Бежанишвили Лолита

Моделирование процессов обучения коллективным методам разработки программного обеспечения

представлена на соискание академической степени
доктора

Грузинский Технический Университет
Тбилиси, 0175, Грузия
Июнь, 2008

საქართველოს ტექნიკური უნივერსიტეტი

ინფორმატიკისა და მართვის სისტემების ფაკულტეტი

ჩვენ, ქვემორე ხელისმომწერნი ვადასტურებთ, რომ გავეცანით ლოლიტა ბეჟანიშვილის მიერ შესრულებულ სადისერტაციო ნაშრომს დასახელებით: “პროგრამული უზრუნველყოფის შექმნის კოლექტიური მეთოდების სწავლების პროცესების მოდელირება” და ვაძლევთ რეკომენდაციას საქარველოს ტექნიკური უნივერსიტეტის ინფორმატიკისა და მართვის სისტემების ფაკულტეტის სადისერტაციო საბჭოში მის განხილვას დოქტორის აკადემიური ხარისხის მოსაპოვებლად.

თარიღი

ხელმძღვანელი: _____

რეცენზენტი: _____

რეცენზენტი: _____

რეცენზენტი: _____

საქართველოს ტექნიკური უნივერსიტეტი

2008

ავტორი:	ბეჟანიშვილი ლოლიტა
დასახელება:	“პროგრამული უზრუნველყოფის შექმნის კოლექ-ტიური მეთოდების სწავლების პროცესების მოდელირება”
ფაკულტეტი :	ინფორმატიკისა და მართვის სისტემების
აკადემიური ხარისხი:	მაძიებელი
სხდომა ჩატარდა:	

ინდივიდუალური პიროვნებების ან ინსტიტუტების მიერ ზემომოყვანილი დასახელების დისერტაციის გაცნობის მიზნით მოთხოვნის შემთხვევაში მისი არაკომერციული მიზნებით კოპირებისა და გავრცელების უფლება მინიჭებული აქვს საქართველოს ტექნიკურ უნივერსიტეტს.

ავტორის ხელმოწერა

რეზიუმე

საიმედო პროგრამული კომპლექსების შექმნისას განსაკუთრებული მნიშვნელობა ენიჭება პროექტის მართვის სწორ ორგანიზებას. პროგრამული უზრუნველყოფის დაპროექტების მეთოდების და მასთან დაკავშირებულ დაპროექტების პროცესების გაუმჯობესება იწვევს დამუშავების ფასის შემცირებას და პროგრამების გამოჩენას, რომელთა ტექნიკური მხარდაჭერის ხარჯები დაბალია. პროგრამული უზრუნველყოფის დაპროექტების და დამუშავების სწორი მართვა იძლევა დაპროექტებისას დაუგეგმავი შეცდომების და ვადების ჩავარდნის თავიდან აცილების საშუალებას და ამის გარდა პროგრამული პროდუქტების ხარისხისა და საიმედოობის გაუმჯობესების გარანტიას. სწორედ ამ მიზეზის გამო გამოკვლევების წინა პლანზე წამოიწია დასამუშავებელი სისტემის ხარისხის გარანტირების საკითხმა. ამ კონცეფციის მიხედვით შესაქმნელი პროგრამული სისტემის ხარისხი გამოცხადებულ იქნა სისტემის თვისებად, რომელიც უნდა იყოს მართვადი და დასაგეგმი.

გარდა ამისა, ცხადია, რომ პროგრამული უზრუნველყოფის შემქმნელი კოლექტივი უნდა იყოს მზად დამუშავების შემოთავაზებული მეთოდების და ხერხების გამოყენებისთვის. წარმოიქმნა კოლექტიური დამუშავების თანამედროვე მეთოდების შესწავლის საჭიროება, რომლებიც შეიცავენ ცოდნის დაგროვებას წინა ამოცანების ამოხსნის საფუძველზე, მის განზოგადებას და გამოყენებას ახალ, ადრე განუხილავ ამოცანებზე. აქედან გამომდინარე, მასწავლი/კოგნიტიური სისტემის შექმნა წარმოადგენს ძალზედ აქტუალურ ამოცანას ორგანიზაციის შესაძლებლობების წინასწარი შეფასებისთვის და ამის გარდა ორგანიზაციული კულტურის პრინციპების სწავლებისთვის.

სადისერტაციო ნაშრომის მიზანია ფორმალური მოდელების და მეთოდების დამუშავება პროგრამული უზრუნველყოფის შექმნის კოლექტიური მეთოდების მასწავლი სისტემის აგების მიზნით.

ამ მიზნის მიღწევისთვის დისერტაციაში ძირითადი ყურადღება ენიჭება შემდეგი ამოცანების ამოხსნას:

- ცოდნის შიდა წარმოდგენის ფორმალური მეთოდების და საშუალებების დამუშავება;
- პროგრამული უზრუნველყოფის შექმნის პროცესის მასწავლი მოდელის დამუშავება;
- დინამიკური მიდგომის გამოყენება პროგრამული უზრუნველყოფის შექმნის პროცესის შესწავლის მიზნით CMM მოდელის ბაზაზე (Capability Maturity Model – შესაძლებლობათა სიმწიფის მოდელი);
- პროცესების ცოდნის ბაზის და მონაცემთა ბაზის აგება;
- მომხმარებელთან დიალოგის ორგანიზება.

ნაშრომში შემოთავაზებულია სწავლების პროცესები აღიწეროს ლაბირინთული ფორმალური კომბინატორული მოდელით, რომლის კონკრეტიზაცია გვაძლებს სწავლების კონკრეტულ ამოცანებს სხვადასხვა

ჭრილში. ცოდნის გარე წარმოდგენისთვის შემოთავაზებულია ობიექტზე–ორიენტირებული ანალიზისა და პროექტირებისათვის გათვალისწინებული უნივერსალური მოდელირების ენა UML-ის გამოყენება. ცოდნის შიდა წარმოდგენისთვის განიხილება კონტექსტურად მართვადი სემანტიკური ქსელები, რომლებიც შეიძლება აღიწეროს XML-ენის გამოყენებით. სწავლების კონკრეტულ ამოცანებში გამოსაყენებლად შემოთავაზებულია სტრუქტურიზებული ცოდნის წარმოდგენების მოდულური ტრანსფორმაციის მეთოდი.

განისაზღვრება ინფორმაცია, რომელიც საჭიროა პროცესების ცოდნის ბაზის და მონაცემთა ბაზის შევსებისთვის. პროცესების მონაცემთა ბაზა შეიცავს მონაცემებს დამთავრებული პროცესების მწარმოებლურობის შესახებ. იგი შეიცავს მონაცემებს რისკების, შრომატევადობის და მისი განაწილების, შეცდომების და მათი განაწილების, პროგრამული უზრუნველყოფის მოცულობის და პროექტის სხვა მახასიათებლების შესახებ. პროექტის მენეჯერს შეუძლია უკეთესად დაგეგმოს თავისი პროექტი, თუ მისთვის ხელმისაწვდომია დასრულებულ პროექტებში დაგროვებული გამოცდილება. პროექტის დაგეგმის ინფრასტრუქტურა გვეხმარება მონაცემებისა და მიღებული გაკვეთილების ეფექტიანად შეგროვებაში, რათა შემდგომში ისინი ხელმისაწვდომნი გახდენ პროექტების მენეჯერთათვის.

განიხილება პროგრამული უზრუნველყოფის დამუშავების პროცესი დინამიკაში. შემოთავაზებულია CMMI (Capability Maturity Model Integration – ინტეგრირებული შესაძლებლობათა სიმწიფის მოდელი) და PSIM (Process Simulation Modeling - პროცესების იმიტაციური მოდელირება) მოდელების კომპლექსური გამოყენება კოგნიტიურ სისტემაში, რაც გვაძლევს საშუალებას ყურადღება მივაქციოთ პროგრამული უზრუნველყოფის შემუშავების დინამიკას და ამის გარდა სისტემის განვითარებას და წარმართვას.

მატესტირებელი სისტემა CMM მოდელის ბაზაზე შეიძლება იყოს გამოყენებული შემდეგი 4 კატეგორიის სპეციალისტების მიერ:

- ორგანიზაციის შეფასების ჯგუფის მიერ (Assessments Team) – ორგანიზაციის ძლიერი და სუსტი მხარეების გამოვლენისთვის;
- კონტრაგენტების შეფასების ჯგუფის მიერ (Evaluation Team) – კონტრაგენტების შერჩევისას რისკის გამოვლენისთვის და კონტრაქტების შესრულების გაკონტროლებისთვის;
- ხელმძღვანელების და ტექნიკური პერსონალის მიერ – პროცესების გაუმჯობესების ღონისძიებების დაგეგმვის და პრაქტიკაში დანერგვის გაგების ხელშეწყობისთვის;
- პროცესების გაუმჯობესების ჯგუფის მიერ – თავისი მოქმედებების სახელმძღვანელოდ.

სისტემის საინჟინრო ასპექტების რეალიზაციისთვის და მონაცემთა ბაზის დაპროექტებისთვის გამოყენებულია პროგრამული პროდუქტები Microsoft Visual Studio Team Foundation Server и Microsoft SQL Server.

Резюме

При создании высоконадежных программных комплексов важное значение имеет правильная организация управления проектами. Совершенствование методов проектирования программного обеспечения и связанных с ним процессов проектирования способствует оптимизации стоимости разработки и появлению программ, требующих низких затрат на техническую поддержку. Правильное управление проектированием и разработкой программного обеспечения позволяет избежать непреднамеренных ошибок проектирования и срывов сроков проектов, а также повысить качество и надежность программных продуктов. Именно по этой причине на передний план исследований стала выдвигаться концепция гарантии качества разрабатываемой системы. Качество разрабатываемой программной системы в соответствии с этой концепцией объявляется свойством системы, которое должно быть планируемым и управляемым.

Кроме того становится ясно, что коллектив разработчиков должен быть готовым к использованию предлагаемых ему методов и средств коллективной разработки программного обеспечения. Возникла необходимость обучения современным методам коллективной разработки, включающим накопление знаний на основе решения предыдущих задач, их обобщение и применение на новых, ранее не рассмотренных задачах. Таким образом, создание обучающей/когнитивной системы, очень актуальная задача для предварительной оценки возможностей организации а также обучения принципам организационной культуры.

Целью диссертационной работы является разработка формальных моделей и методов для построения системы, обучающей коллективным методам разработки программного обеспечения.

Для достижения данной цели в диссертации, основное внимание уделено решению следующих задач:

- разработка формальных методов и средств внутреннего представления знаний;
- разработка обучающей модели процесса разработки программного обеспечения;
- использование динамического подхода к изучению процесса разработки программного обеспечения на базе модели CMM (Capability Maturity Model – Модель Зрелости Возможностей);
- построение базы данных и базы знаний процессов;
- организация диалога с пользователем

В работе предложено процессы обучения описывать лабиринтной формальной комбинаторной моделью, конкретизация которой дает конкретные задачи обучения под разными углами. Для внешнего представления знаний предложено использование объектно-ориентированного, предусмотренного для анализа и проектирования, универсального

языка моделирования UML. Для внутреннего представления знаний рассматриваются контекстно управляемые семантические сети, которые могут быть описаны с использованием XML-языка. В конкретных задачах обучения для преобразования структуризованных знаний предложен метод модульных трансформаций представлений.

Описана информация, необходимая для заполнения базы знаний и базы данных процессов. База данных процессов содержит данные по производительности завершенных проектов. Она включает данные по рискам, трудоемкости и ее распределению, ошибкам и их распределению, размеру ПО и другим характеристикам проектов. Менеджер проекта может лучше спланировать свой проект, если имеет доступ к опыту, накопленному в завершенных проектах. Инфрастуктура планирования проекта помогает эффективно собирать данные и полученные уроки и делать их доступными для менеджеров проектов.

Рассматривается процесс разработки программного обеспечения в динамике. Предложено комплексное использование CMMI (Capability Maturity Model Integration – Интегрированная Модель Зрелости Возможностей) и PSIM (Process Simulation Modeling - Имитационное Моделирование Процессов) моделей в когнитивной системе, позволяющее сосредоточиться на динамике разработки программного обеспечения, а также развитии и ведении систем.

Обучающая и тестирующая система на базе модели CMM может быть использована по крайней мере следующими 4-мя категориями специалистов:

- группа оценки организации (Assessments Team) – для выявления сильных и слабых сторон организации;
- группа оценки контрагентов (Evaluation Team) – для выявления риска при выборе контрагентов и для отслеживания контрактов;
- руководители и технический персонал – для понимания деятельности по планированию и претворению в жизнь программы совершенствования процесса в их организации;
- группа совершенствования процесса – как руководство в своих действиях.

Для реализации инженерных аспектов системы и проектировании базы данных использованы программные продукты Microsoft Visual Studio Team Foundation Server и Microsoft SQL Server.

Abstracts

Creation of software complexes with high reliability, the correct organization of project management has high importance. Perfection of software design methods and associated design processes promotes optimization of expenses due to low technical support costs. Correct management of software design and - development helps to avoid inadvertent design errors and failures in meeting deadlines, while at the same time raises quality and reliability of the software. For this exact reason, the concept of quality assurance of the developed system has been put forward of the research. In accordance to this concept, quality of the developed software system is said to be its characteristic and has to be plan able and manageable.

Besides, it becomes clear, that the team of software developers should be ready to use offered methods and instruments of collective software development. There is a necessity of training to modern methods of the collective software development, including accumulation of knowledge based on the solutions of previous tasks, generalizing them and applying to new problems. Thus, the creation of training /cognitive systems is a very important task for preliminary estimation of possibilities of the organization and training to principles of organizational culture.

The purpose of the dissertation is the development of formal models and methods for creation of the system able to train to collective methods of software development.

For achieving the given goal, this dissertation prioritizes following tasks:

- Development of formal methods and means for internal representation of knowledge;
- Development of training model of software development process;
- Use of the dynamic approach for studying the process of software development process based on CMM (Capability Maturity Model);
- Construction of a database and the knowledge base of processes;
- The organization of a dialogue with the user

The dissertation offers to describe the training processes by the formal labyrinth combinatory model, which, if specified gives concrete learning problems in different views. For the external representation of knowledge, offered is use of object oriented, for the analysis and projects, UML (Unified Modeling Language). For internal representation of knowledge, contextually operated semantic networks are considered which can be described with XML. In specific tasks of training for transformation of structured knowledge, the method of modular transformations of representations is offered.

