2. Място за кооперирание, където екипите постигат синхрон и интегрираност на крайния продукт.

Поснер [13] използва термина „изглед” (view) като синоним на работно пространство в системата IBM ClearCase. Въпреки това тези работни пространства са файлово базирани и те притежават недостатъците на системите, използващи файлове при управление на версии.

1.5.3. Файлово базирани работни пространства.

При файлово-ориентираните системи за управление на версията [28, 75] под работно пространство се разбира файловата директория, в която се извличат от репозиторито версиите на файловете. На Фиг. 17 е представен модел на работно пространство при файлово-ориентираните системи за управление на версията.



Фиг. 17 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:

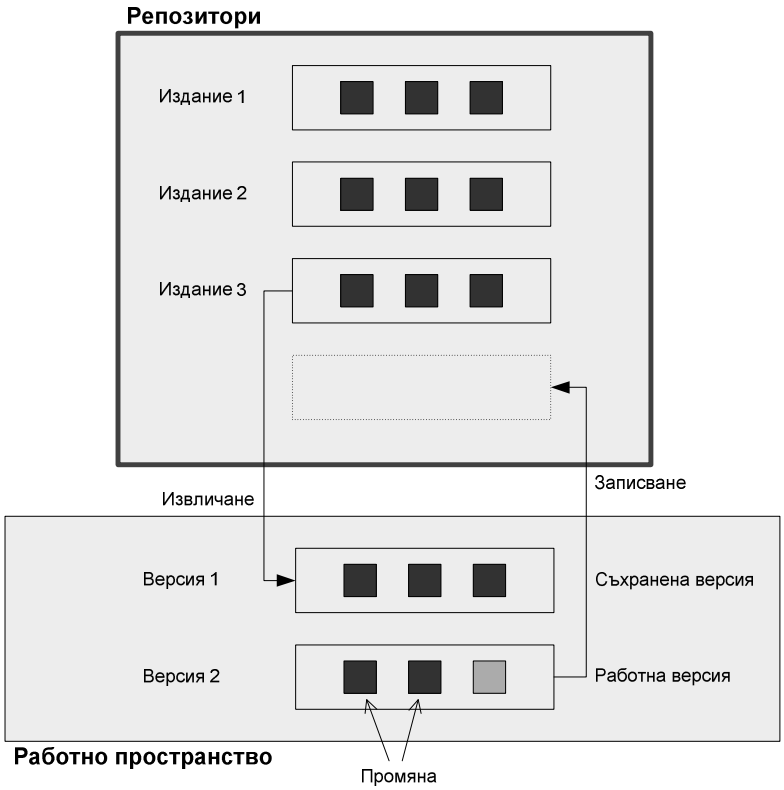
- Правото на достъп до обектите се гарантира от файловата система. Много малко са файловите системи, които са способни да предоставят нужното ниво на сигурност, което се изисква при работа с версионизирани обекти.

- 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 на 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 (човекочасове), и като следствие до недостатъчно ефективна работа на потребителите на системите за контрол на версия.

1.5.4. Модел на работни пространства със съхранена версия

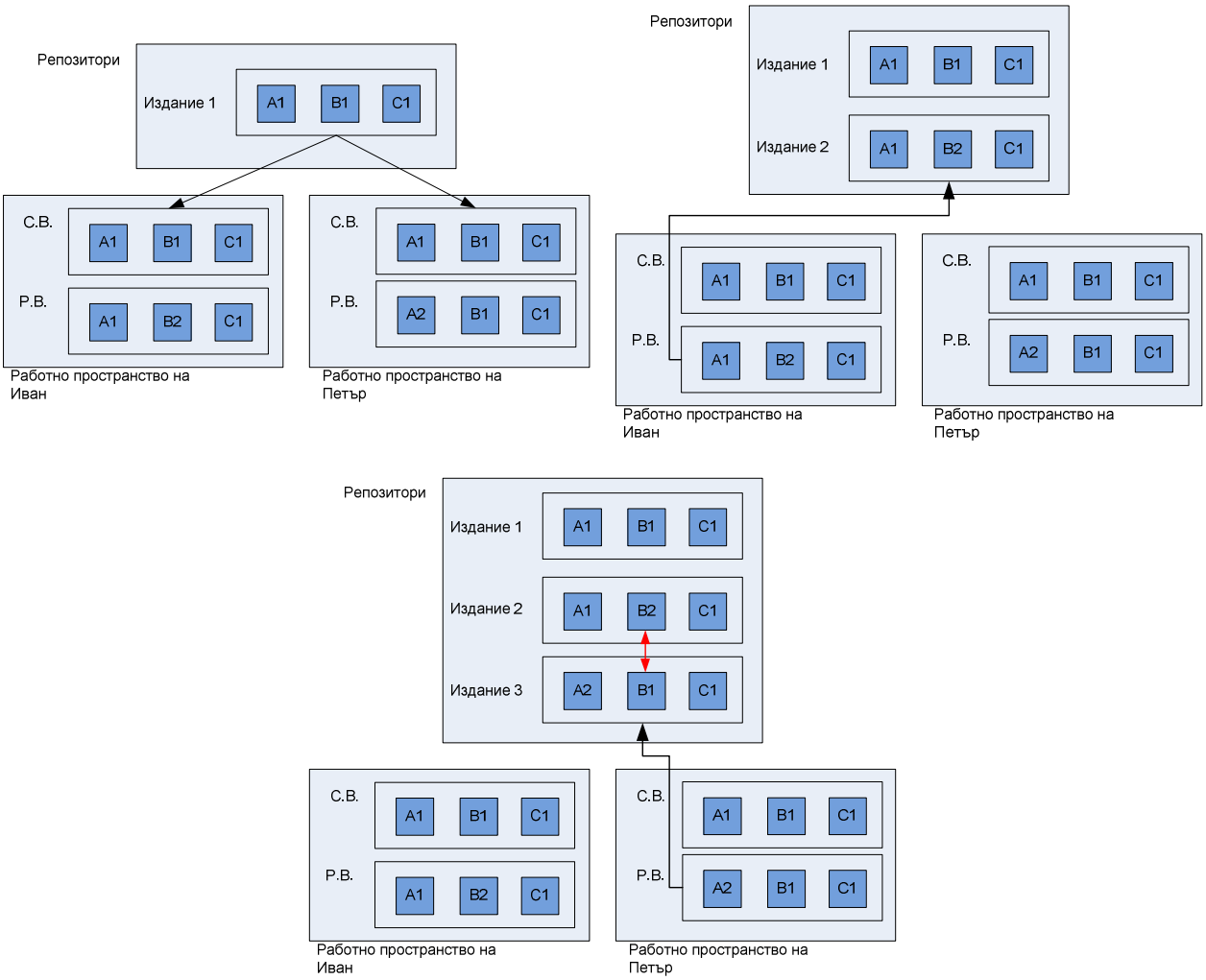
В своето изследване Фейлер [37] представя модел на работно пространство, който е съставен от два вида елемента: съхранена версия и работна версия - Фиг. 18. При извличане на версия от репозиторито, тя се представя в работното пространство като съхранена версия. Съхранената версия не може да бъде променяна в рамките на работното пространство. Нейната роля е да осигури 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. 18 Модел на работно пространство със съхранена версия [37]

Въпреки че разглежданият модел до известна степен решава въпроса с определянето на разликите между две версии и автономността на процеса на разработка, това създава друго проблемно положение. То се изразява в паралелното записване на версия, водещо до 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. 19 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.



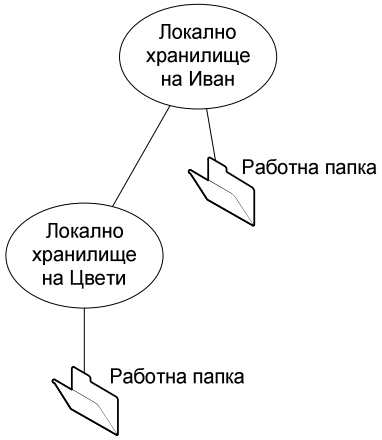
Фиг. 19 Примерна ситуация за конфликт между версии на един компонент

1.5.5. Модел на разпределени хранилища

Разпределените хранилища представляват модерен подход за осигуряване на паралелна работа над огромни продукти. Като най-известния пример за продукт, използващ разпределени хранилища, може да се спомене Git [41, 49]. Той е разработен специално за разработването ядрото на Линукс [14]. Друга популярна система за разпределено управление на версията е Mercurial [61]. 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 и със собствена работна папка

(работно простанство). Репозиторито на тези системи се синхронизира с т.нар. главно репозитори, което от своя страна се явява инсталация на тази система. Следва да се отбележи, че наличието на главно репозитори не е задължително. Освен това всяко едно хранилище може да изпълнява ролята на главно хранилище, за ново инсталираните екземпляри на системата. По този начин се постига структура от йерархично подредени хранилища.

Фиг. 20 Йерархична организация на разпределени хранилища



Фиг. 21 Работни папки свързани с локалното хранилище

При подхода на разпределени хранилища до голяма степен се ускорява аналитическата работа с него, което се намира локално. Въпреки предимствата, които предоставя архитектурата на разпределени хранилища, двете версионизиращи системи наследяват всички недостатъци от използване файлове като версионизиран обект.

1.5.6. Модели на йерархично композиране на работни пространства

В опитите си да намерят баланс между кооперативността и изолираността, различни изследователи лансират концепцията работните пространства да се подредят във формата йерархично-дървовидна структура, т.е. да се композират йерархично. Така например Силва и съавтори [84] търсят начин да разрешат недостатъците при конкурентен достъп до метаданните на общото работно пространство. В изследването те изоставят механизма на заключване, като изплзват 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 [52].

В [88] 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. Някои автори [35, 73] представят виртуални работните пространства, наричани още изгледи (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.

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

1. Драстично се намалява броят на колизиите при конкурентен достъп до версионизираните обекти.

2. Постига се естествена изолираност от промени извън работното пространство, в частност от промени, извършвани от съседни работни пространства.
3. Подобрява се общото бързодействие на разработване в следствие от реализирането на по-ефективен работен процес.

1.6. Методи за проследимост на промените

1.6.1. Проследимост на промените

Лошата практика да се предоставя 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 крайния продукт. Промените в изискванията и обхватът 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 [46] 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 [100] авторите правят опит за такова оценяване, базирано на проследяване на подобни промени в системата, които са били извършвани по-рано.

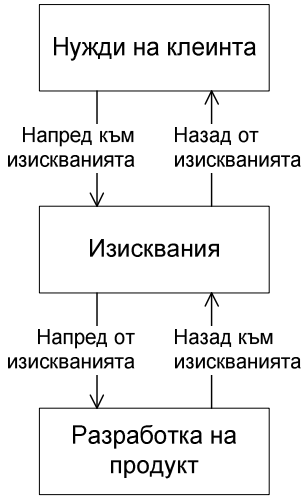
За ефективното управление на софтуерните проекти е необходимо ефективно да се управляват промените, както и да се използват ефективни подходи за оценяване на необходимите ресурси и време за реализация при промяна в изискванията или при определянето на дефекти в софтуерния продукт. Основен инструмент за осигуряването на това е проследимостта

на промените, който освен горните две предпоставки има голямо влияние и в области като [17, 42, 103]:

- 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 програмния продукт се улеснява, особено при използването на процес на системно подобряване и отстраняване на грешки.
- Улесняването на проследяване състоянието на проекта е следствие от използването на политика за прилагане на практики за проследяване на промени.
- Ускоряване на процеса на преработване и оптимизиране на определени части от системата.
- Преизползването на общи елементи в няколко системи се обуславя от наличието на връзки между дизайн, код и тестове.
- Намаляване влиянието на рисковете за проекта, вследствие от подобрените възможности за следене състоянието на проекта.
- При проверка качеството на софтуерните продукти връзките за проследяване предоставят информация затова, 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 предвид наличието на четири типа проследяващи връзки [77, 103]:

- Напред от изискванията – връзките от този тип определят работата или действията, които следва да бъдат направени.
- 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 от тип златна 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 [48, 103] осигурява възможност да се определи дали клиентските нужди са технически реализуеми, както и да се определи до каква степен техническото решение налага ограничаване или модификация на нуждите на клиента.



Фиг. 22 Типове проследяващи връзки [103]

1.6.2. Работни задачи

Хелминг в своя труд [42] въвежда понятието „работна задача” (work item) като дефиниция за работата, която следва да се извърши. Пак там авторите представят следната класификация на работните задачи:

- Задачии за действие (action items). Те представляват определена съвкупност от дейности, която произтича както от изискванията

към крайния продукт, така и от методологията, която се следва в процеса на създаване на продукта.

- Въпросите (issues) представляват работни задачи, резултатът от които е да изяснят, допълнят или коригират работни задачи от другите видове.
- Репорт за дефект (bug reports). Репортът за дефект представлява искане за промяна във функционалността на системата, с цел достигане на пълно съответствие на определени общоприети стандарти и изисквания към продукта.
- Работен пакет (workpackage). Работният пакет може да се разглежда като пакет, в който се съдържа съвкупност от работни задачи от горните три вида. Освен това той може да съдържа и други работни пакети в себе си. Чрез него се осигуряват различните нива на абстракция на заданията към различните участници в процеса.

1.6.3. 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 (оперативна поддръжка на системите, маркетинг, финанси, контрол, отношения с клиенти, планиране, логистика и др.) води до генерирането на голямо количество от изисквания. Те могат както да се променят, така и да се отменят, и разбира се да се заменят с нови такива.

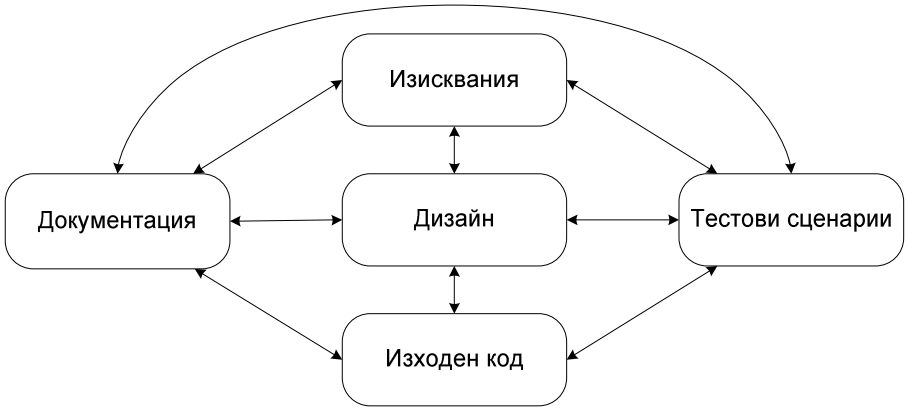
Системите за управление на изискванията предоставят възможности за проследяване на техния жизнен цикъл чрез използване на методологии от версионизирането. Самите изисквания и исканията за промени са добър пример за работни задачи.

1.6.4. Методи за проследяване на промени

Едно често срещано явление при създаването на софтуер е лошата практика – документът с бизнес спецификация или този със системна спецификация да се предоставя в завършен и непроменяем вид. Това неминуемо представлява голям риск за успеха на проекта, тъй като може да доведе до увеличаване на разходите. Един такъв документ много трудно може да покрие всички функционални аспекти, на които трябва да отговаря крайният продукт. Изискванията и обхватът 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 документът с изискванията би следвало да се смята за жив документ, който да се развива паралелно със софтуерния продукт. За улесняване на дейностите по промяна на изискванията и техният обхват, авторите на [77, 46] 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 (Фиг. 23). Те следва да се разглеждат като неделима част от процеса на създаване на софтуерния продукт – изисквания, системен дизайн, тестови сценарии, гарантиращи качеството на продукта, изходният код и документацията. Активното използването на проследимостта между отделните артефакти ни дава следните предимства [17, 42, 103]:

- Улеснен анализ и оценка на влиянието на дадена промяна както над целия 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 др.



Фиг. 23 Свързване на софтуерните артефакти за проследяване на промени

Клеланд-Хуанг [25] представя класификация на 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.

1.6.4.1. 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 с течение на времето е възможно данните в матриците да не съответстват на действителността. Тяхното широко разпространение се обяснява от най-голямото им предимство - тяхната простота.

1.6.4.2. 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 целия жизнен цикъл на продукта. Решаването на тази задача се явява и основна слабост на метода в посока разширяемост и дълговременна поддръжка на връзките.

В своето изследване Жанг [112], използвайки подхода на обратното инженерство, прави опит за създаване на връзки за проследимост между изходния код и документацията на семантично ниво. Следва да се отбележи, че според автора съществува липса на адекватни инструменти за създаване и управление на връзки на проследимост. В допълнение на това може да се изтъкне и проблемът с правилното определяне на семантиката на естествения език в документацията.

1.6.4.3. Аспектно изграждане (Aspect Weaving)

Методът на аспектно изграждане е приложим при използване на Аспектно Ориентиран Подход (АОП, англ. Aspect Oriented Programming) на създаване на системитец [108]. Той се базира върху пресичане на практиката подобни по своята същност функционалности да се намират в различни части на системата. В АОП 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 АОП осигурява връзката между аспектите и имплементацията. За съжаление Аспектното изграждане не предоставя добри възможности за оценяване обхвата и влиянието на промяна над съществуващ продукт.

В [94] се прави опит за свързване на изискванията към изходния код, при който се налага необходимост от структуриране на изискавнията.

1.6.4.4. Извличане на информация

Този метод се появява в следствие на опитите на изследователите да автоматизират дейносттите по изграждане и поддържане на матриците за проследимост. Техните изследвания [26, 60] са насочени именно в посока извличане на нужната информация от изходния код и намиране на тяхното съответствие в изискванията. Основен акцент на метода е обработката на подобности, при която обработка се анализира честотата на срещане, както и самото описание 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.

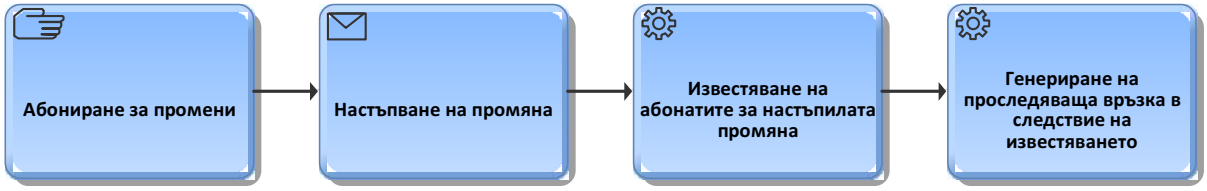
Първоначално определянето на обхвата и влиянието на допълнителните промени не са били предмет на метода. Това предполага използването на ръчни методи за тяхното определяне. Но на по-късен етап изследователи [100] правят разработки в посока автоматизиране на анализа при използване на метода.

Като слабост на методите, базирани на извличане на информацията, може да се изтъкне факта, че те обикновено изключват структурата и семантиката на информацията, която обработват. По този начин те имат по-слаба прецизност и приложимост.

1.6.4.5. Проследимост, базирана на събития (Event-based Traceability)

Проследимостта, базирана на събития, представлява метод, при който се използва механизъм на абониране за промени и известяване на абонатите за настъпилите промени, с цел генериране на проследяващи връзки. 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.

Методът е прозрачен от към семантиката, реализирана в артефактите, водещи до промяна (изисквания, дефекти и др.), като задачата на определянето отговора на въпроса „какво се работи в момента” е предоставен на потребителя. В резултат на това се постига сравнително просто решение във висока степен на адекватност и прецизност.



Фиг. 24 Генериране на проследяващи връзки при метод, базиран на събития

1.7. Изводи

От направеното изложение в Първа глава на настоящата дисертация се налагат следните изводи:

1. Системите за управление и контрол на версии представляват задължителен инфраструктурен инструмент в съвременното софтуерно производство. Може да се каже, че тези модели играят водеща роля в процеса на създаване на софтуерни продукти.
2. Експонирани и систематизирани са различни модели за представяне и съхраняване на версионизирани обекти. Изтъкнати са предимствата и недостатъците на разгледаните модели. Определена е липсата на адекватно ниво на поддръжка на версионизиране на различни нива на абстракция на системите. Тази необходимост може да се трансформира в изискването версионизираният обект да предоставя възможност за определяне на различно ниво на неговата степен на гранулираност.

3. Направен е анализ на различни подходи за съхраняване на версионизирани обекти. Подходът за съхраняване състояния на версионизираните обекти, предполага проста реализация и по-висока скорост на системата. Това го превръща в атрактивен кандидат за реализацията на прототипа.
4. Показани са предизвикателствата, стоящи пред съвместната работа над един продукт. Определена е възможността от научно изследване на моделите и механизмите, използвани в областта на йерархично композираните работни пространства [35 – стр. 406]. Установено е, че йерархично композираните работни пространства служат като инструмент за осигуряване на автономна работа. Това се допълва с възможността за кооперирane на работата между участниците в процеса по създаване на софтуерни продукти.
5. Направен е анализ на темата за проследимост на промените. Представени са различните видове проследяващи връзки, както и методите за получаването им. Идентифицирана е липсата на инструменти, предоставящи адекватно ниво за създаване и управление на връзки на проследимост [112].

Направените изводи са използвани като мотивационна предпоставка при формулирането на целта и задачите на дисертационният труд.

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

В настоящата глава са представени теоретичните модели, чиято разработка е пряко свързана с успешното решаване на задачите на настоящия научно-приложен труд. В параграфите, съставящи главата, са 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.

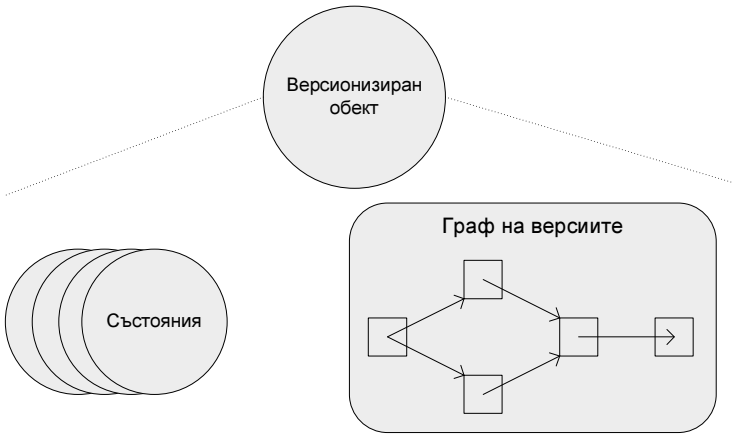
2.1. Модел на версионизиран обект

Настоящият параграф има за цел да представи модел на версионизиран обект. Моделът предоставя възможност за гъвкаво комбиниране между версионизирани обекти, като се изграждат композиции от обекти. Построяването и представянето на модел на версионизиран обект е реализирано като класически модел същност-отношение (Entity-Relationship Model). Той ни предоставя добър механизъм за гъвкаво и свободно реализиране на поставените задачи.

Както беше споменато в Първа глава, водещите автори в областта на управлението и контрола на версиите [29, 85] определят версионизираните обекти като обекти съставени от две части – състояния на обекта (версии) и граф на версиите. Под граф на версиите се разбира такъв граф, чиито върхове представляват отделните състояния (версии) на обекта, а ребрата съответстват на логическата последователност на

създаване на версиите (Фиг. 25). В настоящото научно-приложно изследване термините „версионизиран примитив”, „версия на обект” и „състояние на версионизиран обект” ще бъдат използван като синоними.

