Title

Abstract:

1. Challenges - Problem statement

need of data granulation

workspaces

2. Object Model

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.

Lead author in version control domain [29, 85] determine the version object as two part entity - object states, and object versions' graph. Object version graph is a graph where the nodes represents object states, and arcs represents version transitions.

Main feature that an versioned object suppose to implement is to provide possibility to define data granulation in a free way. This feature suppose to be supported by the model of versioned object. This lease us to the need of defining of object compositions as part of the model. The formal definition of the composed object could be defined as follow:

1. Composed object is an object that is built from other object using composition entity. Съставен обект се нарича обект, който е съставен от други обекти (версии на обекти) посредством композиция.
2. As composition we will comprise the entity that defines the relation between superobject and subobject. One composed object could be superobject for one or more composition instances, i.e. to be built by one or more subobjects.

Adding of the composed objects and subobjects in the domain leads to the need of redefinition of versioning process over versioned objects. On the following diagram is presented an example for change of version composition. Here arrows represent changes in objects’ content among different versions of end product.



Фиг. 26 An example for change in objects’ content

On versions binding of certain versioned object it is necessary to use relation of type foreign key. From this pursue that **versioned object entity** has to consist only of unique and immutable number. It is useful to regard that number as primary key for the entity.

Object’s versions could be разглеждат as its primitives (**versioned primitives**) with the following attributes:

* Versioned object id
* Version number – a serial number which specify in unique way the version within the versioned object.
* Object’s name. Setting the name to be on primitive level allows to the user to track among different versions even then the object is renamed. This model becomes more complete, excluding the weakness related with object (file) renaming in systems as CVS, SVN, Git, Mercury, and etc. [28, 41, 49, 61, 64, 75].
* Object content.

Versioned primitive is determined in unique way using the couple **versioned object id** and **version number**. In spite of the possibility to use that unique pair and compound key, good practice in ER model design [11] recommends that each entity to possess its own not compound key. In our case we will introduce an additional field as a primary key – global version number.

Versioning of composed object requires to define an additional entity – Versioned primitive composition (in short Composition). This entity is a relation entity that bing in a unique way version of a superobject with version of a subobject. The following attributes of composition could be defined:

* Global number of superobject
* Global number of subobject.

Despite of the fact that attribute combination is always unique, we will use additional attribute for primary key – composition id.

Due to accounting and change traceability needs, our model will be extended in order to support th feature of versioned graph. In ER models graph structures could be presented using two entities – entity for nodes and entity for arcs [86]. Looking to versioned object definition we could conclude that graph nodes correspond to versioned primitive entity. The missing part will be implemented as new entity which will correspond to graph arcs – arc of version graph. The new entity requires the following attributes:

* Arc id – primary key.
* Global number of source version.
* Global number of target version.
* User that made the change.
* Date and time of change.
* Additional change data.

On diagram ### is presented the ER model of versioned object.



Фиг. 27 ER model of versioned object

### Versioning of composed object

In current paragraph we will represent the peculiarities in versioning of composed objects with rank one. Based on that, we will specify the versioning process of composed objects with rank N. The following definition specifies the term rank of composed object.

1. Composed object with rank zero, i.e. simple object is an object which doesn’t have associated with itself subobjects. Composed object with rank N is an object for which the largest rank of associated subobject is equal to N-1.

Granularity degree of an object is its rank.

It is important to note that definition of composed object doesn't apply any restrictions on subobjects. This leads to the following conclusion:

1. One subobject itself can be part of composed object therefore we can build a composition of composed objects.

One of core tasks ahead of current article is to avoid unnecessary complication of the models. Havig that as well as missing necessity, we can specify the following restriction rules during building of composed objects:

1. In given composition of composed objects, certain object can be at most once.
2. One object can be part of at most one object composition.



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 (*суперобект*) с подлакътници (*подобекти*). Когато се отделят подлакътниците от стола, се получава нова версия на стола – стол без подлакътници. Трябва да се отбележи, че конкретният *подобект* не си променя версията, т.е. в описания случай подлакътниците си остават подлакътници. Така се получава единствено промяна в композициите на суперобекта. Подобно е положението и при „сглобяването” на композиран обект, където има промяна на версията само на обекта, който става суперобект. 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 *подобект* (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 *подобекта*, се получава нова версия на целия стол – стол със синя тапицерия. Като частни случаи на промяна на *подобект* може да се приеме асоциирането на обект като *подобект*, както и премахването на асоциация с *подобект* и неговата (на *подобекта*) 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 суперобекта – не означава, че има промяна във версията на съставящите го *подобекти*. Така че, ако имаме стол с три крака и червена тапицерия, то добавянето на четвърти крак към стола не променя версията (цвета) на *подобекта* тапицерия (Фиг. 31) .



Фиг. 31 Промяната на суперобекта не влияе на версията на подобекта

От последните две правила може да се изведе следствието:

1. Промяна на версията на даден *подобект* за даден *суперобект*, не влияе на версиите на другите *подобекти*, съставящи същия *суперобект* (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 подобект не се променя версията на съседните подобекти

1. Версия на даден съставен обект е видима в дадено работно пространство само и единствено, когато всички версии на съставящите го *подобекти* са видими в съответното работно пространство.