Information described is necessary for populating the knowledge base and a database of processes. The database of processes contains data on productivity of completed projects. It includes data concerning risks, labor input and its distribution, errors and their distribution, the size of software and to

characteristics of projects. Project manager can plan better if has access to the experience piled up from completed projects. Project planning infrastructure helps to effectively collect data and lessons and make them accessible to project managers.

Considered is the software development process in dynamics. Complex use of CMMI (Capability Maturity Model Integration) and PSIM (Process Simulation Modeling) is offered in cognitive system, allows concentrating on dynamics of the software development, and also conducting of systems.

The training and testing system based on CMM can be used by at least following four categories of experts:

- Assessments Team of the organization – for revealing strengths and weaknesses of the organization;
- Evaluation Team of the counterparts – for revealing risks while choosing counterparts and for tracing the contracts;
- Management and technical staff – for understanding the planning and implementation of the perfection of process software in their organization;
- Group working on perfection of process – as a management in action.

For engineering issues of system and database designs, following software solutions are used:

- Microsoft Visual Studio Team Foundation Server;
- Microsoft SQL Server.

Содержание

Список таблиц

Список рисунков

Список аббревиатур используемых в диссертации

TQM	–	Total Quality Management (Всеобщее Управление Качеством) ,
TSP	–	Team Software Process (Процесс коллективной разработки программного обеспеченмя),
CMM	–	Capability Maturity Model (Модель Зрелости Возможностей),
CMMI	–	Capability Maturity Model Integration (Интегрированная Модель Зрелости Возможностей),
CMM-SW	–	Capability Maturity Model for Software (Модель зрелости возможностей для программного обеспечения),
SEI	–	Software Engineering Institute (Институт Программной Инженерии),
KPA	–	Key Process Areas (ключевые области процесса)
KP	–	Ключевые практики (Key Practicies)
PSIM	–	Имитационное моделирование процессов,
UML	–	Unified Modeling Language (Универсальный язык моделирования)
XML	–	Extensible Markup Language (Расширяемый язык разметки)
PDB	–	Process Data Base (База данных процессов)
LOC	–	Lines Of Code (Число строк кода)
UUCP	–	Unadjusted Use Case Points (Точки нескорректированных вариантов использования)
TCF	–	Technical Complexity Factor (Коэффициент технической сложности)
EF	–	Environment Factor (Фактор окружения)
UCP	–	Use Case Points (Точки вариантов использования)
TFS	–	Team Foundation Server
XMI	–	XML Metadata Interchange (Обмен метаданных XML)
MOF	–	Meta Object Facility (Средство описания метаобъектов)
XSLT	–	EXtensible Stylesheet Language Transformations (Расширяемый язык стилей трансформаций)

Благодарности.

В благодарность моим близким, которые поддерживали меня всегда и во всём.

Введение

В процессе разработки программного обеспечения, которое должно отвечать требованиям к конфиденциальности, целостности и готовности, выявились серьезные проблемы. Для их решения необходим персонал с соответствующим уровнем образования, подготовкой и опытом. При создании высоконадежных программных комплексов важное значение имеет и правильная организация управления проектами. На практике проблемы управления гораздо чаще становятся причинами крушения проектов, чем неудачные технические решения.

Процессы, используемые при разработке, влияют на результат. Совершенствование проектирования программного обеспечения и связанных с ним процессов должно вести к созданию надежного программного обеспечения. Речь идет о программах, которые придают нам уверенности (хотя и не гарантируют абсолютной защиты от ошибок и полной достоверности), включают в себя функции, необходимые для поддержания надежности на должном уровне, и обладают свойствами, гарантирующими правильность и корректность их использования. Совершенствование процессов проектирования способствует оптимизации стоимости разработки и появлению программ с крайне малым количеством дефектов, требующих низких затрат на техническую поддержку, что влечет за собой общее повышение репутации программных продуктов.

Кроме того, статический подход к процессам разработки программного обеспечения, применяемый в большинстве проектов, доказал свою несостоятельность в нашем быстро меняющемся мире. Поэтому возникает необходимость иного, *динамического*, подхода к изучению процессов разработки программного обеспечения.

Для совершенствования технологии проектирования разработан целый ряд процессов и процессных моделей. Однако в используемых моделях остаются не решенными принципиальные вопросы, связанные с

1. Налаживанием процессов обучения коллективным методам разработки программного обеспечения.
2. Необходимостью динамического подхода к изучению процессов разработки программного обеспечения.

3. Использованием системной динамики как методологии имитационного моделирования сложных систем

Основной целью диссертационной работы является разработка формальных моделей и методов для построения системы, обучающей коллективным методам разработки программного обеспечения в динамике.

Для достижения данной цели в диссертации, основное внимание уделяется решению следующих задач:

- разработка формальных методов и средств внутреннего представления знаний;
- разработка обучающей модели процесса разработки программного обеспечения;
- использование динамического подхода к изучению процессов разработки программного обеспечения;
- построение базы данных и базы знаний процессов;
- организация диалога.

В результате проведенных исследований разработаны формальные методы и модели внутреннего представления знаний; структуры баз данных и знаний; методы обучения процессам коллективной разработки с учетом системной динамики и применения моделирования к процессам коллективной разработки программного обеспечения, диалоговые процедуры.

Диссертационная работа состоит из введения и четырех глав. В первой главе проанализированы сущестующие методы коллективной разработки программного обеспечения, раскрыта актуальность темы и приведены задачи, подлежащие решению в диссертационной работе.

Во второй главе представлена формальная модель обучения, описаны методы представления знаний и разработаны механизмы трансформации представлений, используемые в процессе пополнения знаний системы, а также операции над фрагментами семантической сети, структура баз данных и знаний системы. Кроме того предложено обучение методам коллективной разработки с использованием системной динамики.

В третей главе рассматриваются инженерные аспекты реализации системы, приводятся характеристики проекта и делается обзор областей знаний по управлению проектами и процессов управления проектами, а также нюансы

коллективной разработки с использованием MS Visual Studio Team Foundation Server (TFS).

В четвертой главе приводится практическая реализация основных положений работы, разработан список процессов, необходимых для заполнения базы процессов системы.

1. Анализ задачи моделирования процесса обучения методам разработки программного обеспечения

1.1. Постановка задачи и актуальность

Во всем мире около полумиллиона менеджеров проектов выполняют приблизительно миллион программных проектов в год, производя программное обеспечение общей стоимостью в $600 млрд. Многие из этих проектов не удовлетворяют ожидания заказчиков по качеству, в других не удается разработать программное обеспечение (ПО) в соответствии с бюджетом и в установленные сроки. В одной аналитической работе высказано мнение, что приблизительно треть всех проектов превышает стоимость и отстает по срокам более чем на 125%.

Многие уязвимые места появляются вследствие ошибок, которые непреднамеренно возникают при проектировании и разработке программного обеспечения. Согласно анализу, выполненному специалистами CERT/CC, появление большинства уязвимых мест обусловлено общими причинами (10 таких причин порождают около 75% всех «брешей»), причем источниками более 90% уязвимых мест являются известные типы дефектов. Будем трактовать понятие «дефект» шире, относя к ним все то, что обуславливает необходимость каких-либо изменений продукта (вне зависимости от того, связано ли это с несоответствием требованиям, неудачными конструкторскими решениями, недостаточным уровнем надежности и удобства использования или с ошибками кодирования).

При использовании сегодняшних технологий разработки на каждую тысячу строк нового или измененного кода приходится, как правило, от 1 до 7 дефектов. В типичных современных системах, содержащих миллионы строк кода, имеются тысячи дефектов. Естественно, такое программное обеспечение редко гарантирует надежность. Для создания действительно надежного и качественного программного обеспечения организациям необходимо сократить на один-два порядка число дефектов в спецификациях, ошибок при проектировании и реализации.

За ближайшие 20 лет были разработаны несколько общих методов создания программных систем (Структурное Программирование, Структурный Анализ, Объектно-Ориентированный стиль в проектировании и кодировании и пр.) и мощных инструментальных средств (UNIX, APSE, C++, Forthe,... CASE). При этом считалось, что, овладев этими методами и средствами, любой коллектив разработчиков способен обеспечить потребную производительность при заданном уровне качества создаваемого продукта. Однако, как легко видеть, ничего этого реально не происходило. По-прежнему сроки создания программных систем срывались, а их качество во многих случаях не позволяло заказчткам считать сдаваемую им программную систему той, которую они заказывали.

Почему же столько программных проектов заканчивалось провалом? Хотя причин этого много, одной из наиболее важных является неправильное управление проектом. Например, среди основных причин выхода проектов из-под контроля можно назвать маловразумительные цели, плохое планирование, необходимость использования новых технологий, отсутствие методологии управления проектом и недостаточную численность персонала. По крайней мере три из пяти названных причин очевидно связаны с управлением проектом. Оставшиеся две — недостаточная численность персонала и новые технологии — можно считать рисками, управление которыми также относится к сфере управления проектом. Ясно, что, применяя "эффективные" методы управления проектами, менеджер может увеличить шансы на успех создания качественного программного продукта.

Именно по этой причине на передний план исследований стала выдвигаться концепция *гарантии качества разрабатываемой системы*. Качество разрабатываемой программной системы в соответствии с этой концепцией объявлялось свойством системы, которое должно быть *планируемым и управляемым*. Подавляющее большинство "фирменных" технологий программирования содержали базовые схемы деятельности по планированию и управлению качеством создаваемых систем.

Общая теория менеджмента качества вводит так называемый «треугольник качества», вершины которого — люди, технологии и процессы. Неоспоримо, что для достижения качества требуются квалифицированные специалисты и эффективный инструментарий, но ни то ни другое не даст

желаемого результата при отсутствии грамотно организованного процесса — последовательности шагов, приводящих к реализации поставленной цели. *Качество продукта* находится в *прямой* зависимости *от качества процессов,* которые используются для его создания.

Что же такое процесс? Технически процесс для задачи включает последовательность шагов, которой нужно придерживаться при выполнении данной задачи. которой нужно придерживаться при выполнении данной задачи. "Процесс – это система операций при создании чего-либо,… серия действий, изменений или функций, приводящая к требуемому результату" – словарь Webster'a. "Процесс – это последовательность шагов, выполняемых для достижения определённой цели" – стандарт IEEE-STD-610. Процесс создания ПО может быть определён как набор (не последовательность!) операций, методик, способов и преобразований, используемых персоналом для проектирования, разработки и сопровождения ПО и связанными с ними другими продуктами (планы (разработки, тестирования, верификации, сертификации, управления конфигурациями, управления качеством,..), документация, тесты, программы и вспомогательные материалы для обучения,…).

Однако для любой организации процессы, которые она рекомендует использовать своим инженерам и менеджерам проектов,— не просто последовательности шагов: они заключают в себе то, что инженеры и менеджеры проектов узнали об успешно выполненных проектах. Через процессы преимущества накопленного опыта передаются всем сотрудникам организации, в том числе новичкам. Такие процессы помогают менеджерам и инженерам подражать прошлым успехам и избегать просчетов, ведущих к провалам. Для любого проекта процессы инженерии прежде всего определяют, как выполнить техническую работу, например, создать спецификации требований, осуществить проектирование, тестирование и пр. С другой стороны, процессы управления проектом определяют, как установить контрольные точки, организовать персонал, управлять рисками, отслеживать продвижение вперед и т.д.

• Процессы представляют коллективное знание. Их использование повышает шансы на успех.

• Не пользуясь процессами, невозможно точно предсказать результаты проекта.

• Вы и ваша организация не сможете эффективно обучаться, если не будете пользоваться предопределенными процессами. *А обучение и усовершенствование* — это настоятельные требования современного мира, в котором все основано на знании.

• Процессы снижают уровень опасений. Контрольные перечни неизбежно охватывают 80% того, что нужно сделать. Следовательно, ваша задача сводится к проработке лишь 20%.

Постепенно стало понятно, что сами по себе используемые методы и средства разработки не в состоянии гарантировать требуемый уровень качества. Более того, во многих случаях переход на теоретически более эффективные (и, безусловно, подходяшие к условиям конкретного проекта) технологические приемы приводил к краху проекта. Коллектив разработчиков, оказывается, должен быть *готовым* к использованию предлагаемых ему методов и средств, и в этой области оказались невозможными "революционные" скачки.

Деятельность в программном проекте ведется в основном в двух измерениях: в области инженерии и в области управления проектом. Измерение инженерии имеет дело с проектированием, программированием, тестированием и т.п. Измерение управления проектом связано с надлежащим планированием и контролем действий инженерии, которые позволяют достичь целей проекта по стоимости, срокам и качеству.

Если проект невелик (скажем, один-два человека работают над ним несколько недель), то его можно выполнить до некоторой степени неформально. Планом этого проекта может служить сообщение электронной почты, указывающее дату поставки и, возможно, еще несколько контрольных точек.

Эти неформальные приемы, однако, не подходят для более крупных проектов, над которыми многочисленный коллектив должен работать долгие месяцы, — т.е. для большинства коммерческих программных проектов. В таких проектах каждую техническую задачу нужно выполнять тщательно, следуя испытанным методологиям, а рабочие продукты должны быть хорошо документированы, чтобы другие разработчики могли ознакомиться с

ними. Задачи проекта должны тщательно планироваться и распределяться среди выполняющего проект персонала, а затем отслеживаться по мере выполнения проекта. Другими словами, чтобы успешно выполнять более крупные проекты, следует повысить формализм и строгость обоих названных измерений.

Процесс проектирования начинается с определения спецификаций, влияющих на поведение программ и отражающих актуальные требования к безопасности. Необходимо, чтобы все компоненты системы соответствовали этим условиям. Спецификации должны обеспечивать нужную функциональность и не иметь уязвимых мест. Независимо от того, приобретается ли система на стороне или разрабатывается внутри компании, все ее компоненты обязаны отвечать требованиям надежности.