Този модел не отговаря на изискванията на поставените по-горе и така изграденият модел следва да се подобри и развие.



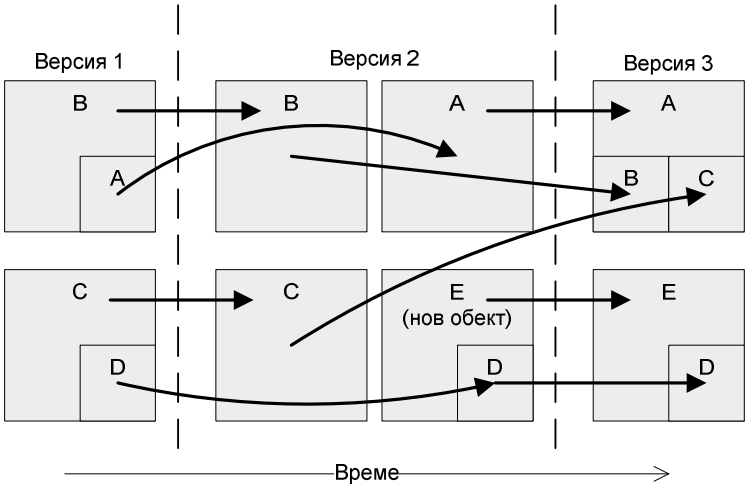
Фиг. 25 Концептуален модел на версионизиран обект

Основна 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 самия т 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 1. Съставен обект се нарича обект, който е съставен от други обекти (версии на обекти) посредством композиция.
Дефиниция 2. Под композиция ще се разбира същността, 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 композираните обекти и съответно на под-обектите в предметната област води до необходимостта да се преопредели процесът на версионизиране на обектите. Така на Фиг. 26 е представен пример за

промени в композицията на обектите, като със стрелки са показани промените в състава на обектите между отделните версии на крайния продукт. Детайлно изследване на версионизирането на съставни обекти е представено в следващата точка на параграфа.



Фиг. 26 Пример за промени на съставността на обектите

При свързването на версиите на един версионизиран обект е необходимо да се използва релационна връзка от тип външен ключ (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.

Версиите на един обект може да се разглеждат като негови примитиви (**версионизирани примитиви**), чиито основни атрибути са следните:

- Номер на версионизиран обект, с който дадената версия е свързана.
- Номер на версия – пореден номер, който определя по уникален начин версията в рамките на обекта.
- Наименуване на обекта. Определяйки наименованието на ниво примитив, потребителят получава възможност да проследява отделните версии даже и при преименуване на обектите. Така

получеият модел става по-пълноценен, елиминирайки недостатъка, свързан с преименоването на обектите (файловете) при системи като CVS, SVN, Git, Metcury и др. [28, 41, 49, 61, 64, 75].

- Съдържание на обекта, включващо 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 [11] препоръчва 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 обекти, следва да се дефинира допълнителна същност - „Композиция на версионизирани примитиви" (накратко композиция), която еднозначно свързва версията на супер обекта с версиите на неговите под-обекти. Като атрибути на композицията може да се определят следните:

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

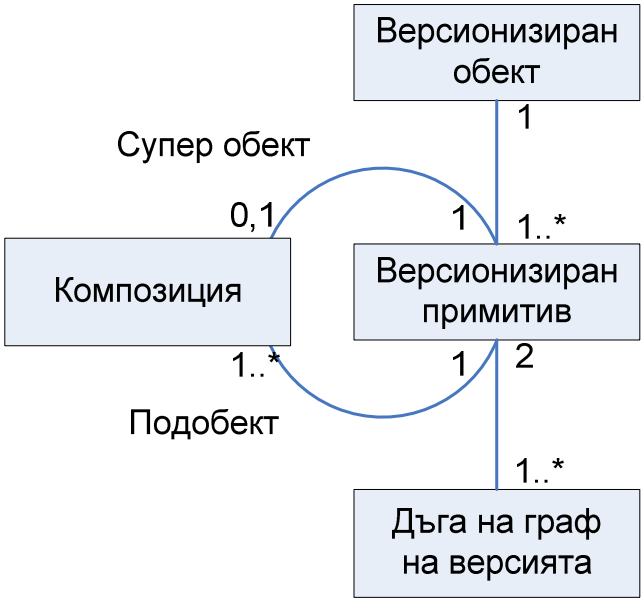
Въпреки че комбинацията от двата атрибута е уникална и еднозначно определя една нейна инстанция, за разглежданата същност е необходимо да се използва отделен атрибут за първичен ключ – номер на композиция.

За нуждите на отчетността и проследимостта на промените, така създаденият модел следва да се разшири с цел да се осигури поддържка на функционал за граф на верссите. В ER моделите е прието графовата структура да се моделира от две същности – същност на възлите и същност на дъгите [86]. 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. 27 е представена ER диаграмата на построения модел на версионизиран обект.



Фиг. 27 ER модел на версионизиран обект

2.1.1. Версионизиране на съставен обект

Настоящата точка 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.

Дефиниция 3. Съставен обект от ред **0**, т.е. прост обект, ще наричаме такъв обект, за който няма асоциирани под-обекти. Съставен обект от ред **N** ще наричаме такъв обект, за който най-големият ред на асоцииран под-обект е равен на **N-1**.

$$R_{\text{обект}} = \begin{cases} 0, & \sum \text{под} - \text{обекти} = 0 \\ N, \max\left(R_{\text{под}-\text{обект}}\right) = N - 1 \end{cases}$$

Степен на гранулираност на обект ще се нарича реда на обекта.

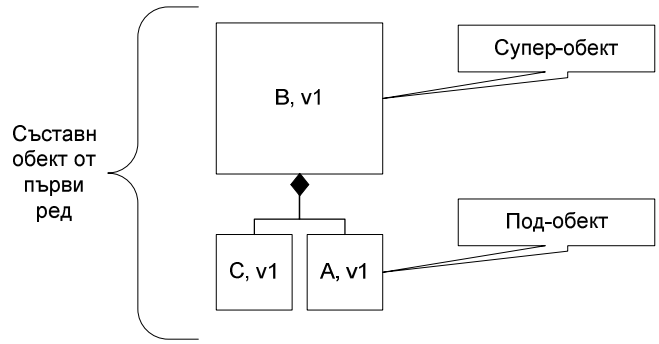
Важно е да се вземе под внимание фактът, че в дефиницията на съставен обект (по-горе) не са налагани никакви ограничения над под-обектите, от което може да се изведе следствието:

Следствие 1. Един под-обект сам по себе си може да се явява съставен обект от други обекти, като по този начин да се създаде композиция от съставни обекти.

Една от основните задачи, която стои пред настоящия научно-приложен труд, е да не се усложнява без необходимост тук създадените модели. Изхождайки от това, както и от факта на липсваща практическа необходимост, при построяването на композиция от съставни обекти следва да въведем следните ограничаващи правила:

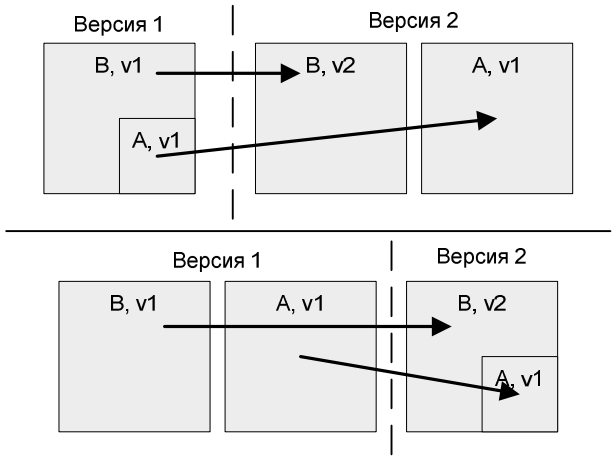
Правило 1.В дадена композиция от съставни обекти, обект може да присъства най-много един път.

Правило 2.Един обект може да присъства най-много в една композиция от обекти.



Фиг. 28 Дърво от обекти

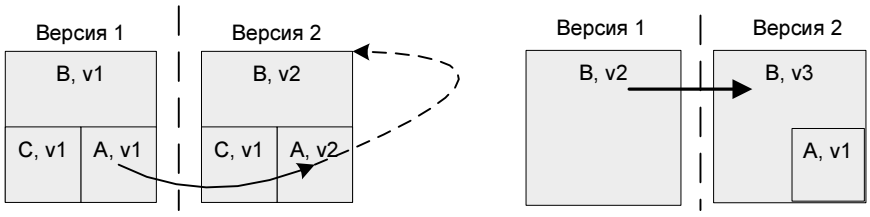
При промяна в съставността между два обекта, следва да разглеждаме версиите на обектите като различни (Фиг. 29). Така например нека се разгледа един стол (супер-обект) с подлакътници (под-обекти). Когато се отделят подлакътниците от стола, се получава нова версия на стола – стол без подлакътници. Трябва да се отбележи, че конкретният под-обект не си променя версията, т.е. в описания случай подлакътниците си остават подлакътници. Така се получава единствено промяна в композициите на супер-обекта. Подобно е положението и при „сглобяването" на композиран обект, където има промяна на версията само на обекта, който става супер-обект. Така, от един стол без подлакътници, при добавяне на подлакътници, се получава нова версия на стола, без да има промяна във версията на подлакътниците.



Фиг. 29 Промяна в композицията на обекти чрез промяна на версия

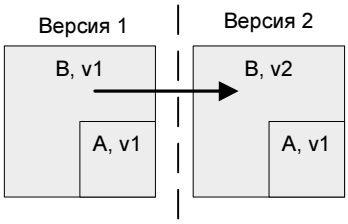
Друга особеност при съставните обекти, която непременно следва да се разгледа, е, че при промяна на под-обект (т.е. промяна на неговата

версия), се получава индиректна промяна в съставния обект (Фиг. 30). Така например при промяна на цвета на тапицерията на един стол от червен към син, на практика, освен новата версия на под-обекта, се получава нова версия на целия стол – стол със синя тапицерия. Като частни случаи на промяна на под-обект може да се приеме асоцииране на обект като под-обект, както и премахването на асоциация с под-обект и неговата (на под-обекта) трансформация като нормален обект.



Фиг. 30 Индиректна промяна на версията на съставен-обект при промяна на съставящ-обект

Обратното положение – при промяна на версията на супер-обекта – не означава, че има 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 променя версията (цвета) на под-обекта тапицерия (Фиг. 31) .

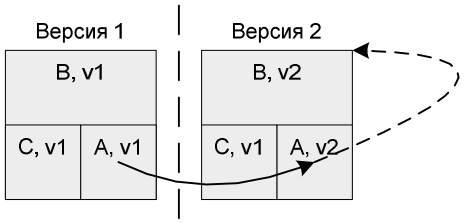


Фиг. 31 Промяната на супер-обекта не влияе на версията на под-обекта

От последните две правила може да се изведе следствието:

Следствие 2. Промяна на версията на даден под-обект за даден супер-обект, не влиае на версиите на другите под-обекти, съставящи същия супер-обект (Фиг. 32).

Моделът на съставен обект и принципите на видимост от предишният параграф ни навеждат на необходимостта от разглеждане на проблема на видимост на съставен обект.



Фиг. 32 При промяна във версията на един под-обект не се променя версията на съседните под-обекти

Следствие 3. Версия на даден съставен обект е видима в дадено работно пространство само и единствено, когато всички версии на съставящите го под-обекти са видими в съответното работно пространство.

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

2.2.1. Модел на йерархично композирани работни пространства

Преди да се представят моделите в следващите точки на настоящия параграф, ще се разгледа моделът, на който те стъпват – моделът на йерархично композирани работни пространства. В рамките на този модел се използват следните понятия:

Дефиниция 4. Продукт се нарича обект на материалното или нематериалното производство, който след своето създаване може да бъде размножен и разпространяван сред клиентите.

Дефиниция 5. Издание на продукт се нарича определена фиксирана негова версия, която е преминала определени количества проверки и отговаря на определени критерии за качество, безопасност и др. Само издания на продукта се разпространяват сред клиентите.

Версии, които не представляват издание, се наричат в практиката работни версии.
Дефиниция 6. Работно пространство се нарича място, където се извършват определени дейности по създаването на версия на продукт.
Дефиниция 7. Главно работно пространство се нарича работно пространство, в което се извършва окончателната сборка и подготовка на издание на продукта.

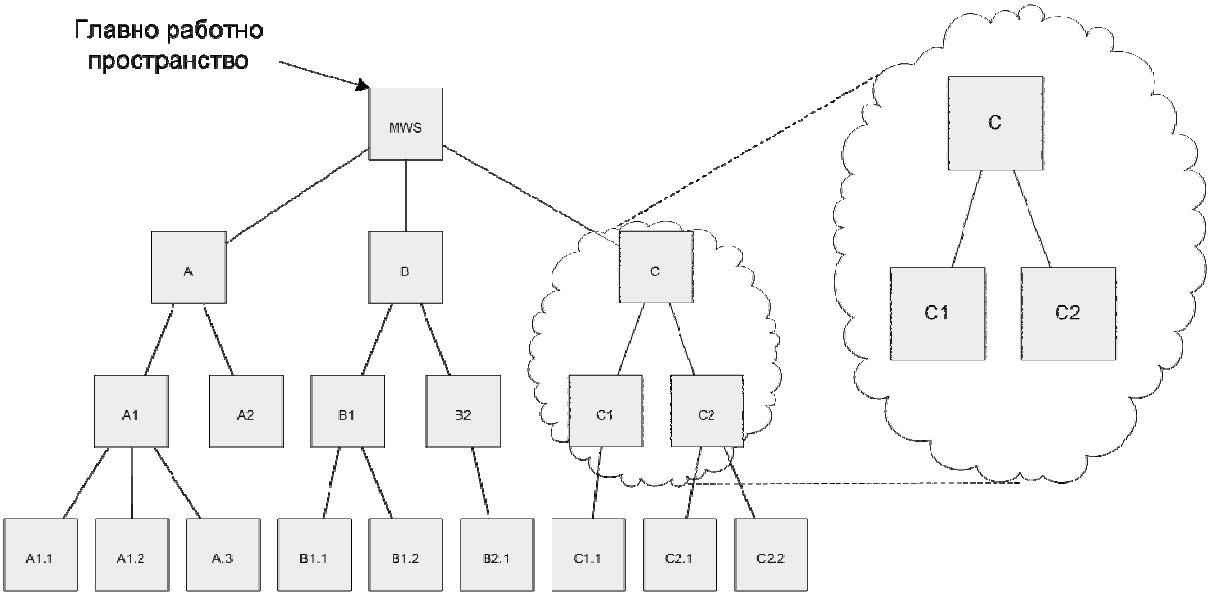
Подреждането (композирането) на работни пространства се предприема с цел да се предостави възможност на всеки един от участниците в процеса на разработване на продукта и неговите издания да извършва своите дейности както самостоятелно, така и кооперирайки се с останалите участници. Работното пространство осигурява именно възможността за самостоятелна работа, която не влияе и не се влияе от работата на останалите участници. От друга страна, композирането на работните пространства в йерархия претендира да бъде механизъм за осигуряване на кооперативната работа. На Фиг. 34 е представена диаграма на йерархично композиране на работни пространства.

Фиг. 33 Клас диаграма на модел продукт-издание-работно пространств

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

Настоящото 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 отношенията родител-наследнъик, като се поставя акцент на видимостта на

версионизираните обекти, определена от представени принципи на видимост.



Фиг. 34 Примерна йерархична композиция на пространства

Дефиниция 8. Под *локална версия на версионизиран обект* за дадено работно пространство, ще се разбира такава негова версия, която е асоциирана с работното пространство.
Дефиниция 9. Под *видима версия на версионизиран обект* за дадено работно пространство, ще се разбира такава версия на обекта, с която потребителят може да работи.

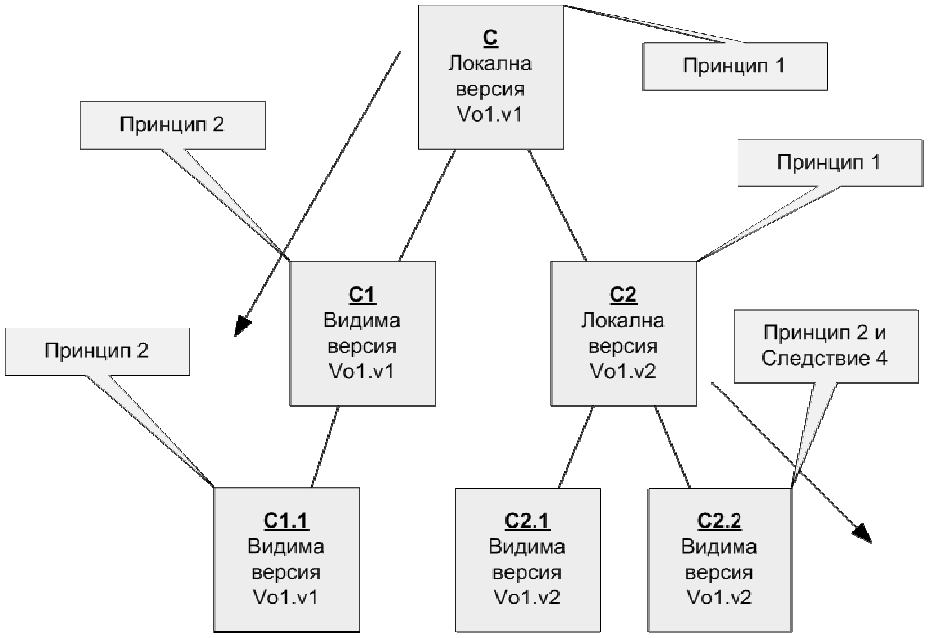
Принципи на видимост:

Принцип. 1. *Локалната версия на версионизиран обект* за дадено работно пространств се явява *видима версия* на този обект в същото работно пространство, въпреки наличието на други локални версии в родителските пространства.
Принцип. 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:

Следствие 4. Във всяко работно пространство, където обектите нямат локална версия, те са представени с тяхна версия, намираща се в най-близкото родителско работно пространство.
Следствие 5. Ако за дадено работно пространство обектът няма версия в нито едно родителско работно пространство, то той не се вижда в първоначално избраното работно пространство.

На Фиг. 35 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. 35 Разпределение на версиите на версионизиран обект съгласно принципите на видимост

С цел постигане пълнота и коректност на представения модел може да се формулира следното **следствие-ограничение**: един обект може да присъства само с една версия в дадено работно пространство.

2.3. Транзакции над версионизирани обекти

2.3.1. Транзакции над версионизиран обект в рамките на едно работно пространство

В този параграф е направен опит да се представят транзакциите над версионизирани обекти в рамките на едно работно пространство. По-долу са разгледани следните транзакции: създаване на версионизиран обект; актуализиране на нелокален 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 представения модел последното състояние се освобождава, а текущата 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 последващ по-бърз и по-лесен анализ на извършената работа.

Актуализирането на нелокален версионизиран обект, т.е. на обект, който няма локална версия в текущото работно пространство, може да се определи като най-важната функционалност на текущия параграф. 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 този параграф транзакции над обекта вече не могат да се извършват, с изключение на транзакцията по отказ от маркер на състояние.

2.3.2. Транзакции над версионизиран обект между две работни пространства

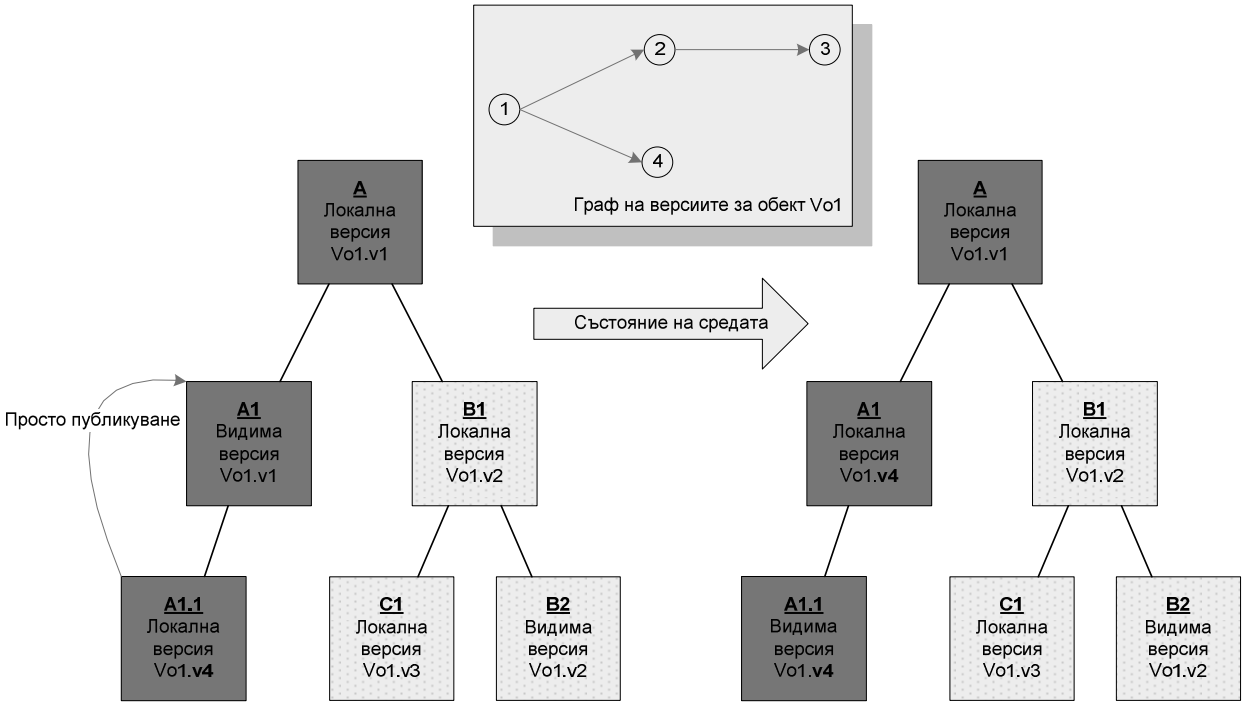
Транзакциите между две работни пространства могат да се разделят на две групи – публикуване на версия на обект и отказ от локална версия. Преди да се разгледат, е необходимо да се въведат термините „производна" и „паралелна" (непроизводна) версия на обект.

Дефиниция 10. Нека разгледаме един версионизиран обект и две негови версии X и Y. Ако съществува път в графа на версии на обекта от версия X до версия Y, то версия Y се нарича *производна версия* на версия X, а версия на X – предшестваща версия Y.
Дефиниция 11. Нека разгледаме един версионизиран обект и две негови версии X и Y. Ако не съществува път в графа на версиите за обекта

от версия X до версия Y, то двете версии се наричат *паралелни* или *непроизводни версии*.

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

Простото публикуване на версия е при ситуация, когато в родителското работно пространство не съществува локална версия на публикувания обект – Фиг. 36.