3. Model of Hierarchical composed workspaces

### 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:

1. Продукт се нарича обект на материалното или нематериалното производство, който след своето създаване може да бъде размножен и разпространяван сред клиентите.
2. 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.
3. Работно пространство се нарича място, където се извършват определени дейности по създаването на версия на продукт.
4. Главно работно пространство се нарича работно пространство, в което се извършва окончателната комплектация 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 пространство

### 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

1. *Локална версия* *на версионизиран обект* е версия, която е асоциирана с конкретно работно пространство.
2. Под *видима версия* *на версионизиран обект* за дадено работно пространство се разбира такава версия на обекта, с която потребителят може да работи.

Принципи на видимост:

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.
2. Локалната версия на обект от дадено работно пространство се вижда рекурсивно във всички *подпространства*, освен ако няма дефинирана друга локална версия в тях.

От изложените принципи можем да изведем следствията:

1. Във всяко работно пространство, където обектите нямат локална версия, те са представени с тяхна версия, намираща се в най-близкото родителско работно пространство.
2. 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.

## Транзакции над версионизирани обекти

### 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 *подпространства*. 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.

### 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.

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

### Транзакции над съставни обекти

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

Нека имаме следната ситуация: локална версия на обекта **В** в родителското работно пространство и негова видима версия в текущото работно пространство. В текущото работно пространство се създава *подобект* **А** за обекта **В** (Фиг. 39). При публикуването версията на *подобекта* **А** е възможно да не води до промяна във версията на обект **В** в родителското работно пространство. Въпреки това при последващо публикуване версията на обекта **В** заедно с неговите композиции, в родителското работно пространство ще доведе до автоматично обновяване (в рамките на работното пространство) на композиционната схема на обектите (Фиг. 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 Новосъздаден подобект към суперобект

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

Нека се разгледа ситуацията, когато имаме локална версия на обекта В в родителското работно пространство, която е видима в текущото работно пространство (Фиг. 40). В текущото работно пространство се извършва промяна в подобекта **А**, която води до промяна на обекта В. Именно създаването на нова локална версия на подобекта води до автоматичното създаване на нова локална версия на съставния обект. Следва да се отбележи, че самостоятелното публикуване на новата версия на подобекта в родителското работно пространство би следвало да не се допуска. Това ограничение следва от факта, че наличието на нова версия на подобекта предполага наличието на нова версия на суперобекта (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:

1. Публикуването на версия на локален съставен обект следва да се извършва в комплект с всички локални версии на неговите подобекти, които имат различна версия в родителското работно пространство ( – зелената и жълтата стрелки с №2).

Локалните версии на подобекти 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 **В** в родителското работно пространство, която е видима в текущото работно пространство (Фиг. 41). От композицията на съставен обект се изключва подобект. 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 подобекта, който в новата версия на суперобекта вече не е съставна част от него.



Фиг. 40 Индиректна променена версия на суперобект, породена от нова версия на подобект



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

Нека имаме видим съставен обект **А** с подобект **В**, като обектът **А** и подобектът **В** са локални версии в родителското пространство. От композицията на даден съставен обект **А** премахнем подобекта **В**. След това потребителят модифицира обекта **В**, т.е. създава нова негова версия (Фиг. 42). При така създадената ситуация публикуването новата версия на подобекта **В** би довело до следната конфликтна ситуация: Версия v1 на обект **В** изисква в работното пространство обектът **А** да присъства (видимо или локално) с версия v1.



Фиг. 42 Публикуване на бивш подобект не е възможно, преди публикуване новата версия на бившия суперобект

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

1. Публикуването на версия на обект, който притежава предишна версия, явяваща се подобект 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:

1. Отказът от локална версия на съставен обект следва да се извършва заедно с рекурсивен отказ от локална версия на всички негови подобекти.

4. Methodology framework

Методологията се определя като „система от [принципи](http://bg.wiktionary.org/wiki/%D0%BF%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF) и [средства](http://bg.wiktionary.org/wiki/%D1%81%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B2%D0%BE) за организиране и провеждане на дадена [дейност](http://bg.wiktionary.org/wiki/%D0%B4%D0%B5%D0%B9%D0%BD%D0%BE%D1%81%D1%82)” [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 рискът 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 участниците 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.

### 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 пространства 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. Етапи по създаване на изсквания и архитектура

1. Той създава тестови план и тестови сценарии за проверка на качеството на бъдещата системна функция.
2. 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.
3. Разработчиците отбелязват като работна задача, над която ще се работи в своите пространства тези на изискването (от стъпка 3) и архитектурата (от стъпка 7).
4. Разработчиците реализират новата функционалност на системата.
5. 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. Етапи по създаване на изходен код и тестови сценарии

1. 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 ).
2. При откриване на дефект тестовият инженер създава нов обект за дефект, отбелязва го като работна задача и го публикува до главното работно пространство.
3. 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.
4. Разработчикът отбелязва дефекта като работна задача, над която ще работи.
5. Той разработва поправка и я публикува до главното работно пространство.
6. Тестовият инженер извършва повторна проверка на функционалността и потвърждава, че дефектът е отстранен.



Фиг. 51 Стъпки по откриване и отстраняване на дефект

### 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 изискванията 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

1. 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.
2. 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

1. 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”.
2. Променените тестови сценарии се публикуват до главното работно пространство.
3. Разработчикът, използвайки справката за несъответствие на причинно-следствените връзки, определя в кои файлове от изходния код следва да извърши корекции.
4. 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 диаграмата 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).

5. Conclusion and future work

...