Еще один момент, касающийся обеспечения надежности, связан с правильностью проектирования и реализации. Программное обеспечение работает правильно только в том случае, когда все выполняемые операции точно соответствуют определенным спецификациям и не допускается ничего лишнего. Сегодня же зачастую программы выполняют действия, о которых не знают даже разработчики и тестировщики.

Следовательно назрела необходимость повышения культуры разработчиков, *обучения* их правильным методам коллективной работы, созданию зрелых процессов и *сохранению и накапливанию знаний* о *процессах* разработки.

Кроме того, следует учесть, что темпы изменений в программных системах продолжают головокружительно расти. Людям, старающимся создать качественное программное обеспечение, приходиться сталкиваться с огромными проблемами в нашем быстро меняющемся мире. На ранних стадиях разработки программного обеспечения разработчик мог заморозить требования к программному обеспечению, разработать программное обеспечения для данных требований и быть уверенным, что по прошествии двух лет результирующее программное обеспечение с этими требованиями все-еще уместно и корректно. Большинство процессов, методов и инструментов программной инженерии было разработано и использовано с допущением об относительной стабильности требований. Однако, в современную динамичную эпоху это непреемлемо. Необходим другой,

динамический подход к созданию систем. *Процесс и динамика проекта* имеют *решающее значение* для управления и совершенствования процесса разработки программного обеспечения.

Создание *моделей обучения* процессам проектирования и разработки программного обеспечения с возможностью моделирования их динамики, *структуризации и накопления знаний* в этой области является *актуадьной задачей* и дает возможность создания модели обучения коллективным методам разработки программного обеспечения, где эффективно и просто происходит аккумуляция знаний и в последствие динамическое их использование в процессе обучения.

1.2. Обзор и анализ существующих моделей процесса разработки программного обеспечения

Претензии потребителей к качеству программных продуктов и обусловили необходимость в создании специфических для этой отрасли моделей управления качеством. Для совершенствования технологии проектирования разработан целый ряд процессов и процессных моделей. Особое внимание уделяется тем из них, которые позволяют существенно уменьшить количество ошибок при разработке спецификаций программного обеспечения, его проектировании и внедрении, минимизировать число уязвимых мест.

Ужесточение требований к устойчивости и надежности привело к тому, что появился ряд подходов к построению высоконадежных программных систем.

- **Коллективное проектирование**

Процесс коллективного проектирования программного обеспечения (Team Software Process, TSP) был определен специалистами института Software Engineering Institute (SEI), созданного при университете Карнеги-Меллона. Данный процесс представляет собой использование при коллективной разработке определенного набора операций. В настоящее время процесс TSP задействуется достаточно широко. Проведенное недавно исследование 20 проектов в 13 организациях показало, что применяющие его

команды выпускают программное обеспечение, в котором имеются около 0,06 дефектов проектирования и реализации на каждую тысячу строк нового и измененного кода. Сроки выпуска этих продуктов по сравнению с намеченными увеличиваются в среднем на 6%. Такой результат весьма неплох, если учесть, что более половины программных проектов завершаются неудачей, либо на их реализацию уходит минимум вдвое больше времени, чем планировалось [3].

Процесс коллективного проектирования надежного (безопасного) программного обеспечения (TSP-Secure) дополняется различными процедурами обеспечения надежности на протяжении всего жизненного цикла разработки. Разработчики проходят дополнительную подготовку в области защиты, изучая общие причины возникновения уязвимых мест, ориентированные на надежность методы проектирования (в частности, проектирование устойчивых машин и анализ их надежности), соответствующие методы реализации (такие как контрольные ведомости надежного кода) и тестирования. Процесс TSP-Secure относительно нов, но ведущие производители уже активно используют его для создания практически свободного от дефектов программного обеспечения; по крайней мере, многомесячный аудит одной из эксплуатируемых систем не выявил в ней дефектов, влияющих на надежность [1].

- **Структурная корректность**

Формальные методы представляют собой математически обоснованные подходы к созданию программного обеспечения. При этом математические модели и формальная логика используются для поддержки строгих программных спецификаций, корректного проектирования, кодирования и контроля качеством. Один из таких способов — обеспечения структурной корректности (Correctness-by-Construction) — был предложен компанией Praxis Critical Systems [4]. Он позволяет с помощью формальных методов проверять программное обеспечение и своевременно устранять в нем дефекты на протяжении его жизненного цикла.

Данный способ включает в себя определение формальных нотаций для спецификаций систем и компонентов архитектуры с учетом их согласованности и корректности. Для надежных систем выделяются категории состояний системы и операций, которые определяются с учетом их

влияния на общую безопасность. Конечной целью является создание архитектуры, которая позволяет свести к минимуму число функций, критически важных для обеспечения защиты, и изолировать их. Это способствует дальнейшему снижению стоимости и сокращению объема работы (возможно, значительно более трудоемкой), связанной с проверкой правильности данных элементов.

Использование метода структурной корректности позволило с 1992-го по 2003 годы завершить пять проектов почти без дефектов: на 1 тыс. строк кода пришлось 0,04-0,75 ошибок. Надо сказать, в двух из этих проектов к системе безопасности предъявлялись весьма серьезные требования.

- **«Стерильная комната»**

Проектирование программного обеспечения методом «стерильной комнаты» (он назван по аналогии с высокоточным проектированием стерильных комнат, предназначенных для производства аппаратуры) представляет собой теоретически обоснованный, ориентированный на командную разработку процесс создания и сертификации корректных программных систем со статистическим контролем качества [5-7]. Данный процесс охватывает весь жизненный цикл разработки, включая управление проектом в ходе итерационного проектирования, определение спецификаций функций и архитектуры, проверку правильности функционала, а также статистическое тестирование при сертификации программ на пригодность к использованию. Роли сотрудников, которые выполняют соответствующие проекты, распределяются так: отдельные коллективы занимаются определением спецификаций, непосредственно разработкой и сертификацией. Проектирование методом «стерильной комнаты» предусматривает статистический контроль качества в ходе разработки, который осуществляется путем последовательного отделения процедуры проектирования от процедуры статистического тестирования.

Многие проекты, основанные на методе «стерильной комнаты», связаны с секретными сведениями и не фигурируют в общедоступных отчетах. Однако опыт показывает, что общее число ошибок в данных случаях — примерно 0,1 дефекта на 1 тыс. строк для приложения, полностью соответствующего принципам «стерильной комнаты», и 0,4 ошибки для приложения с частичным соответствием. Большинство дефектов в

программном обеспечении, построенном с использованием метода «стерильной комнаты», относится к ошибкам кодирования. Проблемы, связанные со спецификациями и архитектурой, встречаются гораздо реже.

- **Процессные модели**

Процессные модели содержат определения целей и ключевые атрибуты конкретных процессов (например, построения системы безопасности), но не включают в себя практических руководств по определению и реализации процессов. Среди наиболее известных моделей совершенствования процессов можно выделить модель **Capability Maturity Model** (**CMM**), отражающую зависимость возможностей системы от ее зрелости. Хотя об отказе от оценки продукта или сертификации системы речь не идет, методы CMM можно использовать для общей оценки эффективности работы организации (в зависимости от определенных в модели критериев) и поиска путей ее повышения. Практика свидетельствует, что применение процессных моделей для улучшения качества программного обеспечения позволяет существенно сократить количество дефектов архитектуры и реализации в конечных продуктах [8, 9].

Универсальная концепция качества **Total Quality Management** (**TQM**) была сформулирована в 50-е годы в Японии и к началу 80-х получила признание в западной индустрии. TQM устанавливает общие принципы, следование которым позволяет компаниям формировать организационную культуру, обеспечивающую создание качественных продуктов и сервисов. Однако требования к системе управления качеством являются универсальными, а потому требующими определенной интерпретации, когда дело доходит до внедрения в конкретных компаниях.

Министерство обороны США, которое забило тревогу в связи с огромным количеством проблем, возникающих при использовании программных систем военного назначения, стало первым заказчиком новой модели качества. Разработкой модели занялся институт Software Engineering Institute университета Карнеги–Меллона и Mitre Corporation. В 1991 году появилась первая версия модели зрелости процессов разработки программного обеспечения (**Capability Maturity Model for Software**, **CMM-SW**).

CMM определяет основные группы процессов разработки, формулирует характеристики разных уровней зрелости этих процессов и дает практические рекомендации по совершенствованию процессов для достижения ими определенного уровня. CMM обеспечивает среду для измерения (**framework**) и организации эволюционного совершенствования процесса создания программного обеспечения. На базе CMM сложилась инфраструктура оценки и сертификации компаний-разработчиков на соответствие тому или иному уровню зрелости. Стимулом для сертификационных процессов стало требование Пентагона к своим подрядчикам и субподрядчикам, обеспечивающим поставки программного обеспечения, получить формальную оценку уровня зрелости не ниже третьего.

Для упорядочивания различий между подготовленной и неподготовленной организациями с точки зрения организации производства была создана структура процессов создания ПО для зрелой организации. Эта структура описывает эволюционный путь из хаоса случайных процессов к дисциплинированной технологии с выверенными и документированными процессами. Эта структура основывается на объединении концепций процессов создания ПО, возможностей по его созданию, методик и производственной культуры.

Зрелость процессов – это степень их управляемости, возможности количественной оценки, контролируемости и эффективности. Повышение зрелости означает потенциальную возможность роста устойчивости процессов и указывает на степень эффективности и согласованности использования процессов создания/сопровождения ПО в рамках всей организации. Реальное использование процессов невозможно без их документирования и доведения до сведения всего заинтересованного персонала, без постоянного контроля и совершенствования. Возможности хорошо продуманных процессов полностью определены.

В организациях, достигших зрелости, процессы создания/сопровождения ПО принимают статус стандарта, фиксируются в организационных структурах и определяют производственную тактику и стратегию. Введение их в статус закона с необходимостью влечёт построение инфраструктуры и

создание требуемой корпоративной культуры производства, обеспечивающей поддержку соответствующих методов, операций и процедур ведения дел.

Развитие CMM и расширение областей ее применения привело к появлению в 2001 году нового варианта модели — **Capability Maturity Model Integration** (**CMMI**). Она описывает уровни зрелости процессов не только для программной, но и для системной инженерии, интегрированной разработки продуктов и процессов, выбора поставщиков. К настоящему времени SEI перестал поддерживать модель CMM, и сертификация компаний проводится только по CMMI.

Модель предлагает *пять уровней зрелости процесса разработки,* которые являются базовыми для последовательного улучшения важнейших характеристик процесса разработки. Они образуют своеобразную шкалу измерения зрелости процесса в конкретной организации и служат для оценки возможностей этой организации.

Уровень зрелости (**maturity level**) - это хорошо определенная эволюционная ступень на пути к совершенному процессу разработки. Каждый уровень является базисом для достижения следующего уровня. В этом смысле перепрыгивание через уровень, если вообще возможно, практически неэффективно и потому нецелесообразно.

Каждый уровень содержит набор *целей*. Достижение цели означает появление и устойчивую реализацию важного компонента (свойства) разработки. Достижение уровня означает внедрение набора компонент в практику разработок, проводимых организацией, что ведет к увеличению возможностей организации.

Уровни зрелости CMMI

Рассмотрим краткую характеристику уровней.

1) ***Initial*** *(Начальный).* Четкая организация процессов отсутствует, процессы непредсказуемы, нет определенных правил и процедур разработки, процессы являются уникальными, реализуемым во всех своих деталях только в конкретной разработке (следующая разработка, даже если она во многом подобна предыдущей, реализуется по-иному). По фиксированным методикам выполняется только небольшое число элементарных действий процесса разработки. Успех разработки зависит от энтузиазма исполнителей, их

фактической (для условий конкретного проекта) квалификации, трудолюбия и иных профессиональных и личных качеств.

2) ***Repeatable*** *(Повторяемый).* Отработаны базовые процессы управления каждым проектом в целях отслеживания фактической стоимости, план-графика и получающегося качества результатов разработки. Процессы определяются и документируются. Достигнута необходимая дисциплина процесса, позволяющая повторять основные характеристики разработки в новых (схожих) проектах.

3) ***Defined*** *(Определенный).* Процесс создания ПО как с точки зрения управления им, так и с точки зрения производственной технологии продокументирован, стандартизован и интегрирован в стандартный (для данной организации) процесс. Все проекты используют утвержденную версию процесса для разработки и сопровождения ПО.

4) ***Managed*** *(Управляемый).* Регулярно собираются результаты детальных измерений характеристик процесса создания ПО и его качества. Как сам процесс, так и продукт его деятельности изучаются и контролируются с точки зрения количественных оценок. Процессы становятся предсказуемыми и появляется возможность управлять такими параметрами проекта, как количество ошибок, трудоемкость, объем переработок и т.д.

5) ***Optimizing*** *(Оптимизирующий).* Имеется возможность постоянного и целенаправленного улучшения процесса разработки ПО за счет количественной обратной связи от процесса, а также за счет внедрения новых идей и технологий. Процессы находятся в состоянии постоянного совершенствования, компания может внедрять существенные инновации в процессы на основе анализа количественных показателей, идентифицировать корневые причины проблем проектов и предотвращать их появление.

Следуя рекомендациям CMM/CMMI, компании выстраивают процессы управления требованиями и планирования проекта, мониторинга и контроля над проектом, управления качеством и конфигурацией, выбора технических решений и т.д. Кроме того, они получают инструмент совершенствования процессов: определяя характеристики процессов на разных уровнях зрелости, эти модели задают вектор развития организации для достижения более высоких уровней. В итоге продукция сертифицированной компании будет дешевле, чем у организации, процессы

которой соответствуют первому уровню, поскольку эффективная организация процессов разработки позволяет затрачивать меньше усилий на получение результата.

Внутренняя Структура Уровней Зрелости

Каждый уровень зрелости состоит из нескольких ключевых областей (key process areas). Каждая ключевая область содержит 5 групп (группы называют common features - общие характерные черты) ключевых практик (key practicies). Воплощение всех ключевых практик для данной ключевой области обеспечивает достижение целей (goals) этой области. Ниже эти понятия рассматриваются подробнее.