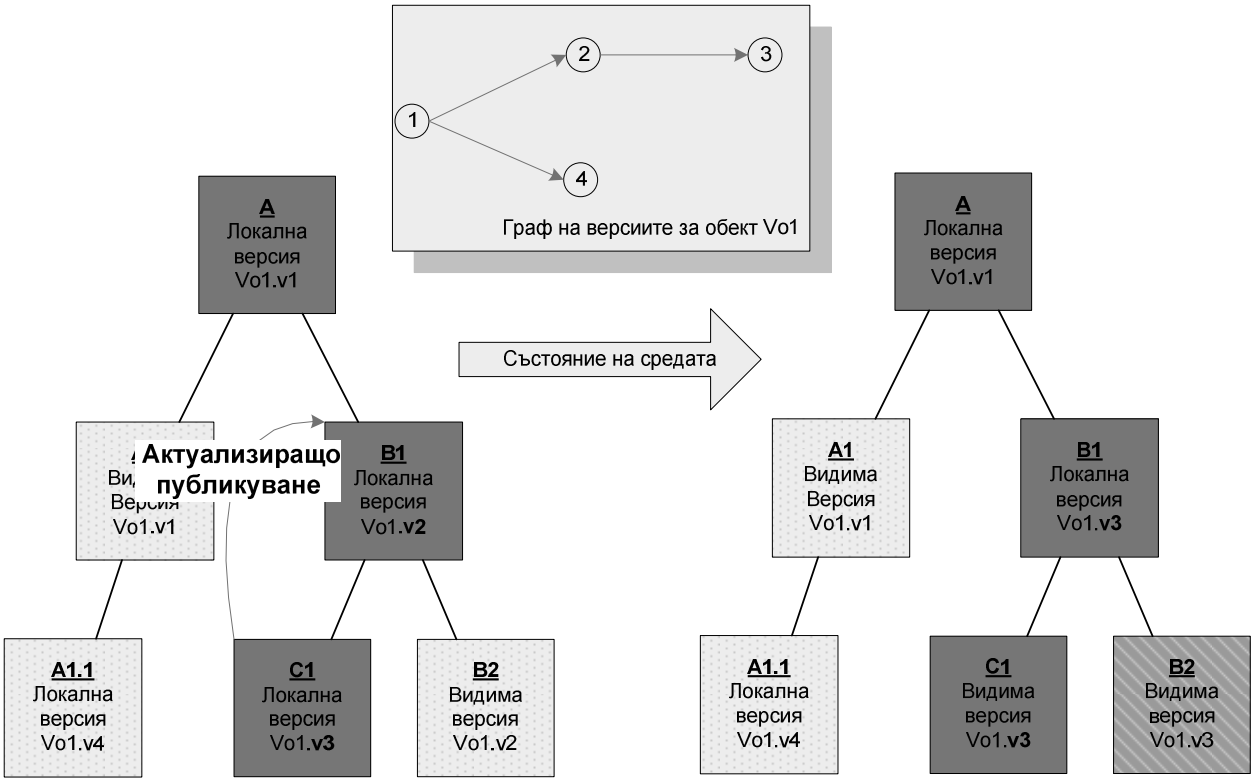


Фиг. 36 Просто публикуване

Следващата транзакция, която е необходимо да се разгледа, е тази за актуализиращо публикуване (Фиг. 37). Характерно при нея е, че тя е възможна, когато бъдат едновременно изпълнени следните две условия:

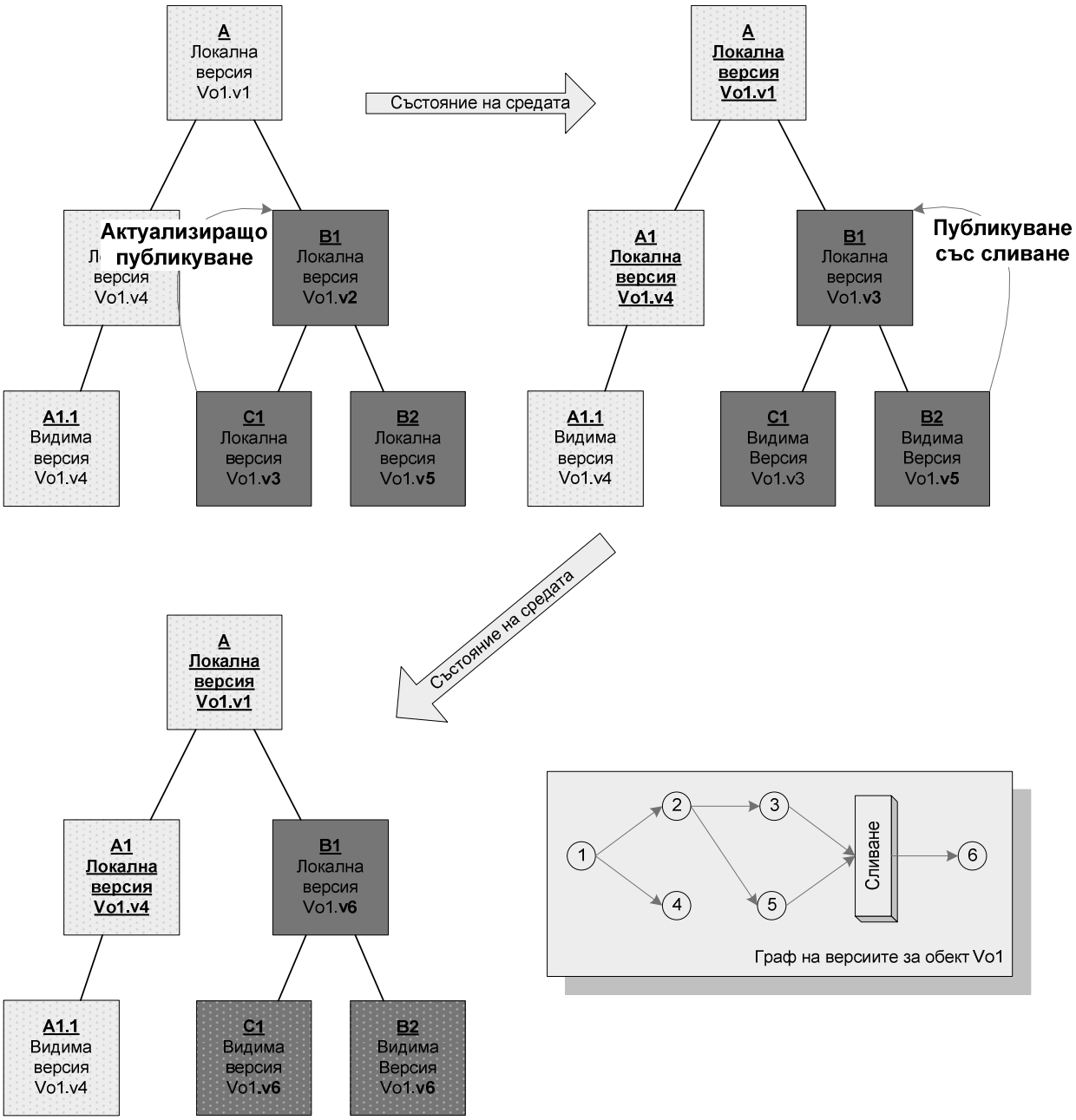
- В родителското работно пространство съществува локална версия на обекта, който се публикува.
- Версията на обекта, който се публикува, се явява производна на версията му в родителското работно пространство.

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 разглеждания обект.



Фиг. 37 Актуализиращо публикуване

Когато версията на обекта, която се публикува в родителското работно пространство, се явява паралелна спрямо намиращата се там локална версия (Фиг. 38), 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 споменатия по-рано алгоритъм на Вестфехтел [101], както и сливане чрез двойно и тройно сравняване [31].



Фиг. 38 Публикуване със сливане

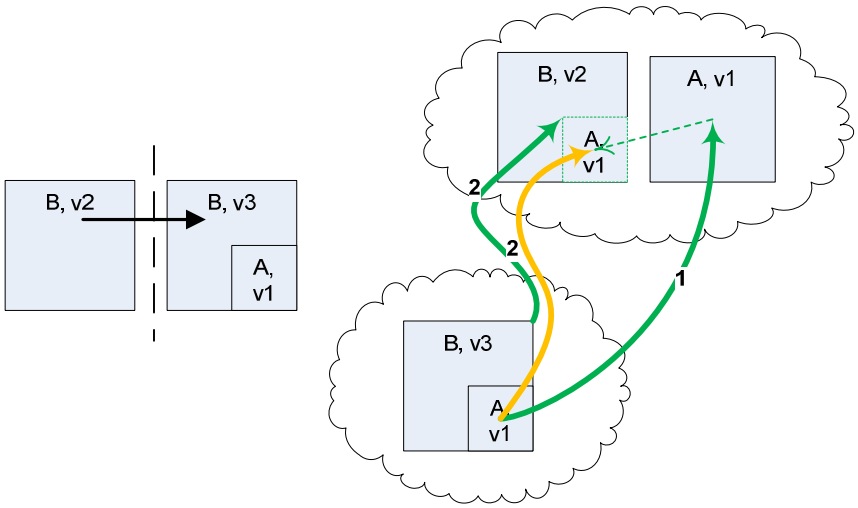
Транзакцията по отказ от локална версия се явява обратна на транзакциите по публикуване на версия. Тя включва само една стъпка: премахване на локалната версия на обекта от работното пространство. При премахването сработват механизмите от **Следствие 4** от модела на видимост на версионизираните обекти. Важно е да се отбележи, че ако в нито едно от родителските работни пространства не съществува версия на избрания обект, то той става недостъпен за последваща употреба, т.е. той

се изтрива. Това положение следва да се отчита, когато транзакцията се извършва в главното работно пространство на изданието на продукта.

2.3.3. Транзакции над съставни обекти

Настоящия пункт има за цел да определяг принципите и особеностите при изпълнение на транзакции над съставен обект след промяна в неговата композиция.

Нека се разгледа ситуацията, когато имаме локална версия на обекта B в родителското работно пространство и негова видима версия в текущото работно пространство. В текущото работно пространство се създава под-обект A за обекта B (Фиг. 39). При публикуването версията на под-обекта A е възможно да не води до промяна във версията на обект B в родителското работно пространство. Въпреки това при последващо публикуване версията на обекта B заедно с неговите композиции, в родителското работно пространство ще доведе до автоматично обновяване (в рамките на работното пространство) на композиционната схема на обектите (Фиг. 39 – зелената пунктирана стрелка). Това е продиктувано от факта, че информацията относно организацията на съставния обект следва да се разглежда като неделима част от него.



Фиг. 39 Новосъздаен под-обект към супер-обект

Публикуването на новата версия на съставния обект B,v3 води до изискването това да се извърши в комплект с версията на новосъздадения под-обект (Фиг. 39 – стрелките с №2).

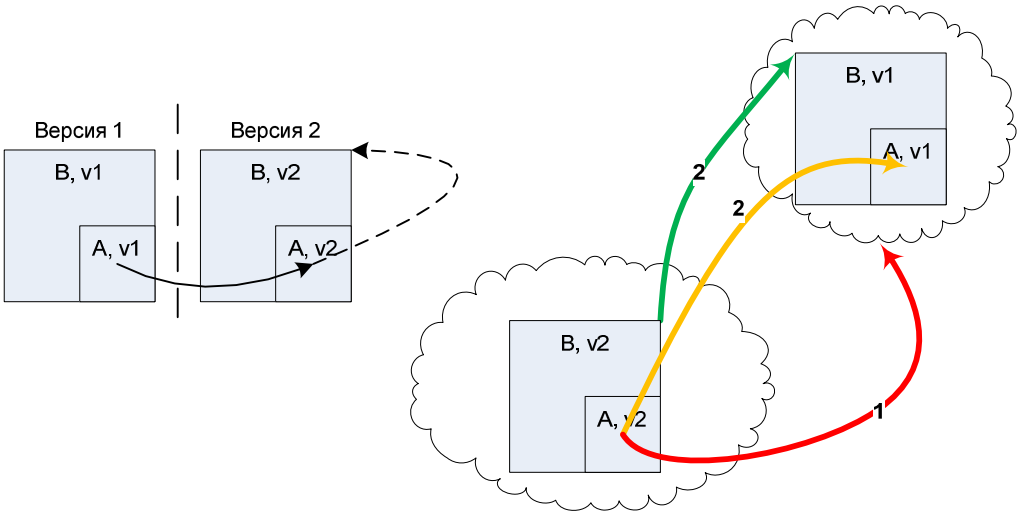
Нека се разгледа ситуацията, когато имаме локална версия на обекта B в родителското работно пространство, която е видима в текущото работно пространство (Фиг. 40). В текущото работно пространство се извършва промяна в под-обекта A, която води до промяна на обекта B. Именно създаването на нова локална версия на под-обекта води до автоматичното създаване на нова локална версия на съставния 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. 40 – 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 3. 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. 40 – зелената и жълтата стрелки с №2).

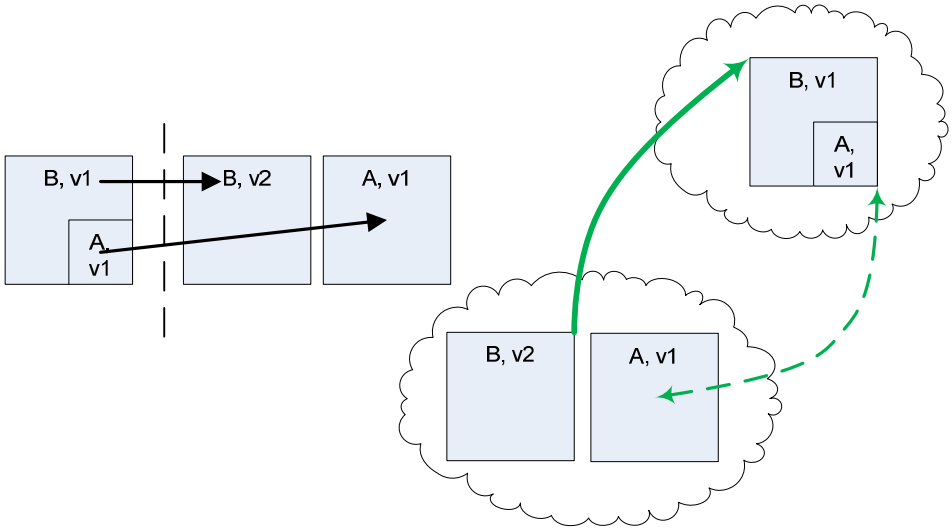
Локалните версии на под-обекти в родителското работно пространство могат да бъдат както производни, така и паралелни. В тези случаи е необходимо да се изпълнят съответните транзакции, които са разгледни в предишния пункт.

Нека се разгледа същата ситуация, когато имаме локална версия на обекта B в родителското работно пространство, която е видима в текущото работно пространство (Фиг. 41). От композицията на съставен обект се изключва под-обект. Отразяването на тази промяна в родителското

работно пространство е достижимо единствено чрез публикуване на съставния обект. Тази публикация води само до отразяване на промяната в композицията в родителското работно пространство, без да се променя версията на под-обекта, който в новата версия на супер-обекта вече не е съставна част от него.



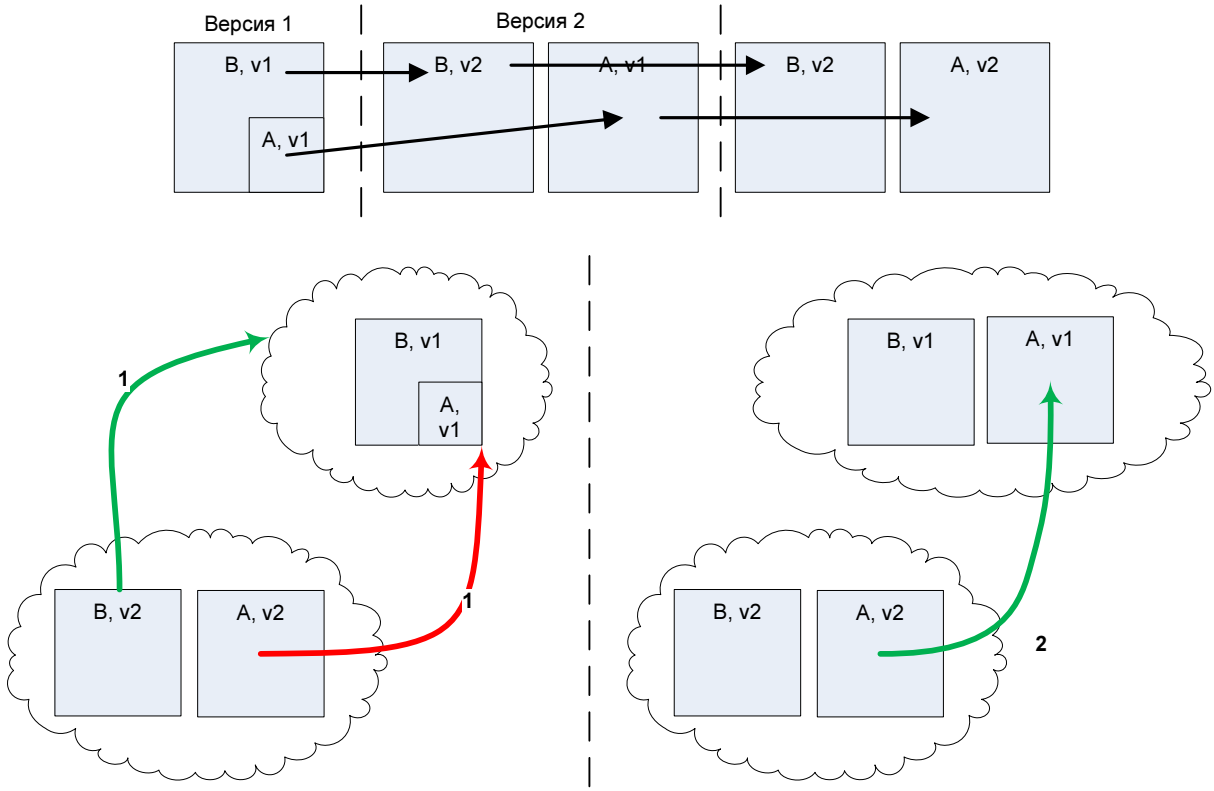
Фиг. 40 Индиректна променена версия на супер-обект, породена от нова версия на под-обект



Фиг. 41 Липса на промяна във версията на обект A, т.е. няма необходимост от неговото публикуване

Нека имаме видим съставен обект A с под-обект B, като обектът A и под-обектът B са локални версии в родителското пространство. От композицията на даден съставен обект A премахваме под-обекта B. След

това потребителят модифицира обекта B, т.е. създава нова негова версия (Фиг. 42). При така създадената ситуация публикуването новата версия на под-обекта B би довело до следната конфликтна ситуация: Версия v1 на обект B изисква в работното пространство обекта A да присъства (видимо или локално) с версия v1.



Фиг. 42 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 формулиране на правилото:

Правило 4. 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 съответния съставен обект.

Както бе отбелязано по-горе, обратната транзакция на публикуването се явява отказът от локална версия. При отказа от локална версия на съставен обект следва да се отчита фактът, че неговата версия до

голяма степен зависи от версията на съставящите го обекти. Това води до следната формулировка на правило за отказ на съставен обект:

Правило 5. Отказът от локална версия на съставен обект следва да се извършва заедно с рекурсивен отказ от локална версия на всички негови под-обекти.

2.3.4. 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 на тези действия в текущия пункт е представена класификация на транзакциите над версионизирани обекти в среда с йерархично композирани работни пространства.

Групиране	**Тип обекти**	**Транзакции**
Транзакции в рамките на едно работно пространство	Прости обекти	Създаване на версионизиран обект
		Маркиране на версия
		Актуализация на нелокален 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.
		Отказ от локална версия

Таблица 1 Класификация на транзакциите над версионизиран обект в среда с йерархична композиция на пространства

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.

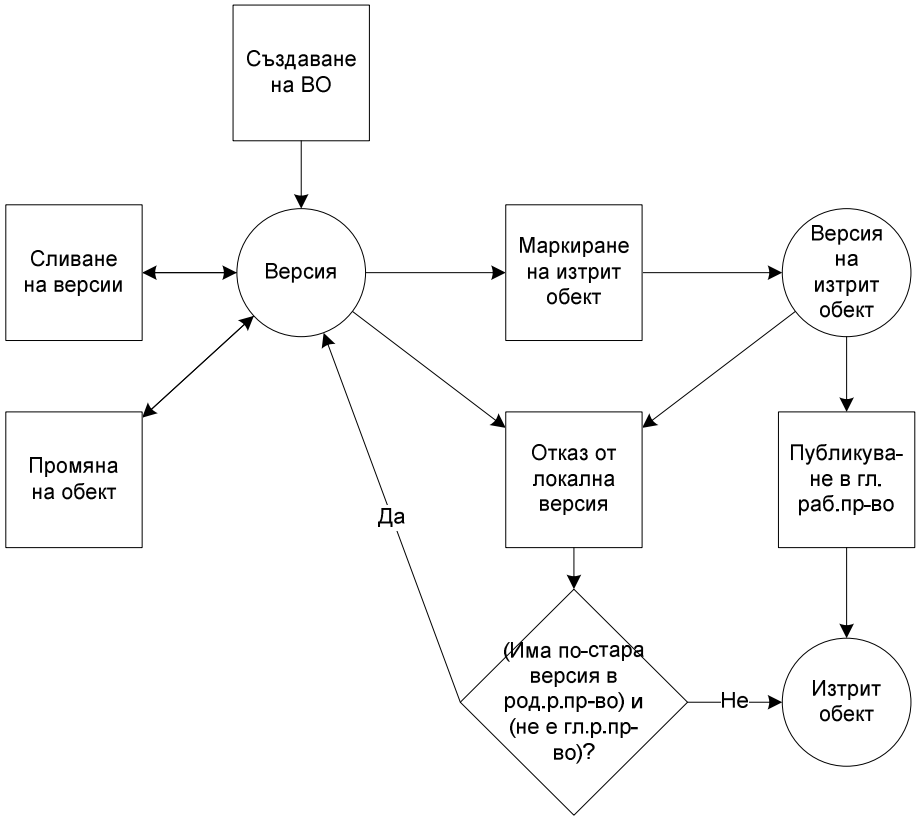
2.3.5. Жизнен цикъл на версионизиран обект

Под жизнен цикъл ние ще разбираме последователността от събития и състояния на даден обект от неговото създаване, до неговото изтриване - изтегляне от употреба. За описване на жизнения цикъл на версионизиран обект са използвани представените в 2.3.4 транзакции. На Фиг. 43 е представен модел на жизнения цикъл на версионизиран обект чрез използването на диаграма на поредица от събития (event-driven chain). Моделът включва следните 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.
- Публикуване на версия на обект, означен като изтрит, в главно работно пространство.

На Фиг. 43 са използвани следните означения: кръглите елементи представляват състояния на обекта в неговият жизнен цикъл, а квадратите – транзакциите по промяна на обекта.



Фиг. 43 Диаграма на жизнен цикъл на версионизиран обект

Както се вижда от диаграмата, в жизнения цикъл на един обект може да се отбележи по-особеното поведение на транзакцията по отказ от локална версия на обект. При нейното изпълнение следва да се отчита какво се явява това работно пространство – главно работно пространсто или не. От това зависи дали обектът да бъде изтрит от текущото издание или не.