Ключевые Области (Key Areas)

Это - набор взаимосвязанных действий (**ключевых практик**), коллективное выполнение которых приводит к достижению целей данной области. Каждый уровень зрелости (кроме первого, являющегося исключением) содержит свой набор ключевых областей. Если по проектам, выполняемым в некоторой организации, достигнуты цели данной ключевой области, это означает: возможности, характеризующиеся данной ключевой областью, для процесса разработки в данной организации достигнуты. Если такой уровень достигнут для всех ключевых областей данного уровня, это значит: Процесс (организация) соответствует этому уровню зрелости.

Ключевые области выделены потому, что их решение проблем, связанных с ними, наиболее эффективно улучшает процесс создания программного обеспечения. Эти проблемы - требования по достижению уровня зрелости. Чтобы "покрыть" ключевую область, надо достичь всех ее целей.

Ключевые практики, относящиеся к данной ключевой области, будут плавно видоизменяться по мере достижения все более высоких уровней. Например, Integrated Software Management (уровень 3) получается путем эволюции Software Project Planning и Software Project Tracking and Oversight (уровень 2) при переходе с уровня 2 на уровень 3.

Общие свойства (Common Features)

Некоторые ключевые области содержат десятки ключевых практик. Работать к таким большим числом практик неудобно. Для удобства все

ключевые практики, принадлежащие каждой из ключевых областей, разбиты на 5 типовых групп (**common feature**).

Commitment to perform (Обязательства по выполнению). Действия, которые должны быть предприняты для того, чтобы гарантировать установку и дальнейшее поддержание процесса создания ПО. Обычно такие действия включают выработку организационной политики и поручительства руководителей.

Ability to perform (Возможность выполнения). Предварительные условия, которые должны существовать для того, чтобы процесс мог бы быть реализован. Имеются в виду ресурсы, организационные структуры, обучение персонала.

Activities performed (Выполняемые действия). Действия, необходимые для воплощения ("покрытия") ключевой области: подготовка планов, выполнение работ, их отслеживание, проведение необходимых коррекций. Эти действия являются основными. Именно они покрывают ключевую область и приводят к расширению возможностей коллектива/процесса. Ключевые практики других групп (Commitment to Perform, Ability to Perform, Measurement and Analisys, Verifying Implementation) всего лишь создают базис для Activities Performed.

Measurement and Analisys (Измерения и Анализ). Проведение измерений процесса создания ПО и анализ его результатов. Они необходимы для понимания того, как реализуются действия, необходимые для воплощения ключевой области, и каков эффект от этого.

Verifying Implementation (Проверяемая/доказательная реализация). Шаги, предпринимаемые для того, чтобы убедиться: действия по совершенствованию процесса разработки ПО выполняются правильно. Обычно для этого руководители и служба качества ПП (Software Quality Assurance - SQA) проводят/организуют просмотры, инспекции и проверки.

Ключевые практики (Key Practicies)

Каждая ключевая область описывается в терминах ключевых практик, необходимых для достижения целей этой ключевой области. Ключевые практики описывают инфраструктуры и действия. Ключевые практики описывают, ЧТО должно быть сделано, но не обязаны описывать, КАК должны достигаться цели ключевой области. Для достижения целей могут

использоваться альтернативные практики. В этом случае вопрос о выборе пути достижения цели должен решаться на основе здравого смысла.

На сегодняшний день уже около 2 тыс. организаций воспользовались преимуществами модели CMMI при определении путей совершенствования процессов и оценки возможностей программного обеспечения.

- **Технические решения**

Некоторые уязвимые места в системе надежности возникают просто по недосмотру. Это приводит к появлению дефектов, которые вполне можно устранить, если придерживаться хорошо зарекомендовавших себя методов проектирования. Появление ряда других брешей обусловлено конкретными особенностями моделирования, архитектуры и проектирования системы надежности, в том числе ошибками при определении угроз, неадекватной аутентификацией, неверной авторизацией, неправильным применением шифрования, ошибками при организации защиты данных и недостаточно тщательным планированием разделения приложений. Для решения всех этих проблем организациям необходимо использовать эффективные методы, позволяющие решать вопросы обеспечения надежности напрямую.

Приведенный перечень испытанных на практике принципов и хорошо известных методов нельзя назвать исчерпывающим, однако он получился достаточно представительным. Большая часть технических методов требует опыта работы с системами обеспечения надежности, особенно при определении спецификаций и проектировании. У многих из применяемых для разработки надежного программного обеспечения технических методов очень мало практических подтверждений эффективности, но насколько это возможно, здесь приведены примеры из реальной жизни.

- **Руководства разработчиков и контрольные ведомости**

Выстраивая свою деятельность на основе указанных основных принципов, разные организации (от Microsoft до SAP) взяли на вооружение несколько универсальных руководств, в подготовке которых принимала участие и группа OWASP. В процессе создания надежного программного обеспечения разработчики должны:

- проверить правильность входной и выходной информации;
- обеспечить защиту от сбоев системы надежности;
- избегать усложнения системы;

- применять и повторно использовать надежные компоненты;
- организовать глубоко эшелонированную защиту;
- избегать так называемой "надежности за счет запутанности";
- предоставлять только абсолютно необходимые привилегии;
- использовать обособление для разделения прав доступа;
- не применять алгоритмы шифрования собственной разработки;
- использовать шифрование при организации всех видов связи;
- никогда не пересылать пароли в открытом виде;
- обеспечить надежность конфигурации, настраиваемой по умолчанию;
- гарантировать надежность доставки и установки программного обеспечения;
- не создавать "потайных ходов".

Все эти правила обязательно должны соблюдаться при разработке спецификаций программного обеспечения, в процессах проектирования, внедрения, тестирования, доставки и установки.

- **Моделирование угроз**

Моделирование угроз представляет собой методологию анализа защиты, которую можно использовать при определении рисков и принятии решений при построении архитектуры, кодировании и тестировании. Данная методология применяется преимущественно на начальных стадиях реализации проекта (к которым относятся разработка спецификаций, архитектурных представлений, диаграмм потоков данных, функциональных диаграмм и т.д.), но ее можно использовать и при написании детально проработанного технического задания, а также кода. Конечная цель заключается в устранении потенциальных угроз приложению, способных нанести наибольший ущерб.

В целом моделирование угроз предполагает декомпозицию приложения и определение его основных свойств с последующим выявлением и классификацией угроз, направленных против каждого свойства или компонента. Угрозы классифицируются в зависимости от степени риска. В результате формируются стратегии ликвидации угроз на этапах проектирования, кодирования и тестирования.

Однако каждая из этих моделей рассматривает процесс проектирования односторонне. Нет единой модели, объединяющей

достоинства всех этих моделей и обеспечивающей *обучение* и *накопление знаний* о проекте и процессов проекта.

1.3. Основные принципы проектирования надежного программного обеспечения

Исходя из вышесказанного можно сформулировать основные принципы создания качественного программного обеспечения. Конечно, для создания надежного программного обеспечения недостаточно лишь соблюдения принципов, но они способны помочь определиться с методами разработки. Восемь принципов разработки, сформулированные еще в 1974 году [10], сегодня отнюдь не менее актуальны:

- *экономия механизмов* - архитектура должна быть как можно более простой и компактной;
- *действия по умолчанию, обеспечивающие защиту от сбоев*, - решения о предоставлении доступа необходимо принимать на основе назначения прав, а не их исключения;
- *полный контроль* - предоставление доступа к любому объекту должно осуществляться только после проверки наличия соответствующих прав;
- *открытая архитектура* - ее не следует держать в тайне;
- *разделение привилегий* - механизм защиты, требующий двух ключей, более гибок и устойчив по сравнению с применяющим для получения доступа один ключ. Поэтому там, где возможно, лучше использовать механизм с двумя ключами;
- *минимизация привилегий* - любая программа и любой пользователь системы должны иметь минимально необходимый для выполнения стоящих перед ними задач набор привилегий;
- *максимальное сокращение количества общих механизмов* - механизмов, доступных или зависимых более чем от одного пользователя;
- *приемлемость с психологической точки зрения* - необходимо продумать особенности человеко-машинного интерфейса, которые

обеспечат простоту использования программного обеспечения, чтобы потребители, не задумываясь, корректно применяли механизмы защиты при выполнении своих повседневных задач.

1.4. Исследование возможностей существующих систем обучения.

На современном этапе развития общества в результате быстрого развития компьтерных технологий во все сферы общественной деятельности, в том числе и в обучение, ворвались информационные технологии. Образовалось новое направление обучения – электронное обучение, которое может проходить как на месте, так и дистанционно с участием учителя, либо без его участия. Преимущества перехода на новую технологию обучения очевидны и не вызывают сомнений, но, как правило, три основные компоненты процесса обучения: подготовка материала; передача; контроль, рассматриваются независимо друг от друга, не существует единых механизмов управления процессом обучения, что часто ухудшает гибкость и эффективность учебного процесса. Кроме этого, каждый из этих процессов для последующего этапа времени характеризуется разным уровнем автоматизации. Если процесс передачи материала и вопросы контроля сравнительно автоматизированы, то автоматизация вопросов подготовки и управления (составления, разбиения, структуризации, обобщения, выбора последовательности вопросов, подключения тестов по темам) находятся на сравнительно низком уровне. Конечно, это обусловлено сложностью вопроса, который представляет одну из наисложнейших задач искусственного интеллекта – *задачу представления и структуризации знаний*.

Полонота и глубина переданных знаний может быть проверена путем тестирования. В этом понимании даже составление хороших тестов, средней размерности и сложности для предметной области знаний представляет задачу экспоненциальной сложности. Поэтому автоматическая генерация тестов согласно тематике представляет актуальную задачу.

В последние годы интенсивные усилия направлены на создание практических программных автоматизированных систем управления

(Learning Management System) процессами обучения (Moodle- модульная объектно-ориентированная обучающая система, WebCT и другие). В этих системах в основном развиты механизмы планирования процессов обучения, подготовки учебных материалов и «ручной» подготовки и визуального отображения тестов, проведения тестирования и автоматической оценки, механизмы получения статистической информации под разными углами. Соответственно, они представляют более системы с управлением на основе внутреннего содержания (Content Management System), чем системы управления на основе знаний.

Публикации в этой области, в основном, относятся к сфере технологии программирования. Их условно можно разбить на следующие направления:

- представление знаний посредством объектно – ориентированных моделей и языков (UML – язык унифицированных моделей);
- трансформация представлений знаний;
- генерация тестов;
- системы управления знаниями.

В предложенной работе осуществляется обобщение использованных в публикациях идей и перенос их на процесс обучения коллективным методам разработки программного обеспечения.

1.5. Краткий обзор методологий имитационного моделирования процессов (PSIM).

Сделаем краткий обзор различных методологий PSIM, которые были применены к области разработки программного обеспечения и системам [18]:

• дискретное моделирование событий (DES)

• динамические системы (SD)

• гибридное моделирование (HS)

• моделирование, основанное на состоянии (SBS)

• моделирование, основанное на агентах (ABS)

• системы основанные на знаниях (KBS)

• моделирование качества (QS),

Обычно используются методы DES и SD. Третий метод, который небезинтересен - HS, который комбинирует DES и SD в одну модель, чтобы достигнуть преимуществ обоих методов.

Надо отметить, что моделирование процессов являются количественным. Процессное моделирование использует графическое описание технологического процесса как основание для того, чтобы добавить количественные метрики и создать модели для предсказания работы процесса.

Дискретное моделирование событий (DES): модели DES являются дискретными и динамичными. Они могут использовать и детерминированные и стохастические (случайные) параметры, позволяющие моделирование Монте Карло. Модели DES представлены через сеть действий и дискретных (индивидуальных) пунктов, которые текут через эту сеть. Действия преобразовывают пункты и обновляют состояние системы. Модели DES типично используют большое количество различных образцовых элементов и иногда полагаются на программирование. Инструменты DES типично имеют сильные графические способности и позволяют большую гибкость в форме уравнений и случайных переменных, которые могут использоваться. У моделей DES есть исключительная способность смоделировать богатство процессов и заданий, рабочих пакетов и ресурсов, что позволяет количественно оценить проектную работу.

Динамические системы (SD): моделирование SD - тип непрерывного моделирования системы. Это является также динамичным процессом и использует ряд отличительных уравнений, чтобы вычислить поведение системы или процесса в течение долгого времени. Систематически варьируя параметрами пользователи могут получать разумную изменчивость в результатах, но SD не является неотъемлемо стохастическим в природе. В отличие от моделей DES типично представляет процессы на более высоком уровне (меньше деталей). Сила моделей SD - их способность к образцовым петлям обратной связи, но модели SD обычно не включают процесс стохастичности [17]. Инструменты SD типично обеспечивают хорошие графические способности к представлению петель обратной связи. Кроме того, недавние инструменты обеспечивают элементарную способность включить некоторые стохастические случайные переменные. Наконец,

динамические системы - методология моделирования для того, чтобы моделировать непрерывные системы. Количества выражены как уровни, нормы, и информационные связи, представляющие петли обратной связи. Уровни представляют накопления реального мира и служат переменными состояния, описывающими систему в любом пункте вовремя (например, количество развитого программного обеспечения, число дефектов, число персонала на команде, и т.д.), Нормы - потоки в течение долгого времени, которые затрагивают уровни для описания предварительного просмотра образцовых элементов.

Гибридное моделирование (HS): модели HS комбинируют два или больше метода моделирования. В моделировании процесса программного обеспечения HS типично обращается к моделям моделирования, которые комбинируют SD с DES. Гибридные модели широко не используются из-за ограниченной поддержки инструмента и увеличенной сложности. В результате модели HS являются и динамичными и стохастическими, и в состоянии отразить петли обратной связи.

Моделирование, основанное на состоянии (SBS): Модели SBS - другой тип динамического моделирования. Модели SBS одобряют представление системы через конечные автоматы. Эта техника использует деятельность и диаграммы состояния, чтобы смоделировать поведение процессов. Модели SBS могут представить богатство процессов, но не то же самое богатство ресурсов и петли обратной связи как могут быть сделаны с DES или моделями SD. В то же самое время, модели SBS являются превосходными при представлении параллельных действий и параллелизма и могут быть стохастическими. Инструменты SBS типично имеют исключительные графические способности и позволяют и статическую и динамическую образцовую проверку. Модели SBS являются поддающимися анализу, многоуровневыми, и включают многократные перспективы. В настоящее время, они широко не используются для моделирования процесса.

Основанное на доверенных лицах (“агентах”) моделирование (ABS): модели ABS представляют системы и процессы как поселения автономных взаимодействующих параллельных “агентов”. Каждый агент управляет простым сводом правил, которые описывают, как он взаимодействует с другими агентами и окружающей средой. Правила могут быть

детерминированными или стохастическими. Модели ABS - очень хорошо разветвляются, и могут представить детальные взаимодействия между индивидуальными юридическими лицами. Модели ABS только начинают применяться к системам и разработке программного обеспечения.

Системы, основанные на знаниях или основанные на правилах (KBS или RBS): RBSs - тип искусственного интеллекта (AI) система или "экспертная система". Эти системы используют свод правил в соединении с двигателем/механизмом вывода, чтобы представить экспертное знание и подражать процессам рассуждения эксперта. В отличие от предыдущих пяти методов, RBSs полагаются больше на текстовые/логические описания. Они прежде всего использовались для понимания процесса и образовательных целей.

Моделирование качества (QS): QS основывается на образцовом программном обеспечении эволюционных процессов и тенденций. Модели QS работают при условии, что информация о программном обеспечении и процессах развития систем нечетка. Модели QS приспосабливают это, представляя важные образцовые параметры как уравнения, которые просто увеличиваются, уменьшаются, или пребывание то же самое в течение долгого времени. Приблизительная величина увеличения или уменьшения также смоделирована. Использование этой информации позволяет быть замеченными общим тенденциям и поведению на стадии становления системы В целом, у каждого подхода есть различные преимущества и ограничения. Выбор типа моделирования будет зависеть от области системы, которая моделируется и типы вопросов, к которым нужно обратиться.

1.6. Выводы по первой главе.

1. Моделирование процессов обучения коллективным методам разработки программного обеспечения является актуальной задачей. Однако в используемых моделях остаются не решенными принципиальные вопросы, связанные с налаживанием процессов обучения коллективным методам разработки программного обеспечения, необходимостью

динамического подхода к изучению процессов разработки программного обеспечения.

2. В Грузии, обучение и сертификация по CMM/CMMI пока носит эпизодический характер. Проблема состоит в том, что проводить официальное обучение получают право только инструкторы, имеющие авторизацию SEI, а для сертификации необходимо обращаться в соответствующие организации. Ее реализация связана с рядом серьезных проблем, прежде всего с жесткой лицензионной политикой американского университета. Поэтому очень актуальна проблема создания тестирующей системы для оценки возможностей организации, а также обучения принципам организационной культуры.

2. Формирование модели процесса обучения методам разработки программного обеспечения

2.1. Формальная модель обучения

В сфере искусственного интеллекта уже существуют традиционные методы структуризации знаний и информации, разработки языков и инструментальных средств. Знания отличаются от информации тем, что они помимо описания информации в виде фактов, представляют средства ее использования в определенном контексте и достижения конкретных целей. Соответственно, для представления знаний необходимо выработать и исследовать механизмы представления как семантики, так и прагматики. Существует два основных метода представления знаний: декларативный и процедурный. В декларативном представлении знания можно описать в виде структур семантических сетей, фреймов, объектов или правил вывода, обладающих соответствующими языками представления. Несмотря на приложенные усилия, желаемые результаты не были достигнуты, основной причиной чего является постановка задачи. В общем, задача структуризации знаний из-за неоднозначности, неопределенности и большой размерности представляет неразрешимую задачу, поэтому желательно, чтобы эта задача была обсуждена всегда с определенной прагматикой, в определенном контексте, с возможностью управления разной детализацией. Необходимо представление одних и тех же знаний под разным углом и среди них разработка механизмов трансформации и, соответственно, автоматизированных систем.

Чем более структуризованно опишутся знания, тем больше шанс успешного завершения процессов обучения и использования знаний.

В течение всего периода интеллектуальной деятельности человека задача обучения, представления и структуризации знаний всегда была актуальной и останется еще надолго. Поэтому даже небольшие результаты, достигнутые в этой сфере будут важны для будущего развития этой области.

Обучение – это неотъемлемое свойство любой интеллектуальной системы, как естественной, так и искусственной, включающее опыт –

накопление знаний на основе решения предыдущих задач, их обобщение и применение на новых, ранее не рассмотренных задачах. Любой процесс обучения относится к какой-то части реального мира, которую называют предметной областью и описание которой должно быть выполнено на каком-либо языке, например на естественном, или на каком-либо формальном языке. Во всех случаях основными единицами знаний являются понятия и отношения между ними (или абстрактное знание) и факты-доказательства в виде предложений, которые описывают свойства объектов, состояния и отношения предметной области. К тому же описание предметной области на каком-либо формальном языке, всегда подразумевает ее соответствие с описанием на естественном языке. Соответственно с введением формального описания следуем из одного описания к другому к необходимости однозначного переноса.

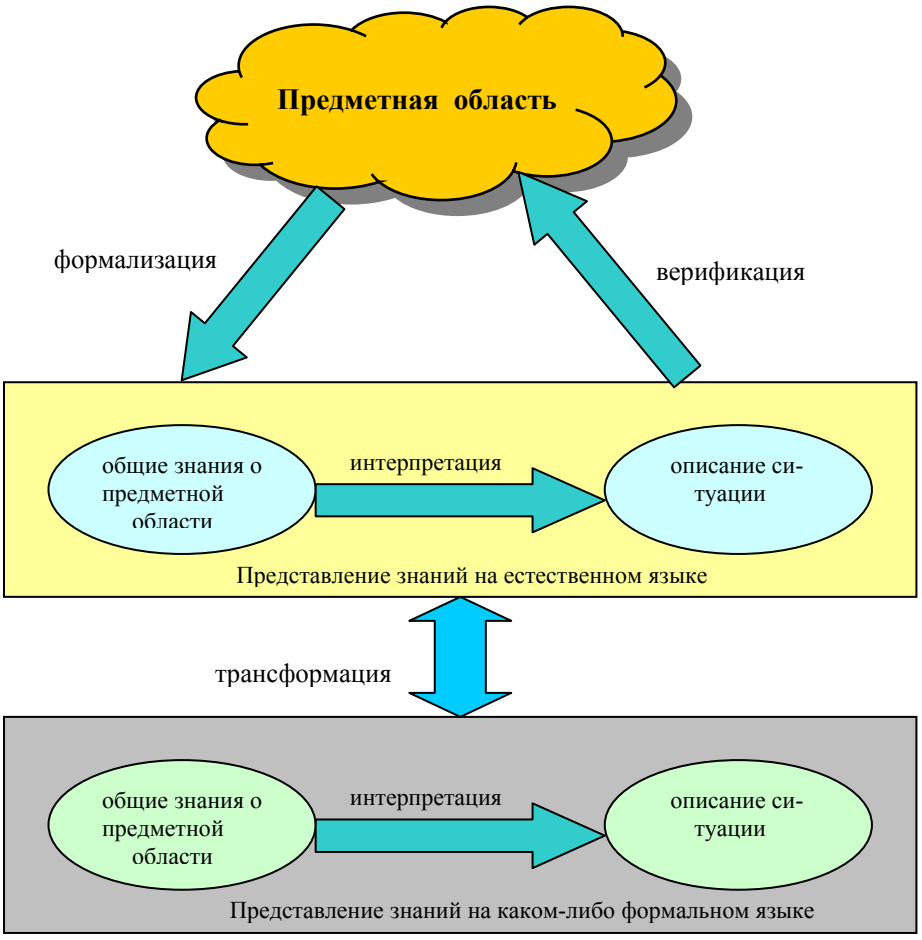


Рис. 1. Формальная модель представления знаний в обучающей системе.

Рассмотрим формальную модель обучения [21], которая основана на модели лабиринтного обучения когнитивной психологии, в которой процесс обучения рассматривается как поиск возможных альтернатив в лабиринте и оценки движений по лабиринту исходя из локальных критериумов.

Введем следующие обозначения:

A – множество компонентов абстрактных знаний, описывающих предметную область;

P- множество доказательств для описания состояния предметной области;

L – множество формальных языков;

C – множество контекстов (прагматик, целей, ограничений);

U = A x P - представляет декартово произведение;

Ů ={U1,U2,U3, Un} обучающий набор , где **Ui Є Ξ(U)-** представляет множество всех подмножеств U;

X={X1,X2,X3,..,Xm} – конечное множество состояний;

Y={Y1,Y2,Y3,..,Yk} - множество выходных величин;

определим **ρ: Ξ(U)** →Y, отображение классификации, которое известно только преподавателю;

допустим, дано **F={fi, i=1,l & fi: Ξ(U) →Y }** множество описаний, которые будут построены в процессе обучения;

определим также выбор набора обучения

φ: U x L x C → Ξ(U) , φ Є Φ;

и операторы трансформации представлений

ψ: U x L → U x L, ψ Є Ψ;

Кроме того, скажем, дан критерий обучения **Λ(ρ,f, φ, ψ)**, который зависит от близости классификации **f** , полученной в результате обучения, к истинной классификации преподавателя - **ρ**.

Соответственно, модель обучения можем представить следующим образом:

M= < Ů , X , Y , ρ, F, Φ, Ψ, Λ >

В общем задача обучения состоит в поиске таких значений **f, φ, ψ ,** при которых будет найден экстремум целевой функции

$$\Lambda(\rho^*, f^*, \varphi^*, \psi^*) = \underset{f,\varphi,\psi}{\text{extr}} \{ \Lambda(\rho, f, \varphi, \psi) \} \quad (1)$$

Разными вариациями формулы (1) можно получить задачи обучения с разными критериумами:

- задача оптимального подбора тем и вопросов
- задача выбора наилучшего представления
- задача составления наилучшей последовательности, выбранной согласно набору обучения

В общем, задача (1) представляет комбинаторную задачу со множеством критериумов экспоненциальной сложности.

2.2. Представление знаний и трансформация представлений.

Для практического представления знаний о предметной области необходимо выбрать базисное представление, которое обладает большими выразительными возможностями и работа с которым будет легка с точки зрения сбора знаний.

В качестве языка представлений базисных знаний может быть выбран предусмотренный для анализа и проектирования объектно-ориентированный универсальный язык моделирования UML и его расширение – язык ограничений OCL, который может быть использован как для декларативного – в виде объектов, так и процедурного представления знаний. UML обладает следующими преимуществами:

- представленные с помощью UML знания прямо доступны как для восприятия человеком, так и для и для машинных процессов;
- знания в модели UML могут легко изменяться исходя из объектно – ориентированного модулирования;
- модели UML могут быть использованы для представления абстрактных знаний;
- новое знание может быть выведено с помощью механизмов вывода OCL – расширения UML

Представление в виде семантических сетей осуществлено с использованием языка XML.

Соответственно, разработаны механизмы объектно – ориентированного представления знаний с помощью диаграмм UML и методы и алгоритмы перевода их в семантические сети.

Существует несколько методов трансформации UML–моделей. Наиболее простым является описание преобразования существующих процессов с использованием языка программирования. Но у него есть такое отрицательное свойство, что для добавления новой трансформации необходимо написание новой программы. Следующий метод имеет в виду, что UML–модели могут быть представлены в виде графов и может быть использован математический аппарат трансформации графов. Но в этом случае теряется семантика, заложенная в первичное представление и от пользователя требуется хорошее знание языка представления и теории графов. Еще один метод состоит в использовании XML для преобразования XML-документов в XMI (XML Metadata Interchange) стандарт, который обладает возможностью представления UML –модели в виде XML документа, но выполнение такой трансформации довольно проблематично.

Решение проблемы предлагается с использованием комбинированного подхода [23].

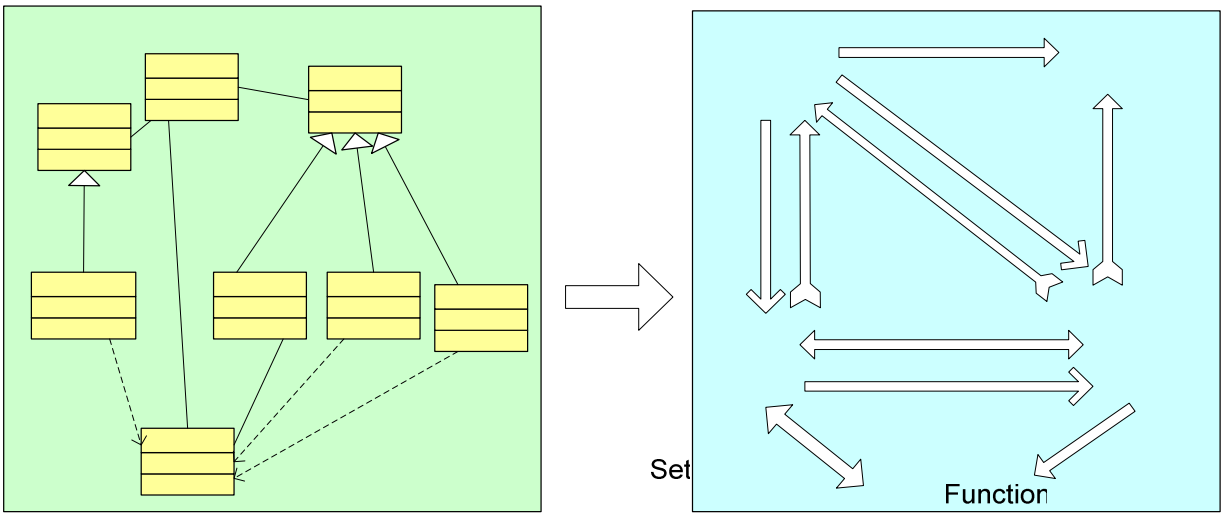
Введем рекурсивное понятие модуля знаний (он может содержать в себе другой модуль и к тому же модуль может иметь разные представления) . Определим следующие операции на модуле

1. создание нового модуля
2. уничтожение модуля
3. преобразование модуля в другой модуль
4. объединение двух модулей в новый модуль
5. разбиение модуля на два новых модуля
6. присоединение к одному модулю другого
7. выделение из одного модуля внутреннего модуля

Пусть будут определены стандартные преобразования, например операции на графах

- поиск пути в графе
- выделение максимально устойчивого подграфа

- поиск цикла Эйлера
- и другие



UML представление отображения множеств

Графовое представление отображения множеств с изучением их анализа

Рис. 2. Пример трансформации двух представлений

Желательно получение графовых моделей. Например, логическую структуру знаний (классы, объекты и отношения) представленную графической нотацией UML можно представить с помощью «и-или» графов: класс (объект) обладает или не обладает некоторым свойством или отношением. В математике хорошо известно, что «и-или» граф можно описать с помощью логической функции.