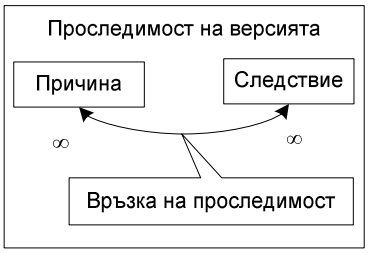
При детайлен анализ на диаграмата, представена на Фиг. 43, може да се формулират следните моменти от жизнения цикъл на версионизираните обекти:

- Жизненият цикъл на всеки версионизиран обект започва с неговото създаване.
- Като крайна транзакция на версионизиран обект може да се отбележи „Маркиране на изтриване”.
- Транзакцията „Отказ от локална версия” играе роля на крайна за жизнения цикъл на обекта, ако във всички родителски работни пространства този обект няма версии.
- Транзакцията „Маркиране на изтриване” е възможно да не е крайна за жизнения цикъл на обекта, при условие че в родителските работни пространства съществува версия на обекта и се приложи транзакцията „Отказ от локална версия”. 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).
- Трите транзакции, свързани с публикуването на обектите в родителското работно пространство, могат да се изпълняват последователно и в зависимост от конкретната версия на обектите между родителското и дъщерното работно пространство, като автоматично се прилага една от трите представени конфигурации, разгледани в предишните пунктове.

2.4. 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 Първа

глава, е задачата за анализирането на промените. Настоящия параграф е фокусиран на създаването на модели, подпомагащи анализа и проследяване на промените. Основните причини, 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 представения модел ще ги наричаме *причини*. За извършването на пълноценен анализ причините следва да бъдат асоциирани с извършените промени над обектите, които могат да се дефинират като *следствие*.



Фиг. 44 Проследяваща връзка между причината и следствието

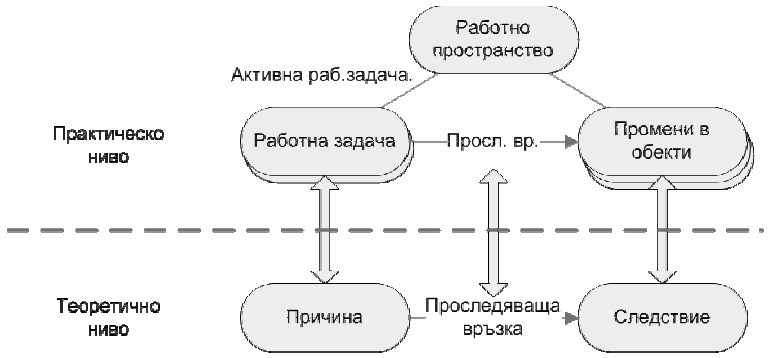
От направения анализ на методите за проследимост на промени в Първа глава може да се приеме, че методът, притежаващ най-висок потенциал за развитие, е този, базиран на събития. В текущия параграф е направен опит за адаптиране и интегриране на метода в разглежданата среда с йерархична композиция на работни пространства. Моделирането на метода за проследяване на промените може да се разглежда като допълнително звено, свързващо модела на работните пространства с този на версинизиран обект, като се запази възможността за гъвкаво дефиниране нивото на гранулираност на обектите, с които се работи.

2.4.1. Работни задачи и работни пространства

В настоящия пункт е представено основно понятие в процеса на създаване на софтуер – работна задача. Тук под работната задача ще се използва разширение на определението, което дава Хелминг в своя труд [42] – Работна задача е работата, която следва да се извърши.

Дефиниция 12. Работна задача се нарича съвкупността от дейности, която следва да се извърши.
Дефиниция 13. Следствие, породено от причина, ще се нарича наборът от промени над обекти в резултат от изпълнението на работна задача.

На Фиг. 45 е направен опит да се представи под формата на диаграма карта на съответствията между термините от теоретична и практическа гледна точка.



Фиг. 45 Карта на съотвествие на теоретични и приложни термини

Работните задачи са основно средство за определяне и разпределение на работата между членовете на екипита. При адаптацията на метода на проследимост, базиран на събития, са определени следните два етапа:

- Настройване на средата за генериране на проследяващи събития.
- Прихващане на събития за осъществена промяна над обект и създаване на проследяващи връзки.

Процесът по настройване на средата представлява представлява ръчен процесс, който се състои от следните две стъпки:

1. Определяне на даден версионизиран обект като работна задача. Тази стъпка следва да се извършва от мениджъра на задачите за съответното ниво на детайлизация на задачата, или от самия инициатор на задачите в рамките на проекта.

2. Подготвяне на работното пространсто за автоматично генериране на проследяващи връзки. Същността на стъпката се състои в асоцииране (активирането) на небходимите работните задачи към работното пространство. В рамките на тази стъпка потребителят избира по кои работни задачи възнамерява да работи. Стъпката следва да се извършва от съответния участник в процеса по създаване на софтуерният продукт.

След изпълнението на втората стъпка системата е способна автоматично да прихваща събития по промяна на обектите и да създава проследяващи (причинно-следствени) връзки.

2.4.2. Модел на данните на система за управление на версията чрез йерархични пространства

Като резултат от представените до тук модели за управление на версии моделът може да се пристъпи към представяне на модела на данните. Моделът на данните ще бъде представен под формата на ER диаграма - формата, под която данните за обектите следва да се съхраняват.

При изграждането на модела е използван еволюционен подход, който се състои в изграждането и апробацията на малки изменения. Тези малки изменения следват посоката на търсене, формулиране, решаване и подобряване на научно-практическите задачи, които в настоящата дисертация се разглеждат. При развитието на модела на всяка стъпка е правен опит промените да засегнат в минимална степен сърцевината на модела, неговият принцип, а също така и неговото оптимизиране. Така например в първоначалния вариант на модел за работни задачи те са били изнесени като отделна същност. На по-късен етап същността се преработва до свойство на версионизирания обект. Това довежда до намаляване на релационните връзки в модела, което съответно води до повишаване на

неговата ефективност и ускоряване работата на програмната му реализация.

В модела (Фиг. 46) са включени следните основни същности: продукт, издание, работно пространство, версионизиран обект, версионизиран примитив, потребител. Моделът се допълва и от релационни същности като: дъга на версиите, причина, асоциирана работна задача, композитор, дъга на изданията. Следва да се отчете фактът, че с цел унифицираност на всички елементи на модела, във всички същности и релации, чиято уникалност се определя от съставен ключ, е използван сурогатен ключ [11].

Продуктът като същност има за цел да позволи разработването на различни софтуерни продукти, използвайки инстанция на едно и също информационно пространство. Изданията от своя страна представляват версия на продукта, която съответства на определен етап от неговото развитие. Фактът, че предлаганият продукт може да съществува като паралелни издания, довежда до необходимостта да се въведе релационната същност – дъга на изданията. Използването на тази същност ни позволява да се строи графова последователност на изданията (ацикличен, ориентиран граф) [43].

Всяка същност ***издание*** е свързана с един единствен екземпляр на същността ***работно пространство***, който не нарича главно работно пространство. В рамките на същността работно пространство е изградена вътрешна (рефлекторна) релация, която се използва за йерархичното им подреждане. С цел определяне на постигане на изолираност на работата, в ***работно пространство*** съществува релация към същността ***потребител***.

Същността ***версионизиран обект*** е тясно свързана със същността ***версионизиран примитив***, чрез обратна релационна връзка (екземплярите на същността ***версионизиран примитив*** указват към кой екземпляр на същността ***версионизиран обект*** са свързани). Екземпляри на

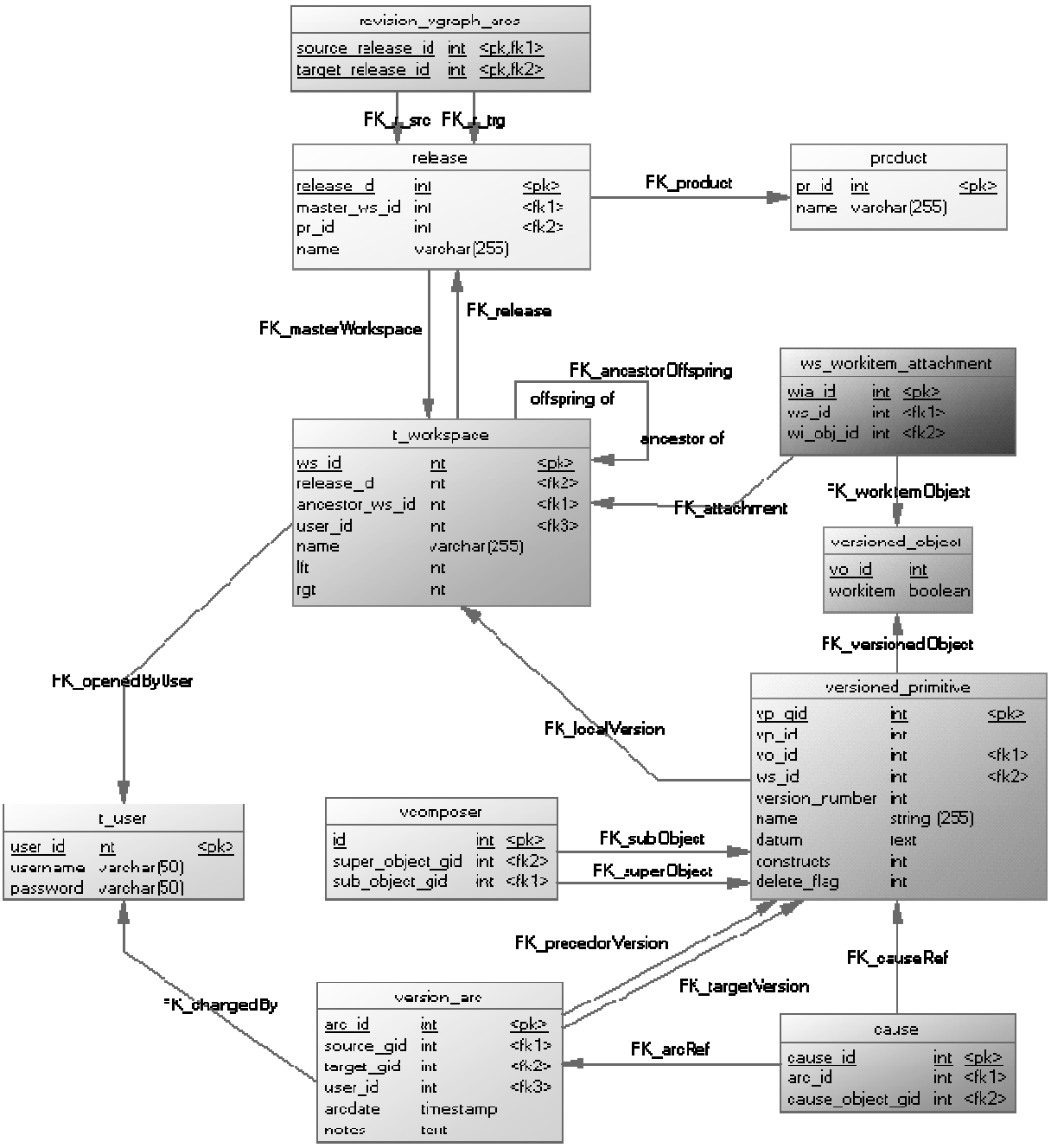
версионизиран обект, за които свойството „работна задача” има истинно значение могат да бъдат свързани към едно или повече ***работно пространство***, като за целта се използва релационната същност ***свързана работна задача***.

За да се определи локална версия на версионизиран обект, в същността ***версионизиран примитив*** е въведена релационна връзка към елемент на същността ***работно пространство***. Чрез тази релационна връзка се определя дали дадената версия на обекта се явява локална за съответното работно пространство.

За композирането на обекти е въведена релационна същност ***композитор***. С нейна помощ е възможно да се определи набор от под-обекти за избран супер-обект на ниво версионизиран примитив.

За нуждите на статистиката и одита на информацията е удачно да се използва инструментът на версионизирания граф. Неговото реализиране се осъществява чрез релационната същност ***дъга на версиите***. Тази същност притежава три релационни връзки – предхождаща версия, целева версия, потребител, извършил промяната. Всеки екземпляр на тази същност може да бъде указан от екземпляр на релационната същност ***причина***. ***Причината*** е въвеждена, за да се осигури проследимост на промените.

Детайлно описание на същностите и релациите е представено в Приложение 1 – Описание на модела на данните.



Фиг. 46 ER модел на данните

2.5. Методологична рамка за използване на разработените модели

Методологията се определя като „система от принципи и средства за организиране и провеждане на дадена дейност" [1, стр.449]. В настоящия параграф е направен опит да се представи методологично решение при активното използване на моделите и

средствата, разгледани в настоящата глава. Това предполага определянето на рамка на методологията.

В рамките на настоящата методологична рамка може да се разгледат следните моменти:

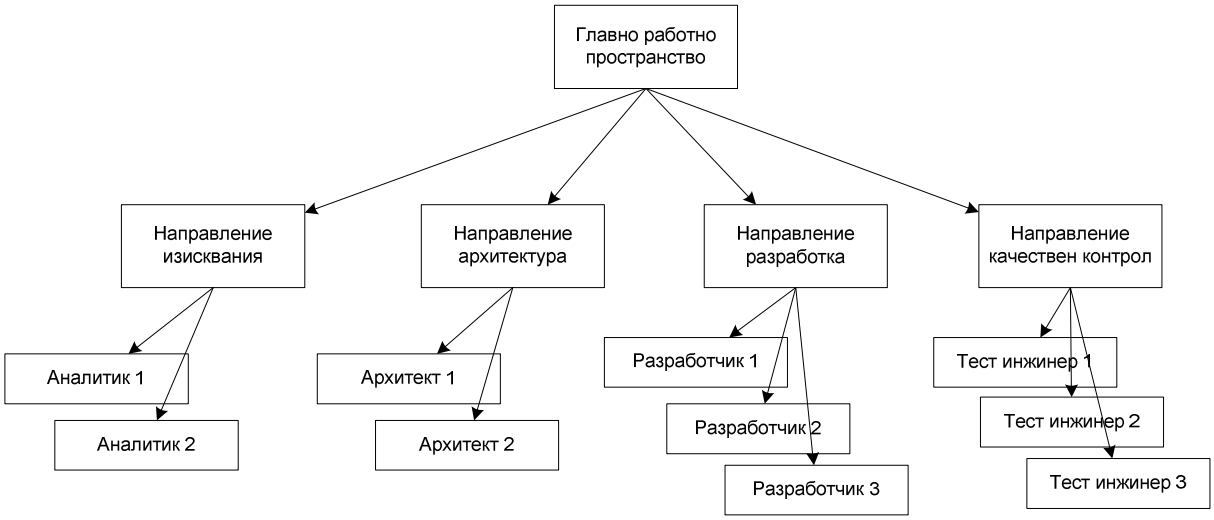
1. Подготовка на средата;
2. Създаване и определяне на задачите;
3. Изпълнение на задачите;
4. Публикуване на изпълнените задачи и сглобяване на крайния продукт.

Под подготовка на средата следва да се разбира процесът на определяне йерархичната архитектура от работни пространства, която съответства на избраната методология и подход на разработване. Тук се създава схемата, по която се организират работните пространства. На Фиг. 47 и Фиг. 48 са представени две примерни схеми на организация на работни пространства.

Тези диаграми демонстрират свободата на методологичната рамка, която тя предоставя при реализацията на проекти. За всеки отделен проект може да се организира самостоятелна схема на работни пространства, в зависимост от неговите особености и потребностите на потребителите.

Във всяка една методология за създаване на софтуерни продукти съществува етап на определяне на изискванията. В рамките на представената тук методология изискванията следва да се създадат под формата на версионизирани обекти. Това е породено от факта, че почти е невъзможно те да не се променят в рамките на целия жизнен цикъл на продукта. Изискванията, създадени във вид на версионизиран обект, позволяват да се проследи тяхното изменение, да се сравнят две техни версии, да се върнем към по-стара версия, както и да се намали риска от изгубване на знания. Като последна стъпка от създаването на изискванията е необходимо те да се отбележат като работни задачи. Този последен

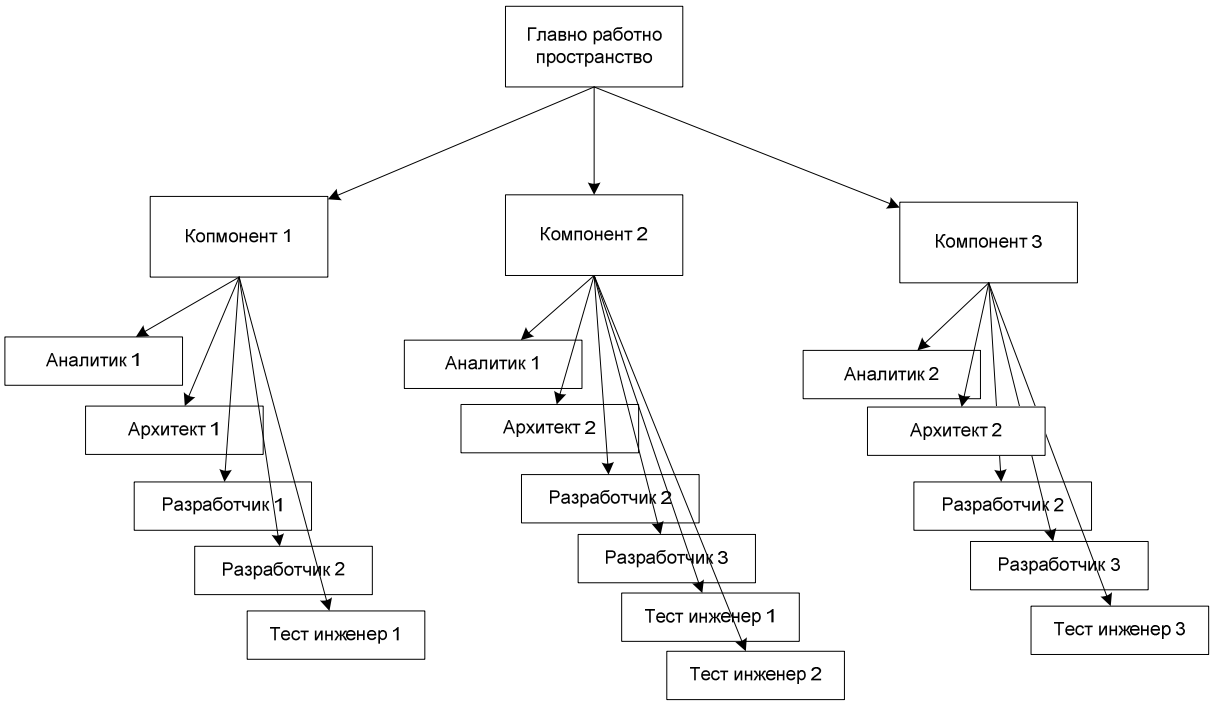
момент представлява основното свързващо звено със следващите компоненти от методологията.



Фиг. 47 Модел на организация на работата в пространства по направления

Под изпълнение на задачи следва да се разбира същинският процес на създаване на софтуерния продукт. Резултатът от изпълнението на една задача може да представлява последваща задача, която разглежда първоначалната в по-големи детайли, с по-голяма прецизност. Така например създаването на архитектура на софтуерния продукт, както и на тестовите сценарии, може да се разглежда като задачи, продиктувани от изискванията, чиито краен резултат представлява задача съответно за разработването на продукта, така и за провеждането на тестовете, гарантиращи качеството на крайния продукт.

Ако се разгледа задачата по създаването архитектурата на един продукт, за нея е необходимо да се избере изискване (работна задача). След създаването на архитектурата като версионизиран обект, тук разглежданата методологгия изисква тя да се отбележи като работна задача. Забелязва се особеността, че компонентите 2 и 3 от методологията в този случай са в обърнат ред.



Фиг. 48 Модел на организация на работата в пространства по компоненти

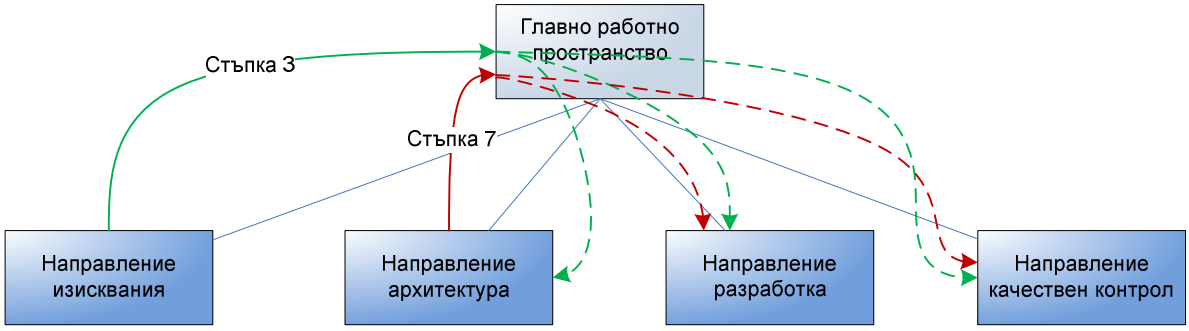
Публикуването следва да се разглежда като средство за интегриране на отделните компоненти на продукта. От модела на видимост на обектите (представен по-горе в текущата глава) следва, че публикуването на обект в по-горно работно пространство води неговата видимост в сестринските работни пространства. Тук под сестрински работни пространства се разбира тези работни пространства, които се явяват дъщерни работни пространства на родителското работно пространство. Именно публикуването представлява механизъм за споделяне обектите, съответно и на сглобяване на крайната версия на продукта. Когато едно изискване се одобри, т.е. по него е достигнат консенсус между учасниците в проекта, то може да се публикува в главното работно пространство. Така то става видимо за всички участници в проекта. Архитектите, инженерите по качество имат възможност да създадат своите артефакти, указвайки като причина за тяхното появяване новото изискване.

2.5.1. Процес на създаване на нова функционалност

В настоящия пункт се демонстрира процесът на създаване на нова функционалност към нова или съществуваща софтуерна система. Тук се използват елементите от методологичната рамка, представени по-горе. За улеснение нека се приеме, че в проекта се използва опростена схема на организация на работните пространств по направления, която е представена на Фиг. 49.

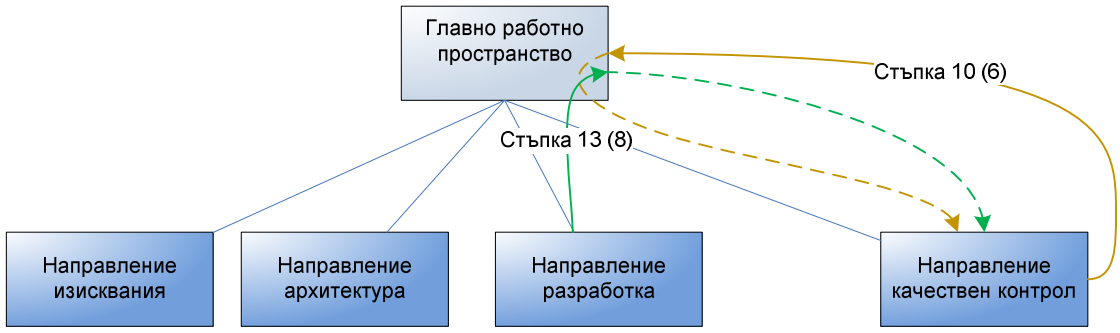
Както е прието в практиката, процесът на създаване на нова функционалност се състои от следните стъпки:

1. Определяне на изискванията. В представената схема те следва да се създадат в работното пространство на Аналитика.
2. След тяхното съгласуване те се маркират като завършени и могат да се избират за работни задачи
3. Новите изисквания се публикуват до главното работно пространство, където стават видими за останалите участници в процеса.
4. Архитектът в проекта избира като работна задача за своето работно пространство новото изискване.
5. Той създава архитектура на системата, чиято реализация съответства на изискването.
6. След завършване на процеса по създаване на архитектурата, новият артефакт се отбелязва, като работна задача.
7. Архитектурата на системата се публикува до главното работно пространство, където тя става видима за останалите участници в системата
8. Тестовият инженер отбелязва в своето работно пространство като работни задачи, над които ще се работи изискването (видимо от стъпка 3), и архитектурата на системата от предишната стъпка (зелената и червената пунктирани линии на Фиг. 49).