С помощью UML диаграмм активности, взаимодействия, состояния и последовательности описаны процедурные знания, которые представлены в виде программных процедур и функций. Вопросы тестирования программных модулей уже давно вырабатываются и используются в процессе тестирования практических систем. Соответственно, графовые представления в программировании могут быть обобщены и распространены на любые процессы представления знаний и для обучения. Исходя из вышесказанного, видно что программные инструментальные модели в автоматизированном режиме дадут возможность пользователю осуществить сбор знаний, выбор режима обучения и трансформации знаний, которые в свою очередь требуют разработки технологии автоматизированной трансформации моделей.

2.3. Планирование процесса разработки программного обеспечения

Для того чтобы команда проекта смогла успешно осуществить проект, она должна выполнить сотни задач, многие из которых взаимосвязаны. Эффективное управление этим процессом очень важно для общего успеха. Набор действий, выполняемых менеджерами проекта, определен в *процессе управления проектом*. Обычно процесс управления проектом состоит из трех основных стадий:

• Планирование проекта

• Выполнение проекта

• Закрытие проекта

На стадии *планирования* проекта менеджер проекта изучает контрактные обязательства и разрабатывает план по их выполнению. Создание плана проекта включает определение процесса жизненного цикла, который нужно соблюдать, оценивание трудоемкости и сроков работ, подготовку детального графика задач и т.д. В этом плане также отражаются управление качеством и конфигурацией и управление рисками. На данной фазе менеджер проекта:

• Выполняет подготовительные и административные задачи

• Создает план и график проекта

- Определяет цели проекта
- Устанавливает подходящий стандартный процесс выполнения проекта
- Адаптирует стандартный процесс к требованиям проекта
- Определяет процесс управления изменениями в требованиях
- Оценивает трудоемкость
- Планирует людские ресурсы и структуру команды
- Определяет ключевые точки проекта и создает график работ
- Определяет цели качества и план их достижения
- Разрабатывает план предупреждения ошибок
- Выявляет риски и создает план по смягчению их проявлений
- Определяет план измерений для проекта

- Определяет план обучения для проекта
- Определяет процедуры отслеживания проекта

• Выполняет экспертизы плана и графика работ по проекту

• Получает одобрение плана и графика работ от вышестоящего руководства

• Создает план управления конфигурацией и проводит его экспертизу

• Ориентирует команду проекта в соответствии с планом управления проектом

Вторая фаза, *выполнение* проекта, включает выполнение плана проекта, отслеживание состояния проекта и внесение коррективов в любой момент, когда характеристики проекта отклоняются от пути, проложенного планом. Иначе говоря, на этой фазе происходит отслеживание и контроль реализации процесса, выбранного для проекта. Это самый длинный этап в процессе управления проектом, в него входят такие периодически выполняемые задачи, как мониторинг состояния и качества проекта и проведение необходимых корректирующих шагов. В этой фазе менеджер проекта предпринимает следующие действия:

• Выполняет проект согласно плану проекта

• Отслеживает состояние проекта

• Вместе с вышестоящим руководством проверяет состояние проекта

• Отслеживает соблюдение процесса, определенного для проекта

• Анализирует ошибки и выполняет действия по их предупреждению

• Отслеживает выполнение на уровне программы

• Проводит экспертизы в контрольных точках этапов и при необходимости изменяет план

Критерием на входе является завершение и одобрение плана проекта, критерием на выходе — поставка всех рабочих продуктов и их приемка заказчиком.

Последняя стадия процесса управления проектом — *закрытие* проекта — включает планомерное завершение проекта после приемки заказчиком. Основная цель этой стадии — изучение полученного опыта для дальнейшего усовершенствования процесса. Основную деятельность составляет анализ данных, полученных в результате выполнения проекта: анализируется система показателей, для будущего использования собирается все, что было

накоплено процессом (например, такие материалы, как шаблоны и инструкции, применявшиеся в помощь управлению самим процессом), фиксируются полученные уроки. Поскольку основная цель — обучение на примере этого проекта, подведение итогов является групповой деятельностью, в которой участвуют менеджер проекта и другие члены команды. Критерием на входе служит приемка заказчиком рабочих продуктов. Критерий на выходе — проведение заключительного совещания, посвященного завершению проекта. Основными результатами этой фазы являются отчет о закрытии проекта и собранные данные о выполнении процесса.

При *планировании* проекта его менеджеру нужно решить, какой процесс должен использоваться для создания ПО. Это важное решение, поскольку оно определяет значительную часть разработки. Существует несколько *моделей процессов для разработки ПО*. К наиболее общим из них относятся *каскадная модель*, *итеративное расширение*, *создание прототипов* и *спиральная модель*. Особенно широко используется каскадная модель, которая организует фазы в линейную последовательность, хотя, как правило, при реализации эту модель адаптируют для сведения ее недостатков к минимуму. На макроуровне стандартный процесс может дать оптимальную организацию фаз для класса проектов, а также создает удобную отправную точку для определения процесса. Однако стандартный процесс не может применяться во всех ситуациях; наилучшим выбором часто оказывается скорректированный стандартный процесс. Следовательно, определяя используемый процесс, менеджер проекта выбирает базовый процесс и решает, как его адаптировать, чтобы получить процесс, подходящий для данного проекта.

Стандартный процесс разработки напоминает каскадную модель, хотя ее традиционные фазы разбиты на более мелкие фазы, или стадии, что позволяет выполнять некоторые из них параллельно. Например, планирование системного тестирования определено как отдельная от собственно системного тестирования фаза, и такая практика дает командам возможность планировать системное тестирование одновременно с программированием, хотя испытания системы проводятся только после завершения программирования. Фазы этого процесса включают анализ требований,

высокоуровневое проектирование, детальное проектирование, создание, тестирование элементов, планирование сборки, сборку, планирование системного тестирования, системное тестирование, документирование, приемку и установку, гарантийное обслуживание. На рис. 3 изображены фазы процесса разработки и зависимости между ними.

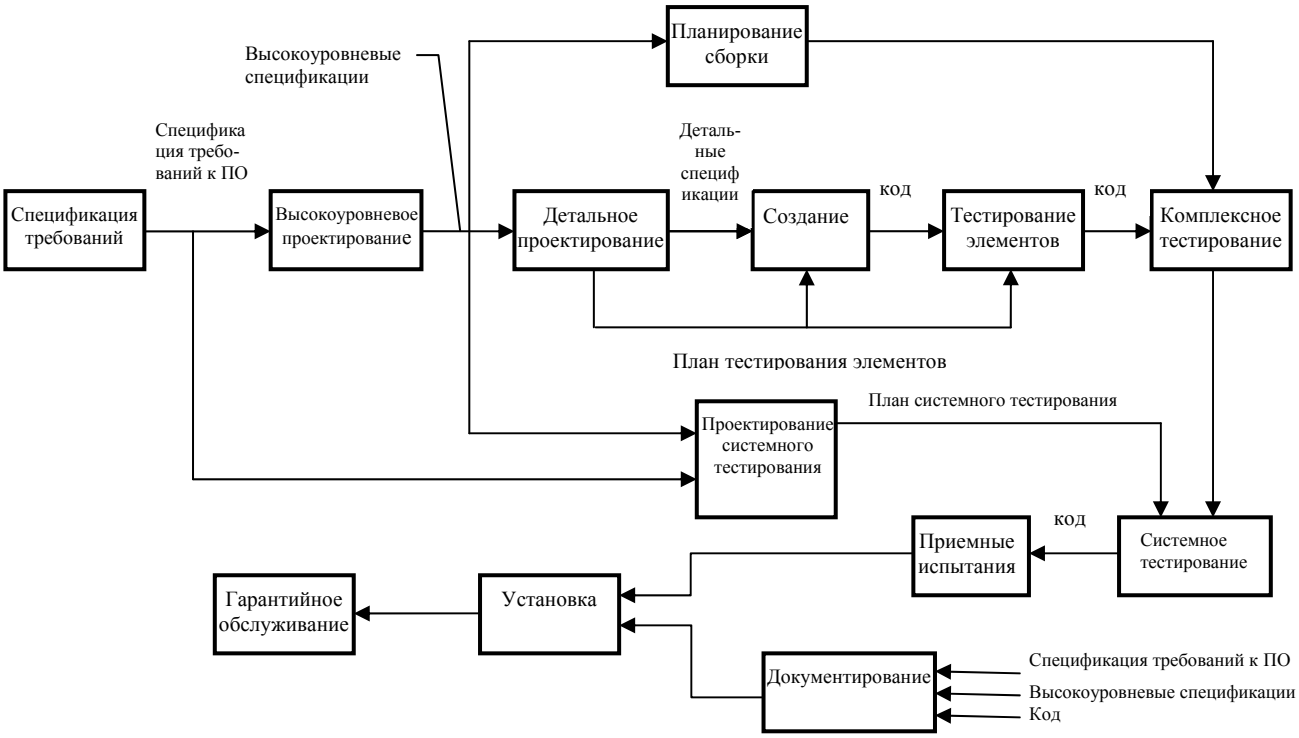


Рис. 3. Процесс разработки программного обеспечения.

Формальное описание этого процесса определяет для каждой фазы (стадии) критерии на входе и выходе, входные и выходные данные, участников, действия и другую информацию. Описания фаз, как правило, кратки; они содержат перечни действий, предпринимаемых в данной фазе. Весь этот процесс остается неизменным даже для проекта, использующего объектно-ориентированный подход, хотя в этом случае некоторые фазы выполняются иначе. Различия заключены главным образом в фазах анализа и проектирования, но для некоторых последних фаз инструкции и стандарты также изменяются. Базовый процесс также используется в проектах с итеративной разработкой, с созданием прототипов и в проектах, где выполняются лишь некоторые стадии жизненного цикла. В подобных

случаях стандартный процесс подстраивается под проект. Эта подстройка выполняется через адаптацию процесса.

2.4. Вероятностная оценка риска проекта

Уже указывалось, что под риском проекта (project risk) понимается степень опасности для успешного его осуществления. Риск, связанный с проектом, характеризуется тремя факторами: событие, связанное с риском; вероятность риска; сумма, подвергаемая риску. Чтобы количественно оценить риск, необходимо знать все возможные последствия принимаемого решения и вероятность последствий этого решения. Выделяют два метода определения вероятности.

Объективный метод определения вероятности основан на вычислении частоты, с которой происходят некоторые события. Частота при этом рассчитывается на основе фактических данных. Так, например, частота возникновения некоторого уровня потерь в процессе реализации проекта может быть рассчитана по формуле:

$$f(A)=n(A)/n;$$

где

f- частота возникновения некоторого уровня потерь;

n(A) - число случаев наступления этого уровня потерь;

n - общее число случаев в статистической выборке, включающее как успешно осуществленные, так и неудавшиеся проекты.

Субъективная вероятность является предположением относительно определенного результата, основывающемся на суждении или личном опыте оценивающего, а не на частоте, с которой подобный результат был получен в аналогичных условиях. Различная информация или различные возможности оперирования с одной и той же информацией объясняют широкое варьирование субъективных вероятностей. Вероятность, равная нулю, означает невозможность наступления конкретного события; вероятность, равная единице, - непременное наступление события. Сумма вероятностей всех возможных вариантов равна единице. Важными понятиями,

применяющимися в вероятностном анализе риска являются понятия альтернативы, состояния среды, исхода.

Альтернатива - это последовательность действий, направленных на решение некоторой проблемы. Примеры альтернатив: приобретать или не приобретать новое оборудование; следует ли внедрять в производство новый продукт и т.д.

Состояние среды - ситуация, на которую лицо, принимающее решение, не может оказывать влияние (например, благоприятный или неблагоприятный рынок, климатические условия и т.д.).

Исходы (возможные события) возникают в случае, когда альтернатива реализуется в определенном состоянии среды. Это некая количественная оценка, показывающая последствия определенной альтернативы при определенном состоянии среды (например, величина прибыли и т.п.).

P-пространство вероятностей;

W- пространство состояний среды;

X-просранство исходов (доход от инвестиционного проекта);

(W, P, X) - случайная величина.

Анализируя и сравнивая варианты проектов, лица, принимающие решение, действуют в рамках теории принятия решений. Как уже было отмечено выше, понятия неопределенности и риска различаются между собой. Вероятностный инструментарий позволяет более четко разграничить их. В соответствии с этим, в теории принятия решений выделяются три типа моделей:

1. *Принятие решений в условиях определенности* - лицо, принимающее решение (ЛПР) точно знает последствия и исходы любой альтернативы или выбора решения. Эта модель нереалистична в случае принятия решения о долгосрочном вложении капитала.

2. *Принятие решений в условиях риска* - ЛПР знает вероятности наступления исходов или последствий для каждого решения.

3. *Принятие решения в условиях неопределенности* - ЛПР не знает вероятностей наступления исходов для каждого решения.

Если имеет место неопределенность (т.е. существует возможность отклонения будущего дохода от его ожидаемой величины, но невозможно даже приблизительно указать вероятности наступления каждого возможного

результата), то выбор альтернативы может быть произведен на основе одного из трех критериев:

1. *Критерий MAXIMAX* (критерий оптимизма) - определяет альтернативу, которая максимизирует максимальный результат для каждой альтернативы.
2. *Критерий MAXIMIN* (критерий пессимизма) - определяет альтернативу, которая максимизирует минимальный результат для каждой альтернативы.
3. *Критерий БЕЗРАЗЛИЧИЯ* - выявляет альтернативу с максимальным средним результатом (при этом действует негласное предположение, что каждое из возможных состояний среды может наступить с равной вероятностью; в результате выбирается альтернатива, дающая максимальную величину математического ожидания).

Соответственно, по своему отношению к неопределенности люди, в том числе персональные инвесторы, подразделяются на пессимистов, оптимистов и нейтральных к неопределенности, принимают решение о выборе проекта в соответствии со следующими условиями:

- временными предпочтениями
- ожидаемой доходностью проекта
- степенью неприятия риска
- вероятностными оценками

По мере осуществления проекта поступает дополнительная информация об условиях реализации проекта и, таким образом, ранее существовавшая неопределенность “снимается”. При этом информация, касающаяся проекта, может быть как выражена, так и не выражена в вероятностных законах распределения. Поэтому в контексте анализа проектов следует рассматривать ситуацию принятия решения в условиях риска. Итак, в этом случае:

- известны (предполагаются) исходы или последствия каждого решения о выборе варианта;
- известны вероятности наступления определенных состояний среды.

На основе вероятностей рассчитываются стандартные характеристики риска:

1. *Математическое ожидание* (среднее ожидаемое значение) - средневзвешенное всех возможных результатов, где в качестве весов используются вероятности их достижения.

где xj - результат (событие или исход, например величина дохода); pj - вероятность получения результата xj.

2. *Дисперсия* - средневзвешенное квадратов отклонений случайной величины от ее математического ожидания (т.е. отклонений действительных результатов от ожидаемых) - мера разброса.

$$s^2 = D = e(xi- E)*p(xi)$$

Квадратный корень из дисперсии называется стандартным отклонением:

Обе характеристики являются абсолютной мерой риска.

3. *Коэффициент вариации* - служит относительной мерой риска:

$$c=s/E$$

4. *Коэффициент корреляции* - показывает связь между переменными, состоящую в изменении средней величины одного из них в зависимости от изменения другого.

R(x1,x2)=Cov(x1,x2)/sx1sx2, RO[-1;+1],

где Cov(x1,x2)=E[(x1-Ex1)(x2-Ex2)]

2.5. Оценивание трудоемкости и составление графика работ

Неточное оценивание — бич менеджеров проектов во многих технических дисциплинах, и разработка ПО не является исключением, как однозначно свидетельствует печальная слава программных проектов. Кажется, будто рассчитанные стоимость и сроки никогда не бывают достаточными для выполнения проекта. В сфере обслуживания неточное оценивание наносит еще больший вред. Почти всегда причиной неожиданностей, возникающих в этих двух областях к концу проекта, является неточная оценка трудоемкости или неточный график работ. Для решения задачи оценивания не существует быстрых и готовых решений. Менеджеры проектов, однако, могут повысить точность своих оценок, используя проверенные, базирующиеся на опыте и полученных ранее данных правила.

Рассмотрим некоторые понятия, связанные с оцениванием и составлением графика. Оценивание трудоемкости обычно предпринимается на начальных стадиях проекта, когда осмысливается ПО, которое нужно создать. Эту работу можно переделать позднее, когда появится больше информации. Оценки высокой степени точности, по существу, и не требуются. Менеджер проекта должен получить разумные оценки, чтобы задача была решена, а команда не исчерпала бы свои силы. Диапазон разумности, зависящий от человеческих факторов, не слишком обширен, но, вероятно, достаточно велик, чтобы дать достаточную свободу для оценивания. Оценивание трудоемкости может основываться на интуиции или на имеющемся опыте, но при более научном — и более желательном — подходе используется модель оценивания.

Модель оценивания трудоемкости

Модель оценивания программного обеспечения определяет характеристики проекта, значения которых (или их оценки) ей нужны, и способы использования этих значений для вычисления трудоемкости. Модель оценивания не работает — и не может работать — в вакууме: ей нужны входные данные, чтобы на выходе она произвела оценку трудоемкости. В начале проекта, когда детали ПО неизвестны, мы надеемся, что модели оценивания потребуются значения характеристик, которые можно измерить на этой стадии. Когда определяют трудоемкость, которая потребуется для создания ПО, его размер является преобладающим фактором. Но поскольку проект существует только в замысле, окончательный размер программ неизвестен. Таким образом, если в модели оценивания нужно использовать размер, его тоже нужно оценить.

Широко распространен подход, в котором для получения оценки суммарной трудоемкости по оценке размера используется формула регрессионного анализа накопленных данных по трудоемкости и размеру. Затем, как только становится известна суммарная трудоемкость, ее

значения для разных фаз и действий можно вычислить как определенную долю от общей трудоемкости. Было предложено много моделей, использующих для оценивания *нисходящий подход*. Также были построены модели, использующие для оценки размера метод функциональных точек (вместо LOC - Lines of code). В эти модели можно поместить другие влияющие на трудоемкость факторы, чтобы уточнить базирующиеся на них оценки. В другом подходе размер системы уточняется на основании этих параметров таким же образом, как на основании числа функциональных точек.

С другой стороны, при восходящем подходе сначала получаются оценки для частей проекта, а потом для проекта в целом. Иначе говоря, общая оценка проекта составляется из оценок его частей [19]. Один восходящий метод предусматривает использование некоторого типа оценок, основанных на действиях. В этой стратегии сперва перечисляются основные действия, затем оценивается трудоемкость для каждого действия. Из этих оценок получается трудоемкость для всего проекта.

Восходящий подход применяется для прямой оценки трудоемкости; как только проект разделен на мелкие задачи, появляется возможность непосредственно оценить трудоемкость, необходимую для их выполнения. Хотя размер, конечно, играет важную роль при определении трудоемкости многих действий в проекте, основное преимущество этого подхода состоит в том, что он не требует явной оценки размера программ. Вместо этого нужен перечень задач проекта, который в некоторых ситуациях подготовить проще. Риск восходящих методов в том, что в списке задач могут быть пропущены некоторые важные действия. Когда трудоемкость оценивается непосредственно для задач, бывает сложно рассчитать трудоемкость для непроизводительных задач, не определенных так ясно, как программирование или тестирование, например: для управления проектом, которое охватывает весь проект. Обоим подходам, и

нисходящему, и восходящему, нужна информация о проекте: размер (для нисходящих подходов) и перечень задач (для восходящих). Во многих отношениях эти подходы дополняют друг друга. Точность оценок обоих типов повышается, если появляется дополнительная информация о проекте или если проект уже выполняется. Например, оценить размер крайне трудно, когда даны требования очень высокого уровня, проще, когда закончено проектирование, и еще проще, когда начата разработка кода. Таким образом, точность оценок зависит от точки, в которой оценивается трудоемкость, причем точность возрастает по мере появления новой информации о проекте.

Оценивание графика выполнения работ

Как только трудоемкость известна или зафиксирована, появляется возможность составить несколько различных графиков выполнения работ в зависимости от выделенных проекту ресурсов (персонала). Например, для проекта, трудоемкость по которому оценивается в 56 человеко-месяцев, при наличии семи сотрудников можно составить общий график на восемь месяцев. Также возможен график на семь месяцев при восьми сотрудниках и график приблизительно на девять месяцев при шести сотрудниках. Однако хорошо известно, что в программном проекте нельзя считать людские ресурсы и месяцы полностью взаимозаменяемыми. Например, если проект в приведенном примере будет выполняться 56 сотрудниками, нельзя составить график на один месяц, пусть даже трудоемкость точно соответствуют требованиям. Точно так же никто не стал бы выполнять этот проект за 28 месяцев с двумя сотрудниками. Иными словами, как только трудоемкость зафиксирована, мы получаем некоторую гибкость в регулировании графика, подбирая численность штата для проекта. Но эта гибкость не безгранична, и этот факт подкреплен данными, которые показывают, что никакую простую формулу, связывающую трудоемкость и график, нельзя приспособить к эмпирическим данным "Растянуть" график

просто: достаточно уменьшить численность штата (хотя проект, выполняемый слишком долго, может потерять ценность). Однако сжать график не столь легко. Вообще говоря, если сжатие графика превышает то, что можно считать "нормальным", трудоемкость увеличивается; когда же численность сотрудников превышает оптимальное значение, возникают ненужные траты времени, растет количество доработок и т.д.

В некоторых подходах обсуждается влияние сжатия графика на общую трудоемкость. Однако чтобы оценить это влияние, сначала необходимо определить нормальный график для проекта. Один из методов определения нормального (или номинального) графика работ состоит в использовании подходящей функции, связывающей его с трудоемкостью. В свою очередь метод определения этой функции заключается в исследовании образцов в статистических данных из завершенных проектов. Например, мы можем получить схему рассеяния трудоемкости и графика работ для завершенных проектов, а затем приспособить к ней кривую регрессии. Эта кривая обычно является нелинейной, поскольку сроки выполнения работ не растут линейно с повышением трудоемкости. Затем по уравнению кривой можно определить график для проекта, чья трудоемкость была оценена.

Оценивание трудоемкости.

Оценивание обычно выполняется после анализа. То есть когда менеджер проекта оценивает трудоемкость, требования ему уже достаточно понятны. Для поддержки этого подхода организуются бизнес-процессы. Например, фаза требований иногда выполняется как отдельный проект, обособленный от проекта разработки ПО. Менеджер проекта может выбрать любой из подходов оценивания, соответствующий природе работы. Предлагается выполнять оценивание, используя несколько методов, чтобы подтвердить результат, полученный первым методом, или

чтобы сократить риски, особенно в том случае, когда количество информации из похожих завершенных проектов ограничено.

Восходящий подход к оцениванию. Поскольку типы проектов, существенно различаются, считается предпочтительным и рекомендуется восходящий подход. Применяется метод разбиения задач на подзадачи, хотя некоторые ограничения, свойственные этой стратегии, приходится преодолевать использованием накопленных данных и базовой линии устойчивости процесса. При методе разбиения задач на подзадачи менеджер проекта сначала делит разрабатываемое ПО на основные программы (или единицы). Каждая программная единица затем классифицируется на основании определенных критериев как простая, средней сложности или сложная. Для каждой единицы классификации менеджер проекта определяет стандартные программирования и самотестирования (которые вместе Называются трудоемкостью создания). Эти стандартные трудоемкости создания могут базироваться на данных из похожего завершенного проекта, на имеющихся внутренних инструкций и на некоторых их сочетаниях. Как только становится известно число единиц в трех категориях сложности и выбрана оценка трудоемкости создания каждой программы, определяется общая трудоемкость фазы создания проекта. Трудоемкости, необходимые для других фаз и действий, определяются по трудоемкости создания в процентном отношении к трудоемкости программирования. По базовой линии устойчивости процесса или базе данных процесса рассчитывается распределение трудоемкости в проекте. Менеджер проекта пользуется этим распределением, чтобы высчитать трудоемкость для других фаз и действий. Из этих оценок получается суммарная трудоемкость для проекта.

Этот подход разумно сочетает опыт и данные. Если подходящие данные недоступны (например, когда запускается проект нового типа), трудоемкость создания можно оценить на основе опыта, проанализировав

проект и узнав все виды программных единиц. Получив такую оценку, можно рассчитать и другие действия, обработав данные о распределении трудоемкости из завершенных проектов. Такая стратегия принимает во внимание даже те требующие труда действия, которые иногда сложно перечислить заранее: в распределении трудоемкости для проекта часто используют категорию "другие", чтобы учесть разнородные задачи. Процедуру оценивания можно объединить в следующую последовательность действий:

1. Определение программ системы и их классификация как простых, средних или сложных (ПР/СР/СЛ). Желательно, насколько возможно, пользоваться стандартными определениями или определениями из прошлых проектов.

2. Если для проекта существует специфичная базовая линия, из нее получают средние трудоемкости создания программ ПР/СР/СЛ.

3. Если для проекта не существует специфичной базовой линии, в базе данных процесса разыскиваются проекты, похожие типом проекта, технологией, языком и другими атрибутами. Данные из этих проектов используются для определения трудоемкости создания программ ПР/СР/СЛ.

4. Если в базе данных процесса аналогичный проект отсутствует, для программ ПР/СР/СЛ применяют среднюю трудоемкость создания из базовой линии устойчивости процессов.

5. Применение специфичных для процесса факторов в целях уточнения трудоемкости создания программ ПР/СР/СЛ.

6. Расчет общей трудоемкости создания на основе трудоемкости создания программ ПР/СР/СЛ и счетчиков этих программ.

7. Оценка трудоемкости для других задач и общей трудоемкости с использованием распределения трудоемкости, данного в базовой линии

устойчивости или найденного для аналогичных проектов в базе данных процесса.

8. Уточнение оценок на основании специфичных для проекта факторов.

В этой процедуре используются база данных процесса и базовая линия устойчивости процесса. Как уже говорилось, если выполняется много проектов одного типа, можно построить базовую линию, специфичную для проекта. Такие базовые линии аналогичны общим базовым линиям, но используют только данные из определенных проектов. Эти базовые линии позволяют наилучшим образом предсказать трудоемкость нового проекта того же типа. Следовательно, их использование для оценивания предпочтительнее других способов. Поскольку на трудоемкость, необходимую для выполнения проекта, могут влиять многие факторы, весьма важно, чтобы оценки учитывали факторы, специфичные для проекта. Вместо того чтобы классифицировать параметры по разным уровням и затем определять влияние на требование к трудоемкости, выделенный здесь подход позволяет менеджеру проекта определить воздействие на оценку специфичных для проекта факторов. Менеджеры проектов могут сделать эту поправку, исходя из собственного опыта, опыта членов команды или данных из проектов, обнаруженных в базе данных процесса. Заметим, что метод классификации программ по нескольким категориям и использование средних значений трудоемкости создания для каждой категории соблюдается для всего оценивания. Однако при составлении детального графика, в котором менеджер проекта назначает каждую программную единицу члену команды для программирования и выделяет резерв времени для действий процесса, в расчет принимаются характеристики каждой единицы, и она получает больше или меньше времени по сравнению со средним значением.

Нисходящий подход к оценивани. Нисходящий подход, начинается с оценки размера ПО, выражаемого в числе функциональных точек. Число функциональных точек можно подсчитать, используя стандартные правила подсчета. Если же известна оценка размера в LOC, ее можно преобразовать в число функциональных точек. В дополнение к оценке размера ПО нисходящий подход требует оценки продуктивности. В базовом подходе

необходимо начать с уровней продуктивности аналогичных проектов (данные по которым собраны в базе данных процесса) или со стандартных значений продуктивности (эти данные имеются в базовой линии устойчивости процесса), а затем, если потребуется, подстроить эти уровни, чтобы они соответствовали проекту. Далее оценка продуктивности и пользуется для подсчета общей оценки трудоемкости. Из общей оценки трудоемкости через процентные распределения получаются оценки различных фаз проекта. (Эти распределения, как и в восходящем подходе, берутся из базы данных процесса или из базовой линии устойчивости.)