Фиг. 49 Етапи по създаване на изсквания и архитектура

9. Той създава тестови план и тестови сценарии за проверка на качеството на бъдещата системна функция.
10. След приключване на работата по тестовия план и сценариите, те се отбелязват като работни задачи и се публикуват в главното работно пространство, където стават видими за останалите участници в процеса.
11. Разработчиците отбелязват като работна задача, над която ще се работи в своите пространства тези на изискването (от стъпка 3) и архитектурата (от стъпка 7).
12. Разработчиците реализират новата функционалност на системата.
13. След завършване фазата на разработване, артефактите на новата функционалност (изходен код, документация и др.) се публикуват до главното работно пространство, където те стават видими за всички участници в процеса.

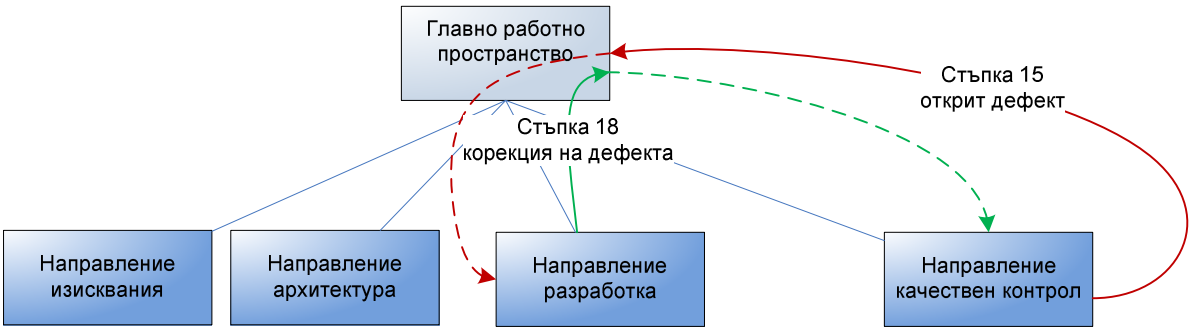


Фиг. 50 Етапи по създаване на изходен код и тестови сценарии

14. Тестовият инженер отбелязва в своето работно пространство като работна задача тестовите сценарии от стъпка 10 и започва да

проверява качеството на продукта (жълтата и зелената пунктирани стрелки от Фиг. 50).

15. При откриване на дефект тестовият инженер създава нов обект за дефект, отбелязва го като работна задача и го публикува до главното работно пространство.
16. Всички участници в процеса проверяват постановката на тестовия сценарий и дефекта и подтвърждават неговата правилност. За улеснение в конкретния случай, нека се приеме, че дефектът описва несъответствие между реализираната функционалност и първоначалните изисквания.
17. Разработчикът отбелязва дефектът като работна задача, над която ще работи.
18. Той разработва поправка и я публикува до главното работно пространство.
19. Тестовият инженер извършва повторна проверка на функционалността и потвърждава, че дефектът е отстранен.



Фиг. 51 Стъпки по откриване и отстраняване на дефект

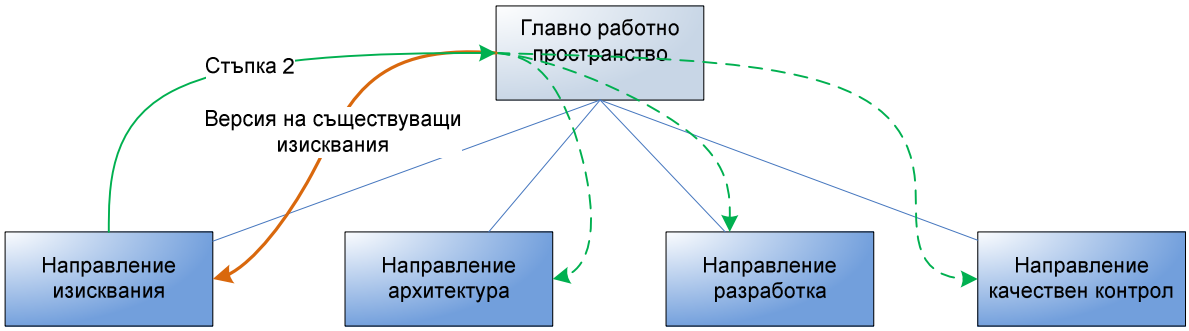
2.5.2. Процес на промяна на съществуваща функционалност

В пункта се демонстрира процесът на промяна на съществуваща функционалност на дадена система чрез използване на елементи от представената методологична рамка.

1. Аналитикът определя съществуващите изисквания, които следва да се променят, и ги редактира. Следва да се отбележи, че исикванията

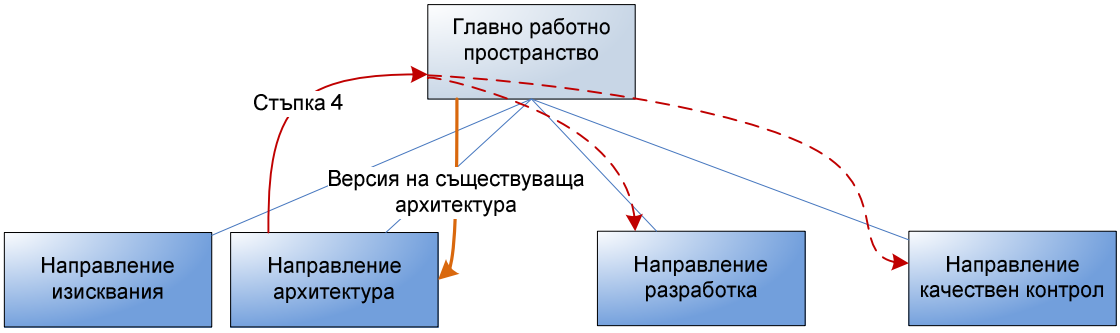
притежават свойството „работна задача” от предишната итерация (тази на създаване или на промяна).

2. След тяхното съгласуване те се маркират като завършени и могат да се публикуват до главното работно пространство, където стават видими за останалите участници в процеса.



Фиг. 52 Стъпки 1 и 2 по определяне промяната в съществуващи изисквания

3. Архитектът в проекта извършва анализ за съответствие на версиите между обектите на архитектурата и изискванията. В справката за несъответствие на причинно-следствените връзки се отбелязва, че текущата версия на някои архитектурни обекти не съответства на текущата версия на изискванията. Архитектът променя архитектурните обекти, така че те да съответстват на променените изисквания.
4. Обстоятелството, че архитектурните обекти в своята предишна версия имат свойството на работна задача, се наследява и в тяхната нова версия. Ако на предишната стъпка са били създадени нови архитектурни елементи, те следва да се отбележат като „работна задача”. Променените архитектурни обекти се публикуват до главното работно пространство, където стават видими за останалите участници в процеса.



Фиг. 53 Стъпки 3 и 4 по промяна в съществуващата архитектура

5. Тестовият инженер, използвайки справката за несъответствие на причинно-следствените връзки, определя в кои тестови сценарии той следва да извърши корекции. Променените тестови сценарии наследяват свойството „работна задача" от предишната си версия. За новите тестови сценарии тестовият инженер добавя свойството „работна задача".
6. Променените тестови сценарии се публикуват до главното работно пространство.
7. Разработчикът, използвайки справката за несъответствие на причинно-следствените връзки, определя в кои файлове от изходния код следва да извърши корекции.
8. Променените обекти на изходен код се публикуват до главното работно пространство.

Следва да се изтъкне фактът, че за представянето на стъпки от 5 до 6 може да се използва диаграмтата от Фиг. 50, където в скоби са посочени номерата на стъпките от по-горе описания процес. Осигуряването на качествен контрол при промяна на съществуваща функционалност е идентично със стъпки 14 – 19 от предишния пункт (процес на създаване на нова функционалност).

2.6. Изводи

От направените теоретични разработки в настоящата глава могат да бъдат направени следните изводи:

1. Предложен е гъвкав модел на версионизиран обект, предоставящ възможността свободно да се определя нивото на гранулираност на обектите. Това предполага неговата приложимост при различни практически задачи, за които е необходимо по-гъвкаво ниво на абстракции, отколкото може да се достигне при използването на файлове. Теоретичният модел е докладван на международната конференция „Central & Eastern European Software Engineering Conference", Москва (2009). Концептуалният ER модел е представен в Научна конференция с международно участие "25 Години Педагогически Факултет", Велико Търново (2009).
2. Разработен е модел на среда с йерархично композиране на работни пространства, включващ модел на данните за тази среда. Определени са теоретичните правила, както и транзакциите над версионизирани обекти. Използвайки модела на транзакциите, е направен опит за създаване на модел на жизнен цикъл на версионизиран обект. Получените резултати бяха докладвани на Международната конференция „Electronics, Computers and Artificial Intelligence" в Питещи, Румъния (2010).
3. Предложена е адаптация на метод за проследимост на промените, базиран на събития. При адаптацията на метода се използва композираността на пространствата и на обектите като механизъм за разбиване на големи задачи на по-малки и тяхното решаване. Това позволява в пълнота да се обхванат обектите и връзките между тях в процеса на създаване на софтуерни продукти. Резултатите от изследването са докладвани на Международната конференция „Автоматика и информатика'10" в София (2010).
4. Предложена е българска адаптация на терминологията в областта на управлението на версии. Съвместно с адаптацията в настоящото научно-приложно изследване са представени нови научни термини

и понятия в областта на управлението на версии, чрез използването на йерархично композирани работни пространства, като са въведени 2 принципа, 13 дефиниции, 5 правила и 5 следствия. В Приложение 2 е представена онтологична диаграма на понятията в предметната област.

5. Представена е методологична рамка за използване на тук разработените модели и методи в практиката. Методологичната рамка е разработена във формата на стандартни работни процеси. Разгледани са конкретни теоретични примери на използването им в процесите на създаване и на поддръжка на софтуерни продукти.

3. Глава трета

Изследване на 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 могат да се споменат са: автоматично дописване на код, възможности за постъпково изпълнение, анализ на производителността на изходния код, средства за интерактивно моделиране.

В втора глава са представени модели за версионизиране на обекти в среда с йерархично композиране на работни пространств. Целта е те да преминат експериментална апробация за увеличаване на ефективността на работния процес при създаването на софтуерни продукти, чрез увеличаване степента на неговата автоматизация.

3.1. Възможности за реализиране на моделите

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 избора на платформа и среда за разработка [6, 7, 8, 30, 39, 40, 47] бяха взети предвид следните критерии:

- Познаване на езика – 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 на възможностите на библиотеките 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 са разгледани PHP [30], JavaEE [47] и C# DotNet [97]. 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**
JavaEE	6	6	5	5	5	27
C# .Net	3	6	6	5	3	23
PHP	6	4	4	4	6	24

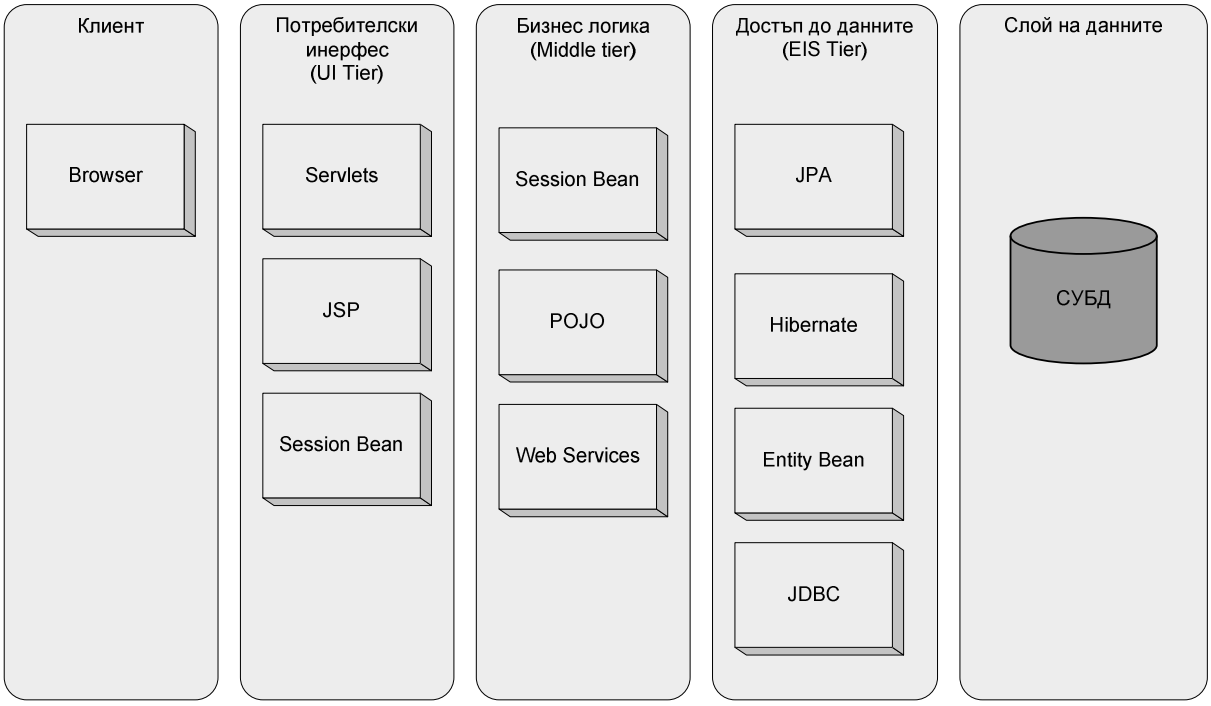
таблица 2 Рейтинг за избор на платформа

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

Технологиите, позволяващи да се ускори работата в J2EE платформата, могат да се обособят 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 потребителския интерфейс и бизнес логика е направена оценка съгласно следните основни критерии:

- Удобство за използване на технологията.



Фиг. 54 Архитектура на J2EE платформата

- 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.
- Познаване на технологията – субективен критерий, който има тежест най-вече при началния етап на разработването.

Като технология на представяне и контролери са разгледани и анализирани [7, 10, 19, 107, 33] следните: Servlet/JSP, Struts, JSF, Spring, като резултатът от направения анализ е представен в таблица 3. В своето изследване Барцис [19] определя JSF технологията като по-гъвкава и разширяема в сравнение с технологичната рамка Struts.

	Удобство	**Скорост**	**Гъвкавост**	**Познаване**	**Рейтинг**
Servlet/JSP	2	2	3	6	13

Struts	4	4	4	2	14
JSF	5	5	6	6	23
Spring	5	5	6	2	18

таблица 3 Рейтинг за избор на UI J2EE технология

След избора на платформа и работна рамка за провеждане разработката на прототипа на системата за управление на версии, възниква въпросът за бързото разработване на достъп до данни. Както е известно, в Java платформата е въведен JDBC от Sun Microsystems през 1997 година [87], като стандартен SQL достъп до релационни бази данни. През 2001 в J2EE се въвежда Entity Beans (EJB2.0) [91], която предоставя стандартен механизъм за достъп до данните. През същата година Гавин Кинг (Gavin King) започва разработването на библиотеката Hibernate [10, 58], като алтернатива на Entity Beans за ORM достъп до данните. От 2006 година с въвеждането на Java Persistence API (JPA 1.0) [92] ORM механизмът за достъп до данни става част от JavaEE платформата, като през 2009 той напълно измества EJB 2.0 и EJB 2.1. Самото JPA изисква използването на провайдер, който го имплементира. Развитието на технологиите именно в тази посока предопределя и избора на технология за достъп до данните. При разработването на прототипа е използвана библиотеката Apache OpenJPA [15], като за неговия избор е използван рейтинга [58, 70], представен в таблица 4.

	Удобство	**Скорост**	**Гъвкавост**	**Познаване**	**Рейтинг**
JDBC	2	6	2	6	16
JPA (OpenJPA)	6	5	5	6	22
Hibernate	6	5	6	4	21

таблица 4 Рейтинг за избор на EIS J2EE технология

Фактът, че JSF като архитектурен елемен предоставя само рамка, но не и стандартна реалирзация предполага да се направи формален избор на неговата реализация. Това е необходимо условие при успешната реализация на функционалния прототип. От направения преглед на продуктите на доставчиците на свободни JSF имплементации са взети под внимание следните библиотеки: Apache MyFaces, Jboss Richfaces, ICEsoft IceFaces и TeamDev OpenFaces. Резултатът от анализа на предимствата и недостатъците на библиотеките [51, 59, 93, 104] е обобщен в таблица 5.

	Удобство	**Гъвкавост**	**Поддръжка**	**Познаване**	**Рейтинг**
Apache MyFaces	4	5	6	4	19
Jboss Richfaces	6	5	6	6	23
ICEsoft IceFaces	4	4	4	4	16
TeamDev OpenFaces	4	5	6	4	19

таблица 5. Рейтинг за избор на JSF имплементазионна библиотека

Следващият основен компонент при разработването на прототипа представлява платформата за бази данни. При избора на подходяща платформа се използвани следните показатели:

- Личен опит от използването на съответната платформа.
- Лекота и скорост на разработка, която предоставят инструментите към съответната платформа.
- Функционални възможности на базата.
- Производителност.
- Интеграция с избраната платформа за разработка (Java & JPA).
- Възможност за използване при демонстрация на продукта.

При попълване на рейтинговата таблица са разгледани следните популярни СУБД: Oracle 10g, Microsoft SQL Server 2008, MySQL 5.3. При построяването на индексите за функционалноите възможности на системи

е използван анализът в [16, 95, 98, 105], където до голяма степен Oracle като платформа предоставя по-големи възможности от системите на Microsoft и MySQL.

	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

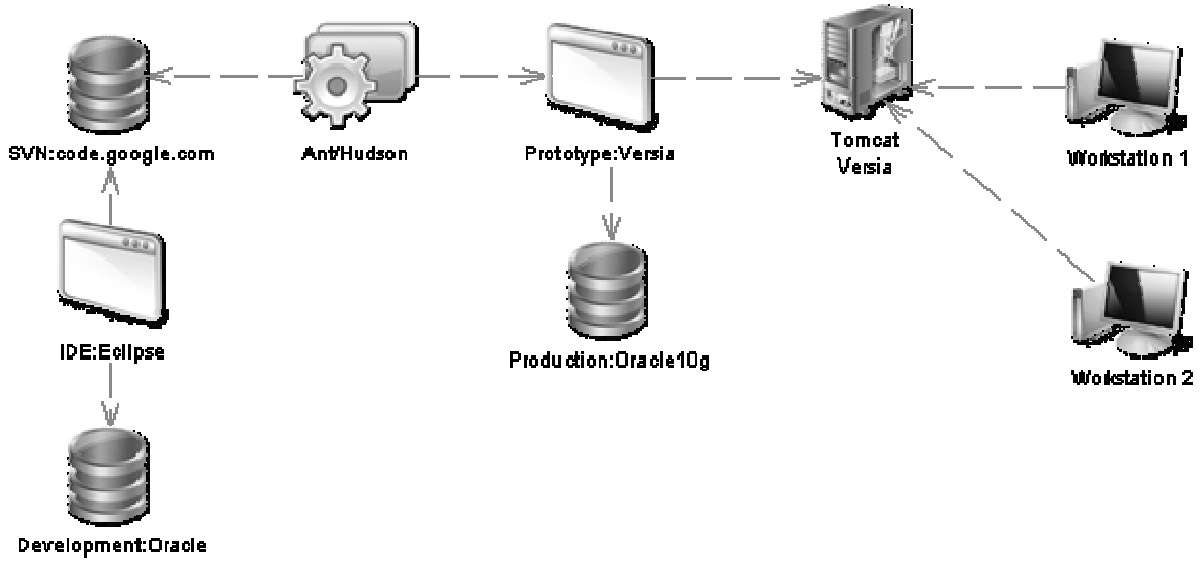
таблица 6. Рейтинг на СУБД

3.2. Разработка на прототип, реализиращ представените модели

3.2.1. Избор на софтуерен инструментариум и определяне процеса 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 разгръщане на приложението е използван Ant [81] сценарий, който са автоматично се изпълнявани от системата Hudson [45]. 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. 55 е представена схема

на цялостния процес: от разработването, през разгръщането, до използването на системата от крайните потребители.



Фиг. 55 Процес на разработка, разгръщане и използване на прототипа

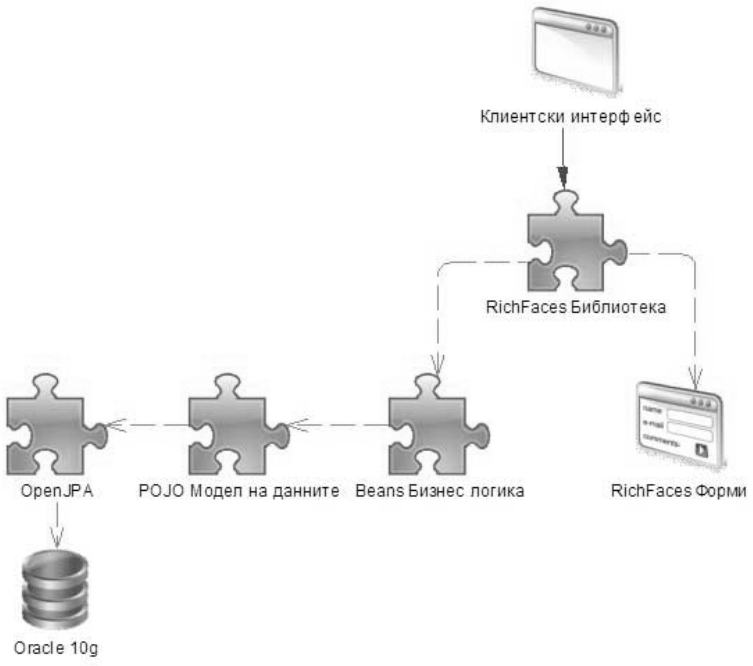
3.2.2. Архитектурен модел

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

- Слой на базата данни.
- 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-ниския слой – слоят на базата от данни, като това се извършва чрез промяна в конфигурационния файл на библиотеката.

  Следва да се отбележи важна характеристика на библиотеката OpenJPA – тя позволява 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 продукционната среда.



Фиг. 56 Архитектура на система-прототип Versia

- 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 продиктувана от факта, че всички Java запазващи (persistent) библиотеки изискват предметната област да е реализирана именно чрез такива обекти.
- Слой на бизнес логиката. Реализацията на бизнес логиката е извършена 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 готовия 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 Асинхронните JavaScript и XML заявки (AJAX).

3.2.3. Архитектурна организация на класовете

От архитектурна гледна точка класовете, реализиращи прототипа на системата, са организирани в пакети съгласно тяхното предназначение. При именуването на пакетите е използвана общоприетият java стил: инвертиран домейн префикс, наименование на продукта, модул:

- com.jotov.versia.beans – в този пакет са разположени общи бизнес-навигационни компоненти, предоставящи базовата инфраструктура на приложението. Като основни направления, реализирани от компонентите, могат да се споменат следните: оторизан достъп до системата, инфраструктура по осигуряване на достъп до данните, инфраструктура по синхронизацията между отделните сесии, компоненти, предоставящи стандартна бизнес функционалност.
- com.jotov.versia.beans.vobj – в този пакет може да се открият специализирани компоненти, реализиращи функционалността

по осигуряване достъпа до версионизираните обекти, техните версии и история, видимост, справочна и одит информация.

- com.jotov.versia.orm – пакетът съдържа POJO класове, използвани от JPA библиотеката (OpenJPA) за достъп до данните в базата. Класовете в пакета до голяма степен повтарят модела, представен в 2.1, 2.2 и 2.4.
- com.jotov.versia.utils – в този пакет е разположен инфраструктурен компонет, осигуряващ бърз достъп до информацията на версионизираните обекти в рамките на конкретно работно пространство.

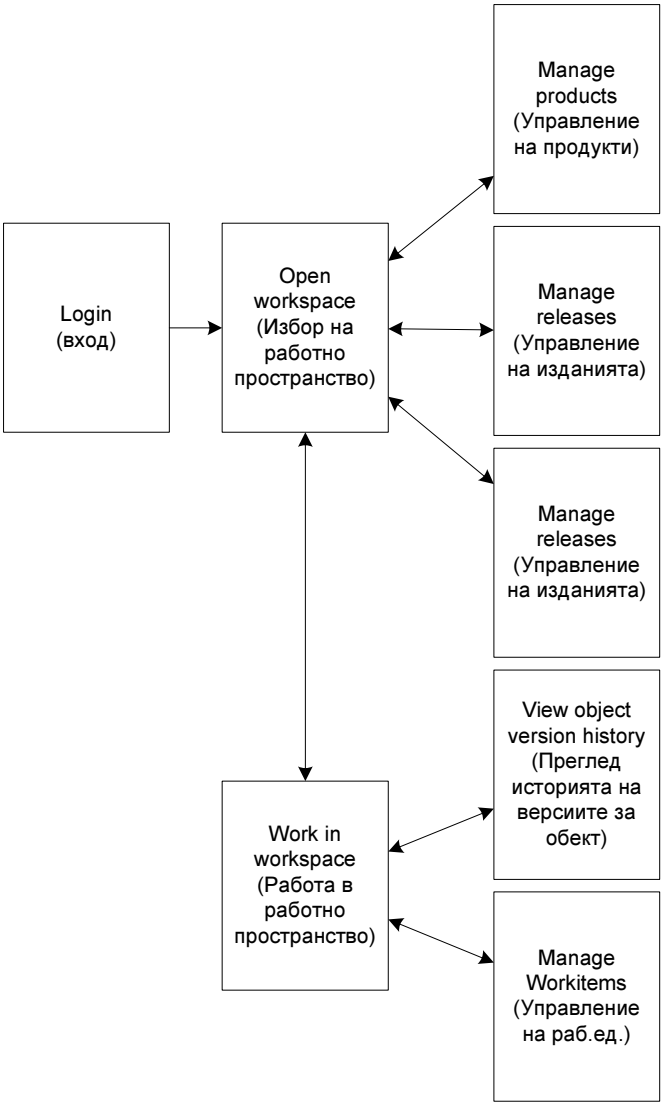
3.2.4. Навигационен модел и потребителски интерфейс

Основната задача при разработването на прототипа е тази за реализирането на миималния функционал, необходим за нормалната работа на разработените модели. Това е тясно свързано и с навигационния модел на системата. Навигационният модел представлява модел, който ни помага да разберем пътя и логиката, реализирани в системата, за достигането да определена функционална точка. Тук са представени и основните екрани на прототипа.

На Фиг. 57 е представен навигационният модел на системата, където с правоъгълници са представени страниците на системата, а със стрелки – възможните преходи между страниците. Моделът се състои от следните страници:

1. Вход в системата.
2. Избор на работно пространство.
3. Управление на продуктите.
4. Управление на изданията.
5. Управление на пространствата.
6. Страница за работа в избрано пространство.
7. Управление на работните задачи в избраното пространство.

8. Преглед на измененията за избран обект.



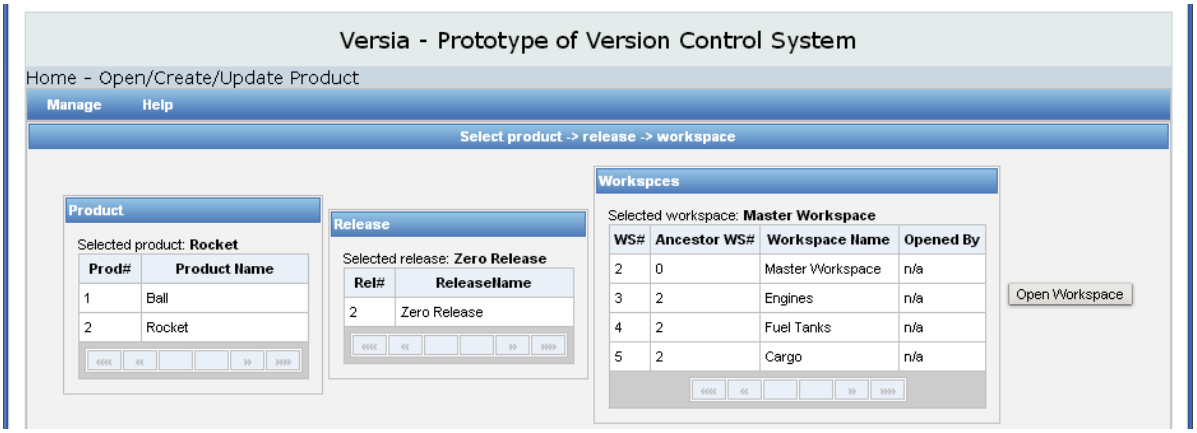
Фиг. 57 Навигационна диаграма на прототипа

На фигурата по-долу е показан екрана за избор на работно пространство. След избор на продукт, системата зарежда всички негови издания, а след избор на издание – всички работни пространства, асоциирани с избраното издание. На потребителя се предоставя информация относно йерархичното разположение на всяко работно пространство.

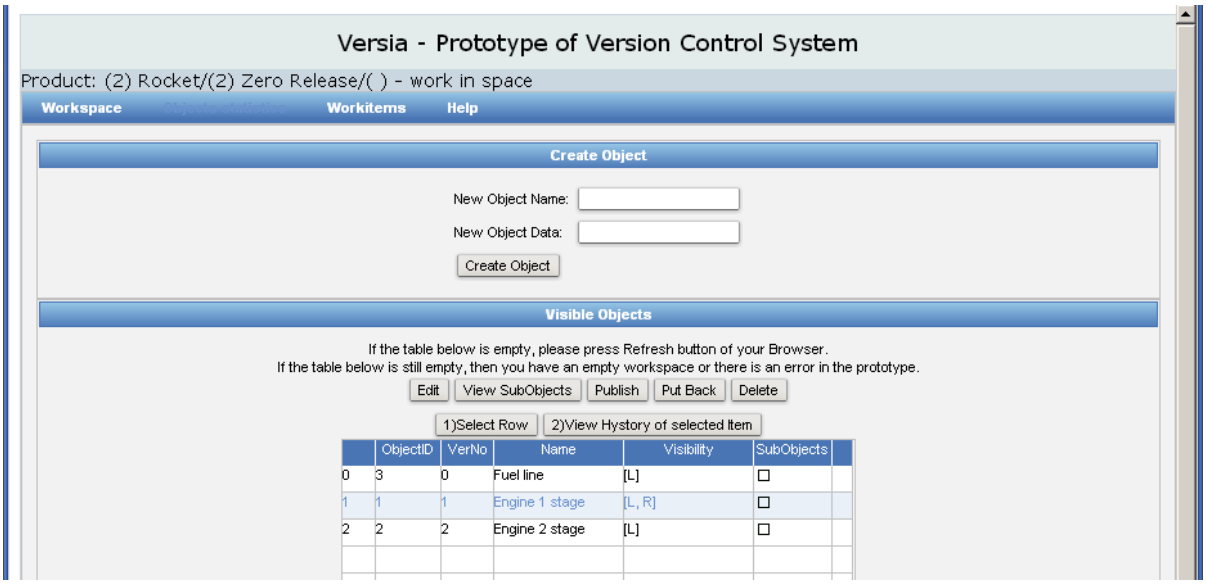
След като се избере и се отвори дадено работно пространство системата зарежда екранът на работното пространство - Фиг. 59. От тук потребителят може да извършва манипулации над обектите. Системата предоставя информация относно разположението на версиите на всеки

обект, който е видим в пространството. Информацията се показва в колона Visibility, като се използват следните съкръщения:

- R (Release) – версия на обект съществува в главното работно пространство.
- P (Parent) – съществува версия в родителско пространство.
- L (Local) – версията, която се вижда е локална.
- C (Child) – съществува версия в дъщерно пространство.
- O (Other) – съществува версия в друг клон от йерархията на пространства.
- D (Deleted) – видимата версия представлява маркер на изтрито състояние.

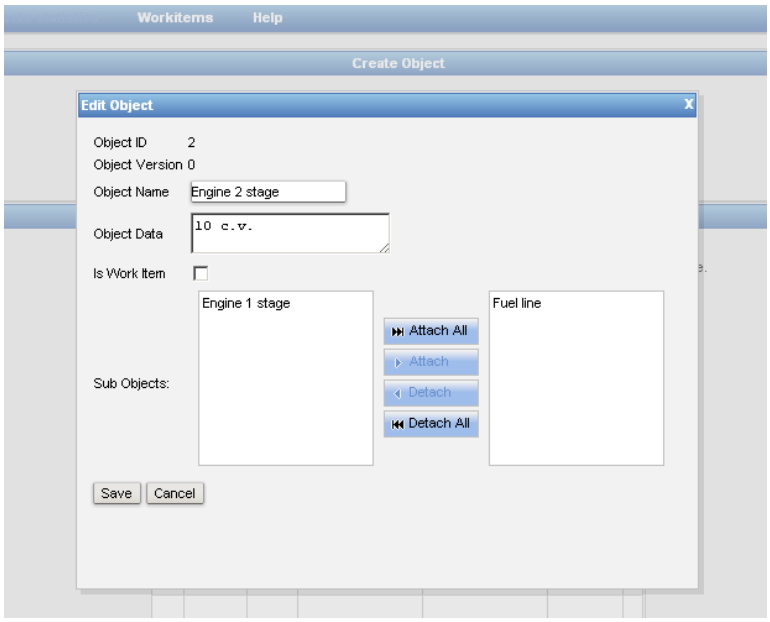


Фиг. 58 Екран за избор на продукт, издание и работно пространство



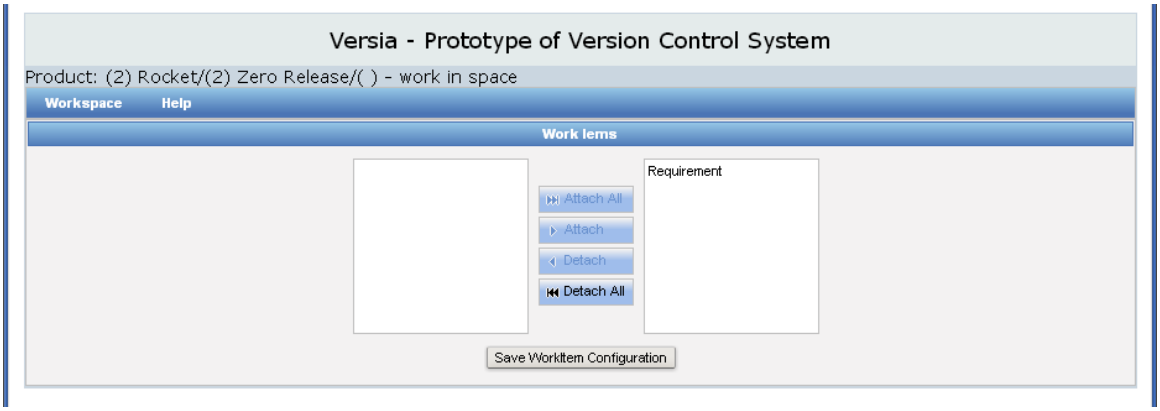
Фиг. 59 Версионизирани обекти в работно пространство

При редактиране на обект (Фиг. 60), той може да се отбележи като работна задача. Пак там се предоставя възможност да се добавят или премахнат под-обекти.



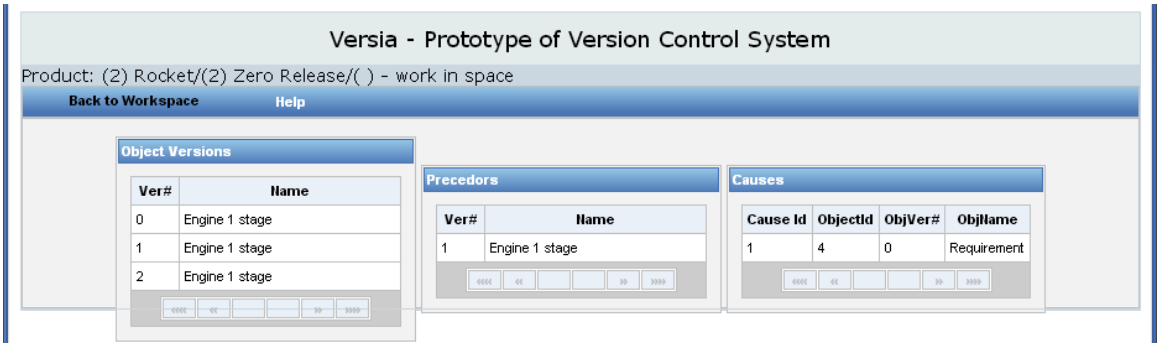
Фиг. 60 Екран за редактиране на обект

На екрана за настройване на активните работни задачи потребителят има възможност да определи над кои задачи работи.



Фиг. 61 Екран за настройване на активните работни задачи

На Фиг. 62 потребителят има възможност да проследи историята на един обект. Там той вижда версиите, предшестващи избраната, както и кои обекти – работни задачи са били активни в момента на промяната.

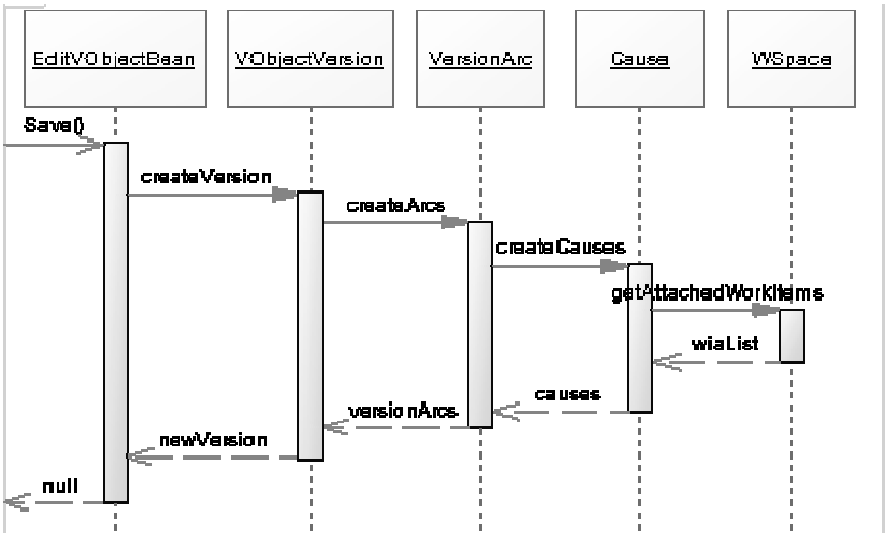


Фиг. 62 Екран показваш историята на един обект и причините за неговата промяна

3.2.5. Алгоритми при реализацията на прототипа

Изграждането на жив прототип на система представлява интересно предизвикателство. Реализацията на алгоритмичната основа, в която се реализират разработените в предишната глава модели и методи, предоставя богато творческо поле.

3.2.5.1. Създаване на нова версия на обект

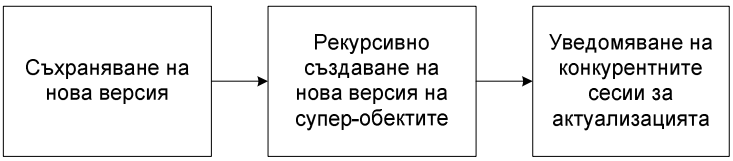


Фиг. 63 Диаграма на последователностите - създаване на нова версия

Основният алгоритъм е изобразен във вид на диаграма на последователностите на Фиг. 63. Алгоритмичната последователност се състои от следните стъпки: (1) съхраняване новата версия; (2) построяване на ребро в графа на версиите за обекта, което ребро свързва предишната с новата версия; (3) създаване на причините за настъпилата промяна, които

представляват (4) асоциираните с текущото работно пространство работни задачи.

Така описаната алгоритмична последователност следва да се изпълни рекурсивно над всички супер-обекти на конкретния обект. При приключване процесът на рекурсивно създаване на нова версия на супер-обектите е необходимо да се уведомят останалите потребители на системата, с цел те да презаредят (ако е необходимо) новата версия на обекта.

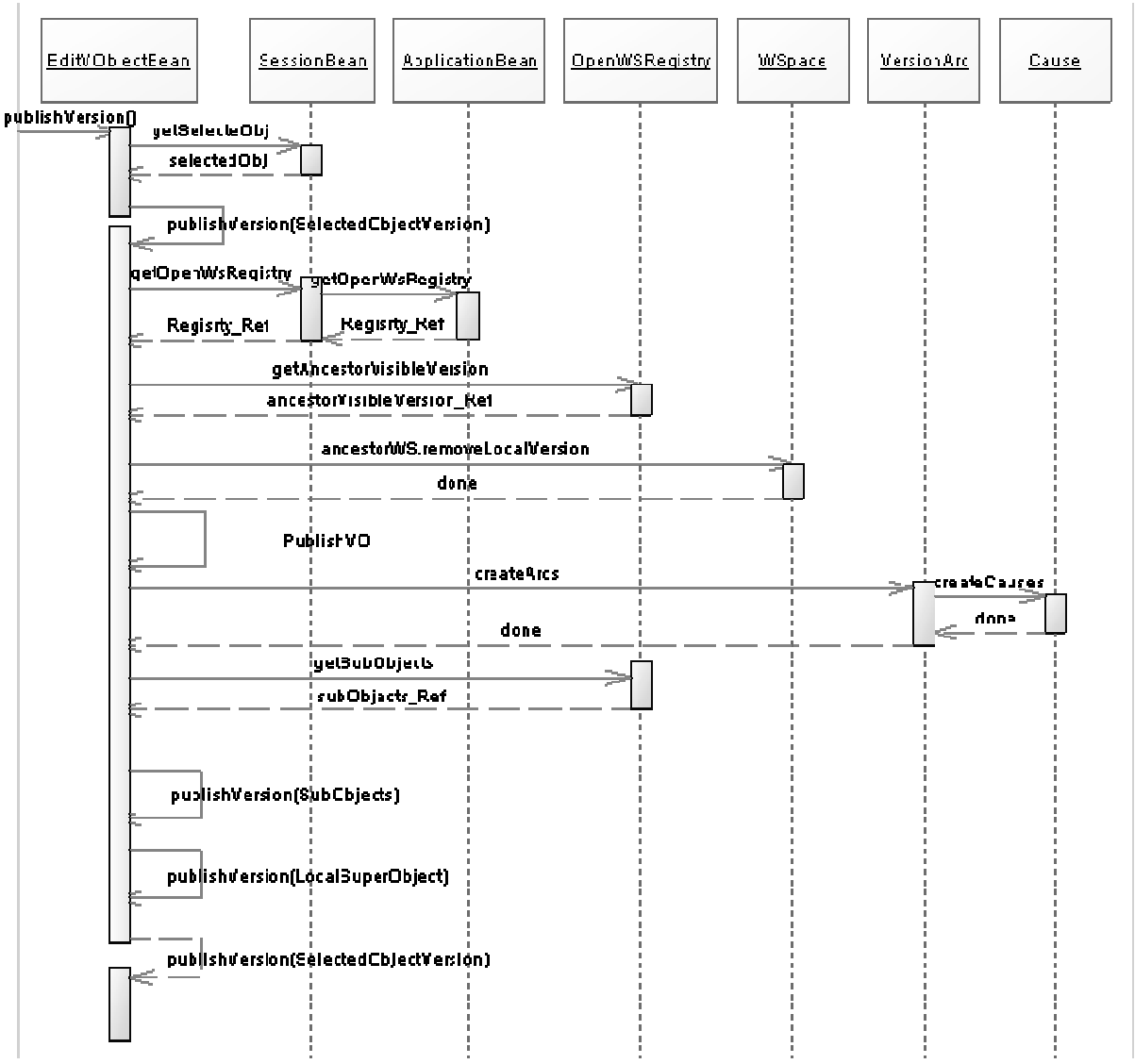


Фиг. 64 Създаване на нова версия на обект и неговите супер-обекти

3.2.5.2. Публикуване на версионизиран обект

Публикуването на версионизиран обект в родителско работно пространство представлява второто, но не по важност, алгоритмично предизвикателство. Процесът на публикуване на локална версия на обект се състои от следните стъпки (Фиг. 65):

1. Определяне на видимата версия в родителското работно пространство (getAncestorVisibleVersion), чрез използването на регистратурата на отворените работни пространства. На тази стъпка се проверява дали тази версия не се явява локална за родителското работно пространство.
2. Публикуване версията на локалната версия в родителското работно пространство. На Фиг. 65 е демонстриран вариантът, при който в родителското работно пространство съществува локална версия на публикувания обект. Това поражда създаването на проследяващи връзки от текугата негова версия към новата, а също така и причинно-следствени връзки.



Фиг. 65 Диаграма на последователностите - публикуване на версия

3. Алгоритъмът се изпълнява рекурсивно надолу за всички под-обекти на избрания обект. Това е продиктувано от Правило 3.
4. Като последна стъпка от алгоритъма, следва да се спомене необходимостта от публикуването на супер-обекта за избрания обект, породено от необходимостта да се спази Правило 4.

3.2.5.3. Регистър на отворените работни пространства

Определянето на видимите и локалните обекти за дадено работно пространство представлява доста тежка изчислителна задача, изискваща рекурсивно обхождане на родителските работни пространства. В търсене

на ефективен способ за повишаване производителността на системата са взети предвид следните обстоятелства:

- Комплектът от видими и локални обекти има смисъл да се определя само за тези работни пространства, в които потребителите на системата работят.
- Създаването на нова версия на обект, както и на публикуването на версия могат да повлияят на колекциите от видими и локални обекти за тези работни пространства, които се явяват наследници на пространството, в което се извършва промяната, респективно публикацията.

С оглед на горните обстоятелства е направен извод за целесъобразността от въвеждане и използване на шаблон «Регистър на отворените работни пространства» (OpenWSRegistry). Обектът на регистъра е реализиран по шаблона Singleton, т.е. той представлява единствена инстанция в работещата система. С цел постигане на конкурентен синхрон между потребителите на системата, всички методи на регистъра са определени като синхронни.

3.3. Примерни модели за композиране на версионизирани обекти

В настоящия параграф се демонстрират възможностите на ER модела на версионизиран обект, като се акцентира на подобренията в сравнение с файловия модел за версионизиране [68, 69]. В резултат на извършения анализ на недостатъците на файловия модел, тук са представени модели на версионизирани обекти, който предоставя възможности за свободно определяне на степента на неговата гранулираност. Това предоставя възможност за значително подобряване на версионизирането както в областта на обектно-ориентираното програмиране, управлението на качеството на софтуера и управлението на изискванията, така и на свързаността между елементите.

3.3.1. Примерен модел на същността „клас”

Въпреки насочеността на настоящия научен труд към създаването на софтуерни продукти, тук представеният модел може да претенира за универсалност и приложимост и в други области на човешката дейност, които се нуждаят от контрол на версията над дадени артефакти. На Фиг. 66 сме представили диаграма на композиция на версионизирани обекти, съставящи същността клас.

Определяща се явява вътрешна и външна структура на класа. Тук под външна структура се подразбира тази, която отговаря за взаимодействието с външния свят – други класове, които са необходими за нормалната работа на класа, или на които той влияе. Вътрешната структура от своя страна определя вътрешното поведение и взаимодействие на класа. Като базови елементи на класа може да отбележат полетата и методите.

Като стартова точка при изграждането на модела са определени метаатрибути на класа. Един клас се идентифицира уникално чрез своето *име* и *пакет*, в който той се намира. Като *тип* на класа сме посочили абстрактен, интерфейс, нормален клас, финален клас и вътрешен клас, като за вътрешния клас имаме допълнителен атрибут – *външен клас*. Някои езици за програмиране, като C++ позволяват да се наследява повече от един клас [9], докато други, подобно на Java [3], реализират принципа на полиморфизма чрез използването на интерфейси. Това довежда до идентифицирането на следните два метаатрибути 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 дадения клас, това е последният метаатрибут, който идентифицираме на ниво клас – *списък с декларираните за употреба класове*.

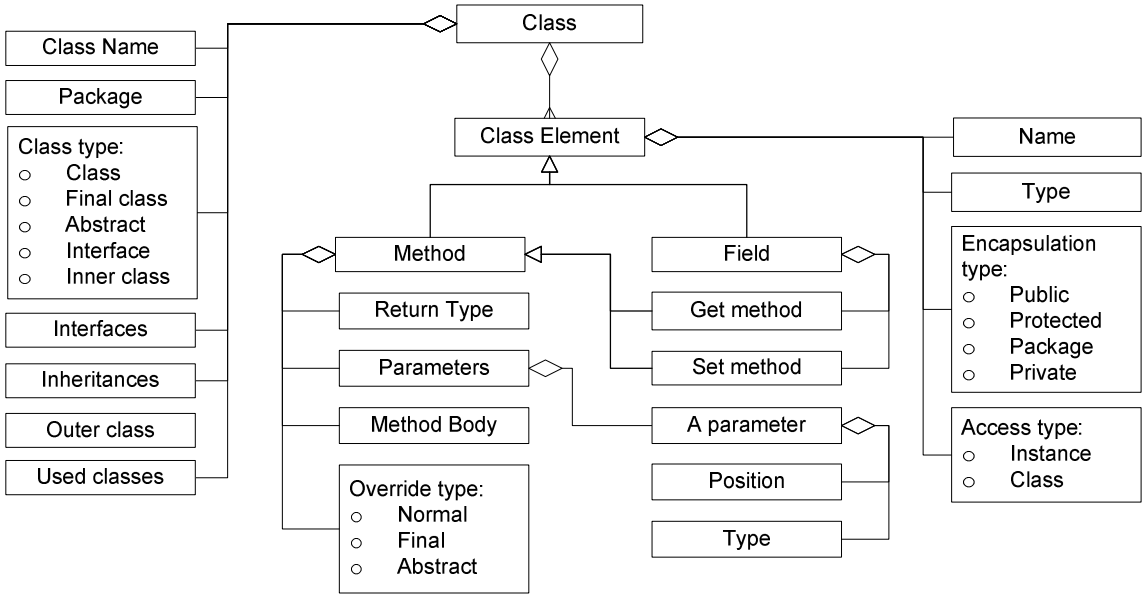
Полетата и методите на класа съдържат следните общи метаатрибути: *име*, *тип*, *тип* на капсулираност, *тип* на достъпа. Типът на достъпа определя начинът за достъп до съответния елемент, той може да бъде през инстанция или през самият клас. Типа на капсулираност определя нивото на достъп до съответният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

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

За същността *метод* може да посочат следните метаатрибути: 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*, се определя 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#, се забелязва еволюционно развитие на същността поле. Новото тук е появата на специализирани методи, познати като *извличащ метод* (getters) и *записващ метод* (setters). Те позволяват да се дефинира определено поведение на класа при опит за достъп до полетата му. Фактът, че тези методи споделят общ интерфейс позволява да ги причислим към същността *поле*.

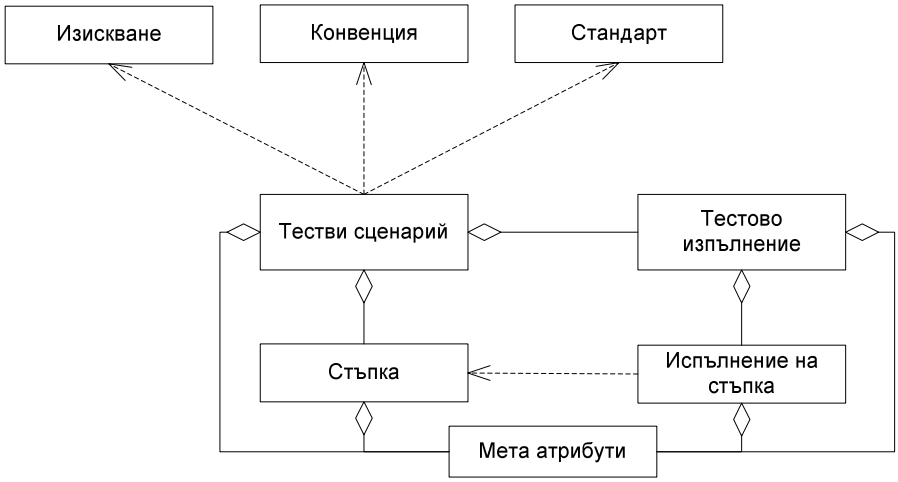
Така построеният модел използва в пълна степен предимствата на фината гранулираност на версионизираните обекти, което се явява основен недостатък при файлово-базираните версионизирани обекти.



Фиг. 66 Примерен модел на композиране на версионизирани обекти за същността клас

3.3.2. Примерен модел на обектите в областта на тестирането

Областта на тестирането и осигуряването на необходимото ниво на качество представлява важен момент от жизнения цикъл на софтуерните продукти. Най-често това се обезпечава от създаването на *тестови сценрии*, както и на *тестовото им изпълнение* [12]. Тестовият сценарий се състои от определен набор *тестови стъпки*. Тестовото изпълнение от своя страна се състои от *отчети от изпълнението* на тестовите стъпки. Споменатите елементи могат да притежават свои метаатрибути като например: *автор на сценария*, *дата и час на създаване*, *дата и час на тестово изпълнение*, и др. Следва да се отбележи, че нито един *тестови сценарий* не може да съществува сам за себе си. Той винаги се създава с цел да се провери удовлетворяването на определено изискване, конвенция и/или стандарт, като често един тестови сценарий проверява повече от едно изискване, конвенция или стандарт. На Фиг. 67 е представена диаграма на композиране на обектите за областта на тестирането.



Фиг. 67 Примерен модел на композиране на версионизирани обекти за областта на тестирането

3.4. Сравнителен анализ на предимствата на моделите в прототипа

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

3.4.1. Постановка на експеримента

В рамките на експеримента е заложено:

- Разработка на продукт (програма) съгласно предварително определени изисквания.
- Симулация на намиране и отстраняване на дефекти – два броя.
- Промяна в едно от първоначалните изисквания и неговата последваща реализация.

В рамките на **първия етап** на експеримента ще се използват следните изисквания:

Игра Тамагочи.

1. Тамагочито е необходимо да притежава следните регистри: *енергия* (стойности 0-100), *глад* (стойности 0-100), *щастие* (стойности0-100).

2. При старта на играта на трите регистъра се присвоява стойност 50.

3. Потребителят (играчът) може да извършва следните действия: *хранене*, *игра* с тамагочито, да *приспи* тамачито.

4. При достигане на регистъра *енергия* стойност 0, тамагочито заспива (състояние сън).

5. Тамагочито може да бъде събудено от играча чрез команди *хранене* или *игра*.

6. В състояние сън тамагочито набира енергия със скорост 1 единица в минута.

7. При достигане на регистъра *глад* стойност 0, тамагочито се събужда.

8. При значения на регистъра *глад* по-малки от 10, скоростта на намаляване на регистъра *щастие* се увеличава с 1 единица в минута.

9. Регистърът глад е постоянно намаляващ със скорост 1 единица за 2 минути.

10. При единично хранене (едно събитие) регистърът *глад* на тамагочито се увеличава с 20 единици.

11. Регистърът *щастие* е постоянно намаляващ със скорост на намаляване 1 единица в минута.

12. При игра с тамагочито регистърът *щастие* се увеличава с 20 единици.

13. Когато трите регистъра *глад*, *щастие* и *енергия* достигнат едновременно стойност 0 и останат в това състояние 10 минути,

тамагочито умира. Играчът в този случай може да започне играта отначало.

Дефектите, които ще се отсраняват във **втория етап** на експеримента, са:

1. Несъответствие между изискванията № 4 и № 7. За успешното решаване на този дефект е необходимо поставянето на условията от изискване 7 да имат по-голям приоритет от тези в изискване № 4.

2. Неправилна реализация на изикване № 10, изразяваща се в това че тамагочито увеличава значенията на регистъра за *глад* за едно хранене с 30, а не с 20 единици.

Третият етап от експерименталната задача включва промяна на изискване №12, като то се променя в следният вид:

12a. При игра с тамагочито регистърът *щастие* се увеличава с 15 единици, като едновременно с това се намалява стойността на регистъра *енергия* с 5 единици.

3.4.2. Решаване на задачата с използване на съществуващите системи и подходи

Един съвременен подход за реализиране на софтуерни продукти предполага използването на система за управление на заявките (изисквания и дефекти) и система за управление на версията (файлово базирани), като в частност ще се разглеждат системите с открит код Trac и Subversion.

В рамките на експерименталната задача следва да се извършат следните действия:

1. Регистриране на изискванията под формата на заявки в Trac. В резултат на това всяко едно изискване получава свой собствен уникален номер, по който може да се идентифицира.

2. Разработване на всяко едно отделно изискване под формата на UML дизайн и файл (файлове) с изходен код.

3. Публикуване на новия/променения файл (файлове) с изходен код и UML дизайн. Необходимо е да се изтъкне задължителното условие програмистът да укаже номера на изискването, което е реализирано, в коментара към публикуването. Паралелно с публикуването на промените, програмистът променя статуса на заявката (изискването) в Trac на реализирана.

4. След успешната реализация на всички изисквания, във фазата на тестирането (**втори етап** от експерименталната постановка) се установяват два дефекта – първият дефект на ниво изискване, вторият – на ниво реализация. Дефектът на ниво изикване се решава, като изискване №7 се преотваря в Trac, коригира се и дописва коментар. За дефекта по изискване №10 заявката в Trac се преотваря с коментар.

5. За намирането на местата, в които е необходимо да се извъши корекция, програмистът използва коментарите, които е оставял в момента на публикуване от стъпка 3 по-горе. Прави впечатление, че използването на файлово-ориентирана система за управление на версията изисква от програмиста допълнителни усилия за намиране мястото на дефекта в изходния код. На ниво архитектура (UML диаграми) системата за управление на версиите съвсем не помага, понеже отделните елементи във файла с диаграмите съответстват на отделно изискване. Това задължава програмиста да извърши детайлен разбор на диаграмите, за да намери мястото, в което е необходимо да се променят елементите. След отстраняване на дефектите програмистът публикува промените по аналогичен на описания в стъпка 3 начин.

6. За реализацията на **третия етап** от експерименталната постановка е необходимо или да се създаде нова заявка в системата Trac или тази, която съответства на изискване **12** да се преотвори и

модифицира. Тук се забелязва необходимостта програмистът отново да извърши анализ подобен на този в стъпка **5** (по-горе).

3.4.3. Решаване на задачата при използване на средствата и подхода на прототипа

Настоящия пункт има за цел да демонстрира предимствата, които предоставя използването на разработените в настоящата дисертация модели, методи и методологична рамка. В рамките на експерименталната задача следва да се извършат следните действия:

1. Като първа стъпка може да се определи регистрирането на изискванията в прототипа под формата на версионизирани обекти, с последващо маркиране като работни задачи от анализатора на проекта в неговото работно пространство. Към момента на приключване създаването на изскванията в системата, анализаторът ги публикува до главното работно пространство.

2. Започвайки работа по всяко едно отделно изискване, програмистът го отбелязва като активна работна задача в своето работно пространство. Така системата автоматично създава за всички създадени и променени обекти причинно-следствени връзки.

3. Програмистът публикува всички нови/променени обекти в главното работно пространство. Прави впечатление възможността от използване на свободно ниво на гранулираност на артефактите.

4. Във фазата на тестирането се установяват двата дефекта от постановката на експеримента – първият дефект на ниво изискване, вторият – на ниво реализация. Дефектът на ниво изикване се решава, като тестерът създава обект-дефект по изисквания №7 и №4 и изпълнител – анализаторът по проекта. За дефекта по изискване №10, тестерът създава нов обект-дефект с коментар и изпълнител – програмистът по проекта. Двата новосъздадени обекта-дефекти се маркират като работни задачи и се публикуват в главното работно пространство на проекта. Работата по

отстраняване на първия дефект се състои в отбелязване на дефекта като активна работна задача, обикновена редакция изискване №7 и последващо публикуване в главното работно пространство.

5. Използвайки причинно-следствените връзки, създадени от системата в стъпка 2, програмистът има възможност бързо и точно да определи в коя част от кода и в UML диаграмите е необходимо да се извършат нужните промени. След извършване на корекциите програмистът публикува новите версии на обектите в главното работно пространство

6. При реализирането на третия етап чрез използване на моделите представени в дисертацията предполага следната последователност от действия: анализаторът създава нова версия на изискване №12. Програмистът чрез използване на причинно-следствените връзки произлизащи от предишната версия на изискване №12 определя бързо и лесно местата в UML модела на системата и изходният код, където се предполага да се извърши допълнителната разработка. След приключване на разработката обектите се публикуват до главното работно пространство, от където може да се сглоби версия за предоставяне на крайният потребител.

3.4.4. Сравнителен анализ на резултатите от експеримента

В настоящия пункт е направен опит да се сравнят резултатите от експерименталните реализации в зависимост от използването на модели и инструменти. В таблицата по-долу е представено сравнение на основните моменти от направения експеримент.

Фаза	**Съществуващи технологии**	**Предложени модели**
Регистриране на изисквания.	Ръчно регистриране в Trac	Регистрация като версионизиран обект
Разработка на	Създаване на отделени	1. Избор на изискване, като

архитектурата, изходния код и тестовите сценарии	файлове.	работна задача. 2. Създаване като отделни версионизирани обекти (автоматично свързани към изисквания).
Регистриране на арх-рата и изходния код в с-ма за упр. на версии	1. Ръчво указване на № изискване в коментара към публикуването. 2. Ръчна смяна на статуса на изискването в Trac.	Публикуване в главно работно пространство.
Тестиране и определяне на причината на дефекта	1. Създаване на дефект в Trac, и ръчно свързване с изискване.	1. Избор на изискване, като работна задача. 2. Създаване на дефект (автоматично свързан към изискване)
Отстраняване на дефект	1. Ръчно намиране на мястото на дефекта – в изходния код или в изскване. 2. Отстраняване на дефекта. 3. Публикуване на корекцията с указване № на дефекта. 4. Ръчна смяна статуса на дефекта.	1. Генериране на справка и намиране мястото на дефекта. 2. Избор на дефекта, като работна задача. 3. Отстраняване на дефекта. 4. Публикуване в главно работно пространство.

Детайлният анализ от реализациите показва следното:

- Сега съществуващите практики предполагат използването на отделни системи за управление на заявките/изискванията и система за контрол на версиите. Това изисква допълнителни усилия по синхронизация на данните между системите от страна на екипа, участващ в проекта. Този факт неминуемо води по

повишаване на риска от човешка грешка в рамките на проекта, а също така увеличава натоварването на сътрудниците често с несвойствени за тях дейности. Използването на единна система, позволяваща както да се управлява изискванията, тестовете, така и да се управлява версията на изходният код, води до намаляването на този риск.

- Третият етап от експерименталната задача – промяна на изскване към системата – демонстрира в пълна сила предимствата от разработените модели, с цел подпомагане и автоматизиране работата на участниците в процеса на създаване на софтуерни продукти. Те биват разтоварени от необходимостта да ръчно преглеждат цялостната архитектура и изходен код, за да определят местата за локална промяна.

3.5. Изводи

От направеното разработване на модели и реализация на прототип (предложени във Втора Глава и реализирани в Трета Глава) на система за управление на версии в среда с йерархично композирани работни пространства може да се направят следните изводи:

От направената практическа разработка на прототип реализиращ разработените модели в настоящата глава могат да бъдат направени следните изводи:

1. Изследвани са възможностите за скоростно разработване на прототип на системата. Анализирани и избрани са технологии за разработка.
2. Определен е архитектурния модел на прототипа. Разработен и представен е навигационен модел на системата. Представени са алгоритмите, които са реализирани.

3. Разработени са примерни модели на композиция на версионизирани обекти на същноста клас и в областта на тестирането.
4. Направена е теоритично-експериментална симулация на процеса по разработка на система, с цел да се направи сравнителен анализ на получените резултати. Направената симулация позволява да се сравнят предимствата, които предоставя моделите, разработени в дисертацията.

Заключение

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

Теоретичните разработки, представени във Втора Глава, бяха насочени в три основни направления – модел на версионизиран обект, модел на среда за версионизиране, базирана на йерархична композиция от работни пространства, и адаптация на метод за полу-автоматично създаване на проследяващи връзки. Предложените модели предоставят възможност за допълнително автоматизиране процеса на разработка на софтуерни продукти, и позволяват на участниците в процеса да се фокусират върху задачите, за които отговарят, а не на процеса на отчитане на свършената работа.

Поставените в увода задачи на дисертацията са решени, в резултат на което са постигнати **следните резултати:**

1. В резултат на изследване на основните въпроси и анализ на дадената предметна област, са формулирани нерешените проблеми в съвременните системи за управление на версии.
2. Създаден е авторски модел на версионизиран обект, който позволява да се определи степента на гранулираност на данните.
3. Предложен е авторски модел на среда с йерархично композирани работни пространства и са определени правилата за управление на версия на обекти в тази среда. Това позволява всички участници в процеса да работи изолирано, както и да се кооперират по определени задачи или направления.
4. Направена е адаптация на метод за проследимост на промени, базиран на събития, за среда с йерархично композирани работни пространства, който осигурява по-добри възможности за анализ на промените.

5. Определена е терминологията в областта на версионизирането с използването на йерархично композирани работни пространства.
6. Предложена е методологична рамка за използване на разработените модели. Направен е сравнителен анализ между използването на съществуващите инструменти и разработените модели. Анализът показва увеличаване на степента на автоматизация на част от дейностите при създаване на софтуерни продукти.
7. Реализиран е функционален прототип на система за управление на версии, с помощта на който е направена апробация на разработените модели.

Литература

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

[2] Белладжио Д., Т. Миллиган, Разработка програмного обеспечения: управление изменениями, ДМК Пресс, Москва, 2009,.

[3] Екел Брус, Да мислим на JAVA, том 1, СофтПрес, София, 2001.

[4] Коуберн Ал. Каждому проекту своя методология, 2005, http://www.citforum.ru/SE/project/meth_per_project/ (посетен през март 2011)

[5] Манева Н., А. Ескенази, Софтуерни Технологии, ИК Анубис, София, 2001.

[6] Наков, Св., и колектив, Въведение в програмирането с Java, Фабер, Велико Търново, 2008.

[7] Наков, Св., и колектив, Въведение в програмирането със C#, Фабер, Велико Търново, 2011.

[8] Наков, Св., Интернет програмиране с Java, Фабер, Велико Търново, 2004, ISBN 954-775-305-3.

[9] Фейсон Т., Borland C++ Обектно-ориентирано програмиране - Част I, Нисофт, София, 1994.

[10] Хемраджани, А., Гибкая разработка приложений на Java с помощю Spring, Hibernate и Eclipse, ООО "И.Д.Вилямс", Москва, 2008.

[11] Ambler, S. W., Pr. J. Sadalage,Refactoring Databases: Evolutionary Database Design, Addison Wesley Professional, 2006.

[12] Ammann, P., J. Offutt, Introduction to software testing, Cambridge University Press, 2008.

[13] Amza, C., Cox, A. L., and Zwaenepoel, W. Distributed versioning: consistent replication for scaling back-end databases of dynamic content web sites. In Proceedings of Middleware '03: Proceedings of the ACM/IFIP/USENIX

2003 International Conference on Middleware (Rio de Janeiro, Brazil, June 16 - 20, 2003), pp. 282–304, Springer-Verlag New York, Inc., New York, 2003.

[14] Andrews, J., Feature: No More Free BitKeeper, http://kerneltrap.org/node/4966, 2005, (посетен през януари 2012).

[15] Apache OpenJPA Project, http://openjpa.apache.org/ (посетен през март 2012).

[16] Arvin, Tr.,Comparison of different SQL implementations, 2011, http://troels.arvin.dk/db/rdbms/ (посетен през март 2012)

[17] Asun ion, H. U. Towards practical software traceability. In Companion of the 30th international Conference on Software Engineering (Leipzig, Germany, May 10 - 18, 2008) pp.1023-1026, ICSE Companion '08. ACM, New York, NY, 2008, DOI= http://doi.acm.org/10.1145/1370175. 1370228.

[18] Azad, A. Implementing Electronic Document and Record Management Systems. Auerbach Publications, Books24x7, 2008 http://common.books24x7.com/book/id_26435/book.asp (посетен през март 2012)

[19] Barcis, R., JavaServer Faces (JSF) vs Struts, 2004, http://websphere.sys-con.com/node/46516 (посетен през декември 2011)

[20] Binkley, D., Horwitz, S., and Reps, T. Program integration for languages with procedure calls. ACM Trans. Softw. Eng. Methodol., Vol.4, no. 1, January 1995, pp. 3-35, 1995, DOI= http://doi.acm.org/10.1145/201055. 201056

[21] Boehm, B., A. Egyed, Kwan J., Port D., Shah A., Madach R., Using the WinWin Spiral Model: Case Study, Computer, Vol. 31, no. 7, pp. 33–44, July 1998.

[22] Bourque, P., R. Dupuis, A. Abran, J.W.Moore, L. Tripp, The Guide to the Software Engineering Boody of Knowledge, 1999, IEEE Software Vol. 16, no. 6, November/December 1999.

[23] 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., September 1991.

[24] 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), pp 69–78, WikiSym '06. ACM, New York, 2006, DOI= http://doi.acm.org/10.1145/1149453.1149469.

[25] 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), pp.14-19, TEFSE '05. ACM, New York, 2005, DOI= http://doi.acm.org/10.1145/1107656.1107660

[26] 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), pp. 362–371, ICSE '05. ACM, New York, 2005, DOI= http://doi.acm.org/10.1145/1062455.1062525

[27] Clemm, G., Amsden, J., Ellison, T., Kaler, C., and Whitehead, J. Versioning Extensions to WebDAV (Web Distributed Authoring and Versioning). RFC. RFC Editor, 2002.

[28] Collins-Sussman B., Fitzpatrick, B. W., Pilato C. M., Version Control with Subversion, book compiled from Revision 10945, 2008, http://svnbook.red-bean.com/en/1.0/index.html (посетен през март 2009)

[29] Conradi, R. and Westfechtel, B. 1998. Version models for software configuration management. ACM Comput. Surv., Vol. 30, no. 2, pp. 232–282, June 1998, DOI= http://doi.acm.org/10.1145/280277.280280.

[30] Converse, T., Joyce P., PHP Bible, 2nd Edition, Wiley Publishing, 2002.

[31] Cruz, José R.C. , Branching Out with Git, MacTech Magazine, Vol. 26, no. 04, http://www.mactech.com/articles/mactech/Vol.26/26.04/BranchingOutWithGit/index.html (посетен през януари 2012).

[32] Dart, S. 1991. Concepts in configuration management systems. In Proceedings of the 3rd international Workshop on Software Configuration Management (Trondheim, Norway, June 12 - 14, 1991), pp. 1-18, P. H. Feiler, Ed. ACM, New York, 1991, DOI= http://doi.acm.org/10.1145/111062.111063.

[33] Doray, A., Beginning Apache Struts: From Novice to Professional, Apress, 2006.

[34] Drupal - Open Source CMS, http://drupal.org/ (посетен през януари 2012).

[35] Estublier, J., Leblang, D., Hoek, A., Conradi, R., Clemm, G., Tichy, W., and Wiborg-Weber, D., Impact of software engineering research on the practice of software configuration management. ACM Trans. Softw. Eng. Methodol., Vol. 14, no. 4, pp. 383-430, October 2005, DOI= http://doi.acm.org/10.1145/1101815.1101817.

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

[37] Feiler, P. P., Configuration Management Models in Commercial Environments, Technical Report, CMU/SEI-91-TR-007, 1991.

[38] Feiler, P. P. Software process support through software configuration management. In Proceedings of the 5th international Software Process Workshop on Experience with Software Process Models (Kennebunkport, Maine, United States, October 10 - 13, 1989), pp. 58-60, International Software Process Workshop. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[39] Ferguson, Jeff, C# Bible, John Wiley & Sons, 2002.

[40] Gilmore, W. J., Beginning PHP and MySQL: From Novice to Professional, Apress, 2010.

[41] Git - Fast Version Control System, http://git-scm.com/ (посетен през януари 2012).

[42] 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), pp. 11-15, International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 2009, DOI= http://dx.doi.org/10.1109/TEFSE.2009.5069576.

[43] Hillyer, M., Managing Hierarchical Data in MySQL, 2005, http://dev.mysql.com/tech-resources/articles/hierarchical-data.html (посетен през май 2010).

[44] Horwitz, S., J. Prins, and Th. Reps.. Integrating noninterfering versions of programs. ACM Trans. Program. Lang. Syst. Vol. 11, no. 3, pp. 345-387, July 1989, http://doi.acm.org/10.1145/65979.65980.

[45] Hudson Continuous Integration, http://hudson-ci.org/ (посетен през януари 2012).

[46] IIBA, A Giude to the Business Analysis Body of Knowledge (Version 2.0) BABOK, 2009.

[47] Jain, P. et al. , J2EE Professional Projects, Premier Press, 2002.

[48] Jarke, M. 1998. Requirements tracing. Commun. ACM, Vol. 41, no. 12, pp. 32-36. December 1998, DOI= http://doi.acm.org/10.1145/290133. 290145.

[49] Jones, M. T., Version control for Linux, 2006, http://www.ibm.com/ developerworks/linux/library/l-vercon/, (посетен през февруари 2009).

[50] Joomla!, http://www.joomla.org/ (посетен през февруари 2012).

[51] Katz, M., Practical RichFaces, Apress, 2008.

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

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

[54] Kögel, M., Towards software configuration management for unified models. In Proceedings of the 2008 international Workshop on Comparison and Versioning of Software Models (Leipzig, Germany, May 17 - 17, 2008), pp. 19-24., CVSM '08. ACM, New York, 2008, DOI= http://doi.acm.org/10.1145/1370152.1370158

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

[56] Krahn, R., Ingalls, D., Hirschfeld, R., Lincke, J., and Palacz, K., 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), pp. 1-10,WikiSym '09. ACM, New York, NY, 2009, DOI= http://doi.acm.org/10.1145/1641309.1641324

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

[58] Linwood, J., Minter, D., Pro Hibernate 3, 2008.

[59] Mann, Kito D., Interview with Manfred Geiler, 2004 http://www.jsfcentral.com/articles/geiler-04-04.html (посетен през януари 2012).

[60] Marcus, A. and Maletic, J. I.. 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), pp. 125-135, International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 2003.

[61] Mercurial SCM, http://mercurial.selenic.com/ (посетен през януари 2012).

[62] Microsoft Sharepoint, http://sharepoint.microsoft.com/ (посетен през юни 2010).

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

[64] Morse, T., CVS, Linux J., vol. 1996, no. 21, page 3, January 1996.

[65] Morse, T., CVS: Version Control Beyond RCS, Linux Journal, Jan 01, 1996, http://www.linuxjournal.com/article/1118 (посетен през юни 2011).

[66] Munson, J. P. and Dewan, P., A flexible object merging framework. In Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work (Chapel Hill, North Carolina, United States, October 22 - 26, 1994), pp. 231-242, CSCW '94. ACM, New York, NY, 1994, DOI= http://doi.acm.org/10.1145/192844.193016

[67] 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 (посетен през януари 2012).

[68] 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), pp. 215-224, ICSE '05. ACM, New York, 2005, DOI= http://doi.acm.org/10.1145/1062455.1062504

[69] 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), pp. 35-36, OOPSLA '04. ACM, New York, 2004, DOI= http://doi.acm.org/10.1145/1028664.1028684

[70] Nitin, K. L., Ananya S., Mahalakshmi K., and S. Sangeetha, iBATIS, Hibernate, and JPA: Which is right for you?,JavaWorld.com, 07/15/08, http://www.javaworld.com/javaworld/jw-07-2008/jw-07-orm-comparison.html (посетен през юли 2010).

[71] O'Donovan, Brian, RCS Handbook, 1992, http://www.burlingtontelecom.net/~ashawley/rcs/rcs_handbook.html (посетен през март 2011).

[72] Official RCS Homepage, http://www.cs.purdue.edu/homes/trinkle/RCS/ (посетен през март 2011).

[73] 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 (посетен през септември 2011).

[74] Pressman, R. S., Software Engineering: A Practitioner's Approach, McGraw-Hill Professional, 2005.

[75] Price, Derek R., CVS—concurrent versions system v1.11.22, http://ximbiot.com/cvs/manual/cvs-1.11.22/cvs.html, 2006 (посетен през април 2009).

[76] Puntikov, N. I., Volodin, M. A., and Kolesnikov, A. A.. AVCS: the APL version control system. In Proceedings of the international Conference on Applied Programming Languages (San Antonio, Texas, United States, June 04 - 08, 1995), pp. 154-161, M. Griffiths and D. Whitehouse, Eds. APL '95. ACM Press, New York, 1995, DOI= http://doi.acm.org/10.1145/206913.206995

[77] Ramesh, B., Matthias J., Towards Reference Models for Requirements Traceability, 1999.

[78] Rochkind, M. J., The Source Code Control System. In IEEE Transactions on Software Engineering SE, Vol. 1, no. 4, pp. 364–370, December 1975.

[79] Royce W.W., Managing the Development of Large Software Systems, Proceedings of IEEE WESCON, pp. 328-338, 1970.

[80] Ruparelia, N. B. 2010. The history of version control. SIGSOFT Softw. Eng. Notes, Vol. 35, no. 1, pp.5-9, January 2010, DOI= http://doi.acm.org/10.1145/1668862.1668876

[81] Schmetzer, J., Introduction to Ant, http://www.exubero.com/ant/antintro-s5.html (посетен през декември 2011).

[82] Schmidt, M., Wenzel, S., Kehrer, T., and Kelter, U. History-based merging of models. In Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models (May 17 - 17, 2009), pp. 13-18. International Conference on Software Engineering. IEEE Computer Society, Washington, 2009, DOI= http://dx.doi.org/10.1109/CVSM.2009.5071716

[83] Schwaber, K., Agile Project Management with Scrum, Microsoft Press, 2004.

[84] Silva, M., Gedye, D., Katz, R., and Newton, R.. 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), pp. 264-269, DAC '89. ACM, New York, 1989. DOI= http://doi.acm.org/10.1145/74382.74427

[85] 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, Vol. 5, no. 1, pp. 17-24, March 1997, DOI= http://doi.acm.org/10.1145/253452.253474

[86] Stephens, S. M. , Johan Rung , Xavier Lopez, X.: Graph data representation in oracle database 10g: Case studies in Life science, IEEE Data Eng. Bull, vol. 27, pages 61-67, 2004.

[87] SUN SHIPS JDK 1.1 -- JAVABEANS INCLUDED, http://web.archive.org/web/20060706192215/http://www.sun.com/smi/Press/sunflash/1997-02/sunflash.970219.0001.xml (посетен през март 2012).

[88] Sun WorkShop TeamWare User's Guide, http://docs.sun.com/source/806-3573/TeamWareTOC.html (посетен през юли 2010).

[89] Sun WorkShop TeamWare http://caligari.dartmouth.edu/doc/solaris-forte/tw-help/TWHelp.html (посетен през юли 2010).

[90] The 1998 ACM Computing Classification System, http://www.acm.org/about/class/1998 (посетен през юли 2009).

[91] The Java Community Process, JSR 19: Enterprise JavaBeansTM 2.0, 2002, http://www.jcp.org/en/jsr/detail?id=19 (посетен през март 2012).

[92] The Java Community Process, JSR 220: Enterprise JavaBeansTM 3.0, 2006, http://www.jcp.org/en/jsr/detail?id=220 (посетен през март 2012).

[93] The JSF Matrix (reborn) http://www.jsfmatrix.net/ (посетен през март 2012).

[94] Tiako, P. F., Lindquist, T., and Gruhn, V,. Process support for distributed team-based software development workshop. SIGSOFT Softw. Eng. Notes, Vol 26, no. 6, pp. 31-33, November 2001, DOI= http://doi.acm.org/10.1145/505532.505539

[95] Terrence R., 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 (посетен през декември 2011).

[96] Tichy, W. F., RCS—a system for version control. Softw. Pract. Exper., Vol. 15, no. 7, pp. 637-654. July 1985, DOI= http://dx.doi.org/10.1002/spe.4380150703

[97] Troelsen, A., Pro C# 2010 and the .NET 4 Platform, Fourth Edition, Apress, 2010.

[98] Turvey, St., Duelling databases: Four apps tested, ZDNet.com.au , 2005,December 23rd, http://www.zdnet.com.au/duelling-databases-four-apps-tested-139226455.htm (посетен през януари 2012).

[99] WebDAV Resources, http://www.webdav.org/ (посетен през март 2010).

[100] Weiss, C., Premraj, R., Zimmermann, T., and Zeller, A., How Long Will It Take to Fix This Bug?, In Proceedings of the Fourth international Workshop on Mining Software Repositories (May 20 - 26, 2007), p. 1, International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 2007, http://dx.doi.org/10.1109/MSR.2007.13

[101] Westfechtel, B., 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), pp. 68-79, P. H. Feiler, Ed. ACM, New York, NY, 1991, DOI= http://doi.acm.org/10.1145/111062.111071

[102] Whitehead, E. J., 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), pp. 195-204, HYPERTEXT '01. ACM Press, New York, NY, 2001, DOI= http://doi.acm.org/10.1145/504216.504265

[103] Wiegers, Karl E., Software Requirements, Second Edition, Microsoft Press, 2003.

[104] 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 (посетен през декември 2011).

[105] Wikipedia contributors, "Comparison of relational database management systems", http://en.wikipedia.org/w/index.php?title=Comparison_of_relational_database_management_systems&oldid=465424149, (посетен през декември 2011).

[106] Wikipedia contributors, “Document management system”, Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/w/index.php?title=Document_management_system&oldid=358652743 (посетен през април 2010).

[107] Wikipedia contributors, "Java Server Daces", http://en.wikipwdia.org/w/index.php?title=JavaServer_Faces&oldid=45747157 (посетен през декември 2011).

[108] Winter, V., Harvey S. , Mansour Z. , Prasanna R. A., Aspect traceability through invertible weaving, In Early Aspects Workshop at AOSD’06, 2006.

[109] WordPress › Blog Tool, Publishing Platform, and CMS, http://wordpress.org/ (посетен през януари 2012).

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

[111] Zeller, A., Snelting, G., Unified versioning through feature logic. ACM Trans. Softw. Eng. Methodol., vol. 6, no. 4, pp. 398-441, October 1997, DOI= http://doi.acm.org/10.1145/261640.261654

[112] Zhang, Y. , Witte R., Rilling J., Haarslev V., An Ontology-based Approach for Traceability Recovery, 3rd International Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering (ATEM 2006), pp. 36-43, 2006.

Приложение 1 – Описание на модела на данните

Таблица **product**		
Таблицата представлява същността Продукт.		
product_id	**Int, PK**	
name	**Characters(50)**	Наименование на продукта

Таблица **release**		
Таблицата представлява същността Издание на продукт		
release_id	Int, **PK**	
product_id	Int, **FK** to **product**	Вторичен ключ към продукта, чието издание се явява записа
master_ws_id	Int, **FK** to **worspace**	Вторичен ключ към главното работно простанство на изданието
name	Characters(50)	Наименование на изданието (име на доставяната версия)

Таблица **release_arcs**		
Релационна таблица, представяща информация относно последователността от изданията на даден продукт		
arc_id	Int, **PK**	
source_release_id	Int, **FK** to **release**	Издание първоизточник
target_release_id	Int, **FK** to **release**	Целево издание

Таблица **workspace**		
Таблица представлява същността Работно пространство		
ws_id	Int, **PK**	
release_id	Int, **FK** to **release**	Вторичен ключ, указващ изданието към което се намира работното пространство
ancestor_ws_id	Int, **FK** to **workspace**	Вторичен ключ, указващ на родителското работно пространство
user_id	Int, **FK** to **user**	Вторичен ключ, указващ към потребителят, който работи в момента в даденото пространство
name	Characters(50)	
lft	Int	флаг, лява граница на записа
rgt	Int	флаг, дясна граница на записа

Таблица **user**	
Таблица представлява същността Потребителски профил	
user_id	Int, **PK**
login	Characters(50)
password	Characters(50)

Таблица versioned_object
Таблица представлява същността Версионизиран обект

user_id	Int, **PK**	
workitem	Boolean	Флаг, указващ дали обектът представляван от записа се явява работна задача

Таблица **object_version**		
Таблица представляваща същността Версия на обект		
gid	Int, **PK**	Сурогатен първичен ключ
vo_id	Int, **FK** to **versioned_object**	Вторичен ключ, указващ към същността версионизиран обект
ws_id	Int, **FK** to **workspace**	Вторичен ключ, указващ версията за кое работно пространство се явява локална
version_number	Int	номер на версията, в рамките на версионизираният обект
name	Characters(50)	наименование на обекта във версията
datum	Text	данни във версията на обекта
delete_flag	Boolean	Фалаг, указващ дали версията се явява версия на изтриване за обект

Таблица **workitem_attachment**		
Релационна таблицата, която съдържа информация относно асоциираните със съответното работно пространство работните задачи.		
wia_id	Int, **PK**	Сурогатен първичен ключ
ws_id	Int, **FK** to **workspace**	Вторичен ключ, указващ пространството, в което е избрана работната задача
wi_obj_id	Int, **FK** to **versioned_object**	Вторичен ключ, указващ обект, с вдигнат флаг за работна задача.

Таблица **vcomposer**		
Релационна таблица, която съдържа информация за свързаните под-обекти на даден обект		
composer_id	Int, **PK**	Сурогатен първичен ключ
super_object_gid	Int, **FK** to **object_version**	Вторичен ключ, указващ към версия на супер-обект
sub_object_gid	Int, **FK** to **object_version**	Вторичен ключ, указващ към версия на под-обект

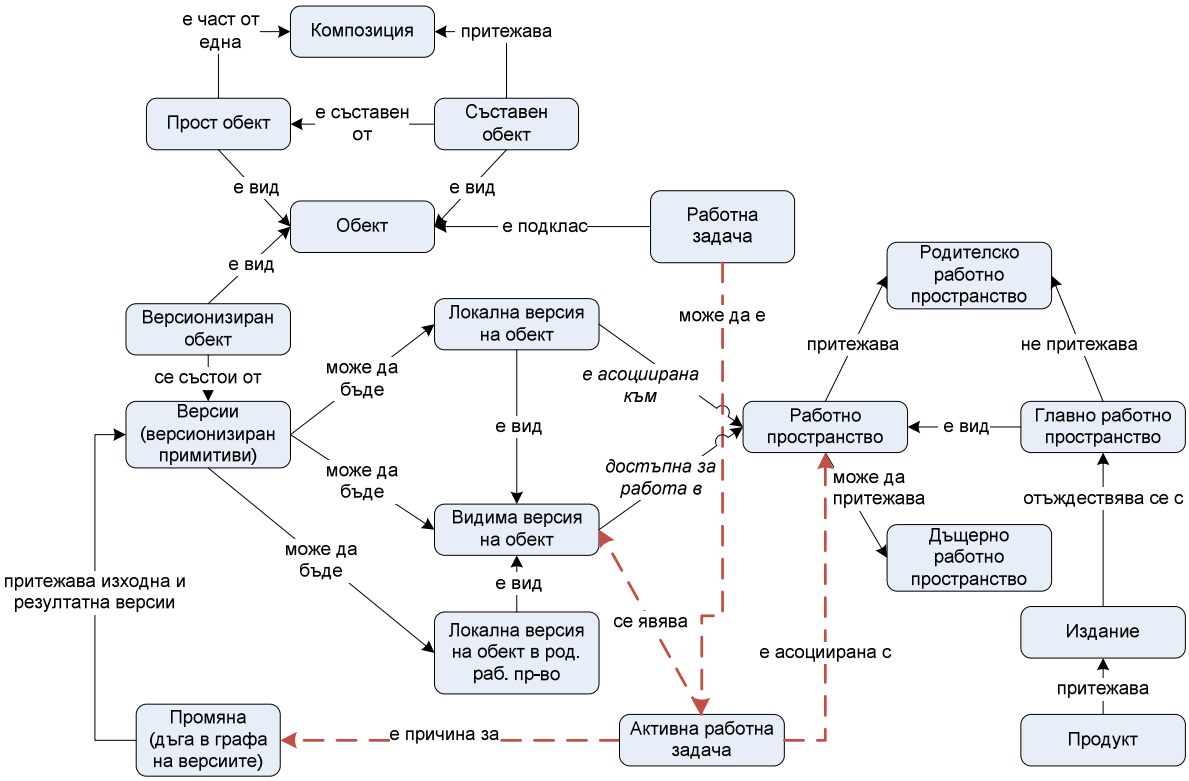
Таблица **version_arc**
Релационна таблица, която съдържа информация за история на промените

на обект (прехода от една версия към друга)		
arc_id	Int, **PK**	Сурогатен първичен ключ
source_gid	Int, **FK** to **object_version**	Вторичен ключ, указващ към изходна версия на обект
target_gid	Int, **FK** to **object_version**	Вторичен ключ, указващ към резултатна версия на обект
user_id	Int, **FK** to **user**	Вторичен ключ, указващ към потребителски профил, който е извършил промяната
arcdate	Timestamp	Дата и час на промяната
notes	String(50)	Забележки

Таблица **cause**		
Релационна таблица, която съдържа информация относно причините (т.е. работните задачи), довели до конкретна промяна версията на конкретен обект		
cause_id	Int, **PK**	Сурогатен първичен ключ
arc_id	Int, **FK** to **version_arc**	Вторичен ключ, указващ към дъга (промяна) от версионизиращият граф (version_arc), коя е причинена от работна

		задача
cause_object_gid	Int, **FK** to **object_version**	Вторичен ключ, указващ към версия на обект, който се явява работна задача причинител на промяна.

Приложение 2 – Речник и онтология на термините



Фиг. 68 Онтологична диаграма на термините

Активна работна задача – задача, асоциирана с текущото работно пространство и над която работи потребителя.

Версионизиран обект – обект съставени от две части – състояния на обекта (версии) и граф на версиите [29, 85].

Версия (версионизиран примитив) – екземпляр от състоянията на обекта.

Видима версия на обект – Под *видима версия на версионизиран обект* за дадено работно пространство, ще се разбира такава версия на обекта, с която потребителят може да работи *(Дефиниция 9)*.

Главно работно пространство – Главно работно пространство се нарича работно пространство, в което се извършва окончателната сборка и подготовка на издание на продукта *(Дефиниция 7)*.

Дъщерно работно пространство – за дадено пространство дъщерно пространство се нарича такова пространство, за което като родителско пространство е определено даденото.

Издание (на продукт) –определена фиксирана негова версия *(Дефиниция 5)*.

Композиция – същност, определяща връзката между супер-обект и под-обект *(Дефиниция 2)*.

Локална версия на обект – Под *локална версия на версионизиран обект* за дадено работно пространство, ще се разбира такава негова версия, която е асоциирана с работното пространство *(Дефиниция 8)*.

Обект – явление, предмет, към който е насочена някаква човешка дейност [1, стр. 546]. Тук, определена съвкупност данни, резултат от дейност.

Продукт – обект на материалното или нематериалното производство, който след своето създаване може да бъде размножен и разпространяван сред клиентите *(Дефиниция 4)*.

Промяна, следствие – Следствие, породено от причина, ще се нарича наборът от промени над обекти в резултат от изпълнението на работна задача дъга в графа на версиите на версионизиран обект *(Дефиниция 13)*.

Прост обект – обект, за който няма асоциирани под-обекти. *(Дефиниция 3)*.

Работно пространство – Работно пространство се нарича място, където се извършват определени дейности по създаването на версия на продукт *(Дефиниция 6)*.

Ред на обект – $R_{\text{обект}} = \begin{cases} 0, \sum \text{под} - \text{обекти} = 0 \\ N, \max\left(R_{\text{под}-\text{обект}}\right) = N - 1 \end{cases}$ *(Дефиниция 3).*

Родителско работно пространство – работно пространство, към което е асоциирано дадено работно пространство.

Степен на гранулираност на обект – реда на обекта *(Дефиниция 3)*.

Съставен обект – обект, който е съставен от други обекти посредством композиция *(Дефиниция 1)*.

Работна задача – съвкупността от дейности, която следва да се извърши *(Дефиниция 12)*.

Приложение 3 – Прототип на система за управление на версии, базирана на йерархични композиции от пространства (на DVD носител)

Дискат съдържа всичкият необходим софтуер за разгръщане на прототипа на Windows базирана система, инструкция за разгръщане, среди за разработка и изходният код на прототипа. Тук приложеният DVD диск е неразделна част от дисертационния труд. Прототипа е разработен с лиценз GNU GPL v2 (*http://waww.gnu.org/licenses/old-licenses/gpl-2.0.html*).