В целом нисходящий поток можно представить следующей последовательностью действий:

1. Получение оценки общего размера ПО в функциональных точках.

2. Фиксирование уровня продуктивности для проекта, используя данные о продуктивности из специфичной для проекта базовой линии устойчивости, общей базовой линии устойчивости процесса или из аналогичных проектов.

3. Получение оценки общей трудоемкости из оценок продуктивности и размера.

4. Использование данных распределения трудоемкости из базовой линии устойчивости процесса или из аналогичных проектов для оценки трудоемкости различных фаз.

5. Уточнение оценок с учетом специфичных для проекта факторов.

Как и восходящее оценивание, нисходящий подход позволяет уточнять оценки с помощью специфичных для проекта факторов. Это допущение, не определяя реально сами факторы, подтверждает, что каждый проект уникален и может обладать такими характеристиками, которых другие проекты лишены. Эти характеристики невозможно перечислить, и тем более невозможно формально смоделировать их влияние на продуктивность. Следовательно, решение о рассматриваемых факторах и их влиянии на проект остается за менеджером проекта.

Подход с числом вариантов использования. Подход с числом вариантов использования аналогичен методам, основанным на числе функциональных точек. Этот подход можно применить, если спецификация

требований определяется с помощью вариантов использования. В этот подход входяг следующие шаги:

1. Классификация каждого варианта использования как простого, средней сложности или сложного. Основой для классификации является число транзакций в варианте использования, включая вторичные сценарии. Транзакция определяется как неделимый набор действий, который или выполняется весь полностью, или не выполняется вообще. Простой вариант использования имеет не более трех транзакций, средний - от четырех до семи транзакций, а сложный — более семи транзакций. Простому варианту использования присваивается коэффициент 5, среднему — 10, сложному — 15. В таблице приведены классификация и коэффициенты.

2. Расчет суммарного числа точек нескорректированных вариантов использования (unadjusted use case points, UUCP) как взвешенной суммы коэффициентов вариантов использования в приложении. Иными словами, для каждого из трех классов сложности сначала вычисляется произведение числа вариантов использования определенной сложности и коэффициента для этой сложности. Сумма трех произведений будет числом UUCP для приложения.

3. Коррекция первоначального числа UUCP, чтобы отразить сложность проекта и опыт команды. Для этого сначала вычисляется коэффициент технической сложности (technical complexity factor, TCF) на основе факторов, заданных в таблице , каждому из которых определяется рейтинг от 0 до 5. Рейтинг 0 означает, что данный фактор для этого проекта несуществен; рейтинг 5 означает основной фактор. Для каждого фактора его рейтинг умножается на вес из таблицы, эти числа складываются, и вы получаете число TFactor. TCF высчитывается по следующей формуле:

TCF = 0,6 + (0,01 *TFactor)

Тип варианта использования	Описание	Коэффициент
Простой	Не больше 3 транзакций	5
Средний	4–7 транзакций	10
Сложный	Более 7 транзакций	15

Таблица 1. Сложность и коэффициенты вариантов использования

Порядковый номер	Фактор	Вес
1	Распределенная система	2
2	Требование быстрого ответа и высокой пропускной способности	1
3	Эффективность оперативной работы конечного пользователя	1
4	Сложная внутренняя обработка	1
5	Требование повторного использования кода	1
6	Простота установки	0,5
7	Простота использования	0,5
8	Переносимость	2
9	Легкость изменений	1
10	Параллельное выполнение	1
11	Включение специальных средств безопасности	1
12	Прямой доступ к программам независимых производителей	1
13	Специальные средства обучения пользователя	1

Таблица 2. Технические факторы и их вес

4. Аналогично вычисляется фактор окружения (environment factor, EF) с помощью таблицы 4.3. Для факторов, связанных с уровнем опыта, рейтинг означает отсутствие опыта по предмету, 5 — эксперт, 3 — средний опыт. Для мотивации 0 означает отсутствие мотивации в проекте, 5 — высокую мотивацию, 3 — среднюю. Для стабильности требований 0 означает нестабильные требования, 5 — неизменные требования, 3 — среднюю стабильность. Для сотрудников, занятых неполный день, 0 означает отсутствие таких сотрудников, 5 означает, что весь технический персонал занят неполный день, 3 — что половина сотрудников занята неполный день. Для степени сложности языка программирования 0 означает легкий в использовании язык, 5 — очень сложный язык, 3 — средний. Взвешенная сумма дает EFactor, из которого получается EF по следующей формуле:

EF= 1,4 + (-0,03 *EFactor)

5. Применяя эти два фактора, можно вычислить окончательное число точек вариантов использования (use case points, UCP) по следующей формуле:

UCP = UUCP * TCP * EF

Порядковый номер	Фактор	Вес
1	Знакомство с технологиями Интернета	1,5
2	Опыт в прикладной области	0,5
3	Опыт объектно-ориентированного программирования	1
4	Способности ведущего аналитика	0,5
5	Мотивация	1
6	Стабильные требования	2
7	Персонал с частичной занятостью	-1
8	Сложный язык программирования	-1

Таблица 3. Факторы окружения для команды и их вес

Для оценивания трудоемкости в среднем назначаются по 20 человеко-часов на каждую UCP для всего жизненного цикла. Это дает грубую оценку. Уточнить ее можно следующим образом. Посчитайте, сколько факторов имеют значения меньше 3, а сколько — больше 3. Если общее число факторов, имеющих значения меньше 3, невелико, то 20-человеко-часов на одну UCP будет подходящей оценкой. Если число таких факторов велико, тогда определяется 28 человеко- часов на UCP. Иначе говоря, в зависимости от самых разных факторов менеджер проекта может решить, какое значение из диапазона от 20 до 28 человеко-часов на UCP нужно использовать.

Эффективность подхода в целом

Для анализа эффективности подхода оценивания можно воспользоваться распространенным способом: посмотреть, насколько оценка трудоемкости сравнима с фактической трудоемкостью. Мы уже знаем, что такое сравнение дает только общее представление о точности оценок и не показывает, насколько они оптимальны. Чтобы получить эту информацию, необходимо изучить влияние оценок на команду программистов (например, "напряжена" она или "недоиспользована"). Тем не менее сравнение фактически затраченного труда и оценки трудоемкости дает представление об эффективности метода оценивания. База данных процесса включает информацию об оценке и фактическом значении трудоемкости для завершенных проектов. На рис. 4 показан график рассеяния оценок и фактических значений трудоемкости для некоторых завершенных процессов разработки. Как видно из графика, подход оценивания работает довольно хорошо; большинство точек находится вблизи линии, проведенной под углом 45° (если бы все оценки совпадали с фактическими значениями

трудоемкости, то все точки оказались бы на этой линии). Эти данные также показывают, что более 50% проектов находятся в пределах 25% от оценки трудоемкости. Тем не менее данные свидетельствуют, что оценки обычно ниже фактической трудоемкости: большинство точек находятся не ниже, а выше линии. Таким образом, люди чаще склонны недооценивать ситуацию, и от этого страдает индустрия программных средств.

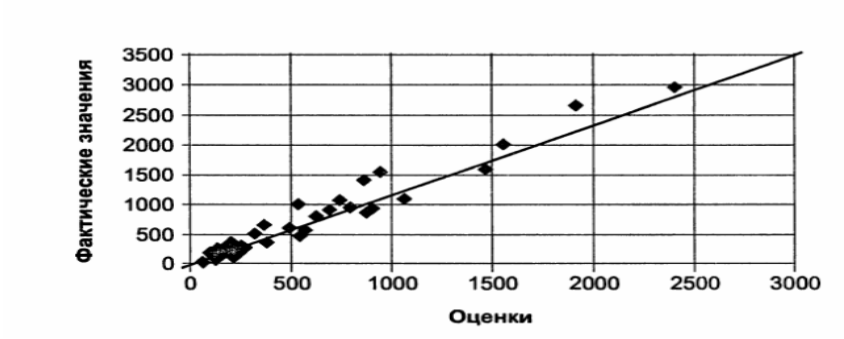


Рис. 4. Сравнение фактических значений трудоемкости с оценками трудоемкости

В среднем фактическая трудоемкость была на 25% выше оценки. Однако хотя возможности для улучшений существуют, подход оценивания доказал свою эффективность.

2.6. Моделирование процессов разработки программного обеспечения в динамике.

Изучение динамики процессов разработки программного обеспечения является одной из важнейших задач. Технология имитационного моделирования процессов (**PSIM**) может использоваться, чтобы оценить проблемы, связанные с усовершенствованием процессов проектирования, руководства и контроля проектом [18]. Тенденции в производстве программного обеспечения к сокращению затрат труда и стоимости проекта усилили потребность в инструментах для лучшего планирования и управления процессами разработки. В результате организации расценивают PSIM как привлекательный инструмент, который может иметь деловую ценность сегодня. Имитационное моделирование процессов (PSIM) может помочь компаниям улучшить свои процессы и достигнуть более высоких уровней зрелости процессов разработки как того требует Модель Зрелости Возможностей (CMM/CMMI®) [SEI 2006]. CMMI был разработан командой,

состоящей из участников от промышленности, правительства и **Институтом Разработки Программного обеспечения** (**SEI**). Эта модель ориентирована на практиков, особенно менеджеров проектов разработки программного обеспечения, и исследователей, изучающих процессы разработки программного обеспечения. CMMI – ведущий стандарт промышленности для измерения процессов разработки программного обеспечения. PSIM помогает организациям улучшать процессы разработки и достичь более высоких уровней зрелости возможностей CMMI.

Имитационное модели использовались в течение многих лет во многих отраслях промышленности такой как автомобильный, химический, производство, и информационная технология, чтобы предсказать работу сложных систем. Имитационное моделирование - компьютеризированные представления систем, разработанные для того, чтобы показать существенные динамические особенности систем, которые они представляют. Эти модели часто сосредотачиваются на том, чтобы воспроизводить поведение системы в течение долгого времени или мультиплицирование работы системы в терминах определенных размерами. Для менеджера моделирование - инструмент, который поддерживает принятие решения, прогноз, и анализ обменов и сценарии "что - если". Имитационное моделирование часто используются в типах ситуаций, где

• поведение в течение долгого времени особенно интересно или важно, и это поведение является трудно прогнозируемым из-за влияния неожиданной обратной связи. В этой ситуации последствия случая или действия вызывают последовательность ответов, петли назад укрепляют или возмещают оригинальный случай или действие. (Проекты разработки программного обеспечения часто характеризуются сложной сетью этих петель обратной связи.)

• работа системы менее эффективна, чем хочется и ее трудно оценить из-за высокой степени неуверенности или возможной стохастичности в системе

• альтернативные конфигурации системы рассматриваются, но вести или осуществить новую конфигурацию, вероятно, сопряжено со значительным риском или стоимостью.

• живое экспериментирование непрактично из-за экономической и/или большой стоимости управления фактическими моделями

Моделирование системы, часто используют для

• улучшения понимания моделируемой системы на основе экспериментирования

• оценки работы системы

• ответов на вопросы “что – если”

• определения вероятного воздействия параметрических и структурных изменений.

PSIM - определенный тип имитационной модели, которая сосредотачивается на том, чтобы копировать поведение и работу организационных процессов.

В комплексе CMMI и PSIM сосредотачиваются на динамике программного обеспечения, а также развития и ведения систем[18].

Потенциальные преимущества PSIM включают уменьшенный риск, уменьшенное усилие и стоимость оценки новых технологий, лучшее (более надежное, быстрое) принятие решения, идентификация улучшенных или оптимальных подходов и процессов. Определенные льготы PSIM при разработке включают

• выбор самого лучшего процесса развития для данной ситуации/обстоятельства

• улучшенное планирование проекта с помощью объективного и количественного основания для принятия решения

• улучшенное проектное решение (контроль и эксплуатационное управление), потому что альтернативные ответы на незапланированные события могут быть быстро оценены, используя PSIM прежде, чем решения будут приняты

• средство отвечать на горячие вопросы, задаваемые менеджерами проектов, такие как

− Каково воздействие на проектную работу увеличения или уменьшения тестирования, осмотров, или обоих? Каков риск?

− Как могли бы изменения процессов развития (таких как процесс требований или критический процесс проекта) воздействовать на работу?

− Какие этапы разработки/шаги являются существенными для успеха?

− Какие фазы/шаги могли быть пропущены или минимизированы, чтобы сократить время цикла и уменьшить затраты, не жертвуя качеством? Каков риск?

− действительно ли осмотры стоит проводить в моем проекте? В каком пункте (ах) в проекте осмотры обеспечивают большую ценность? Что, если мы применим более строгую проверку качества (в терминологии CMMI) только к критическим компонентам продукта?

− Какова ценность применения автоматизированных инструментов для поддержки развития и тестирования? Какую технологию стоит для этого выбрать ?

− Вообще, каковы вероятные льготы (и затраты) осуществления специфического изменения процесса? Связано ли с риском создание специфического изменения?

− Как я объективно сравниваю и располагаю по приоритетам изменения процесса?

− Какие определенные изменения процесса помогли бы мне достигнуть более высоких уровней стандарта CMMI? Они обеспечивают материальную деловую ценность?

• улучшенное понимание многих факторов, которые влияют на проектный успех, для сложного развития программного обеспечения

• увеличение способности выбора процесса и альтернативы через создание PSIM более видимых и конкретных "неосязаемых" процессов

• лучшее обучение и изучение для менеджеров проектов и участников проектной команды

• поднятие руководства проектом к более стратегическому уровню, позволяя проанализировать проекты по их полным циклам жизни.

Рис. 5 иллюстрирует центральную роль, которую PSIM может играть, оценивая альтернативы процесса.

CMM (CMMI) – описательная модель в том смысле, что она описывает существенные (ключевые) атрибуты, определяющие уровень зрелости рассматриваемой организации. Это – нормативная модель в том смысле, что детальное описание методик устанавливает производственный уровень

организации, необходимый для выполнения проектов различной сложности и продолжительности.

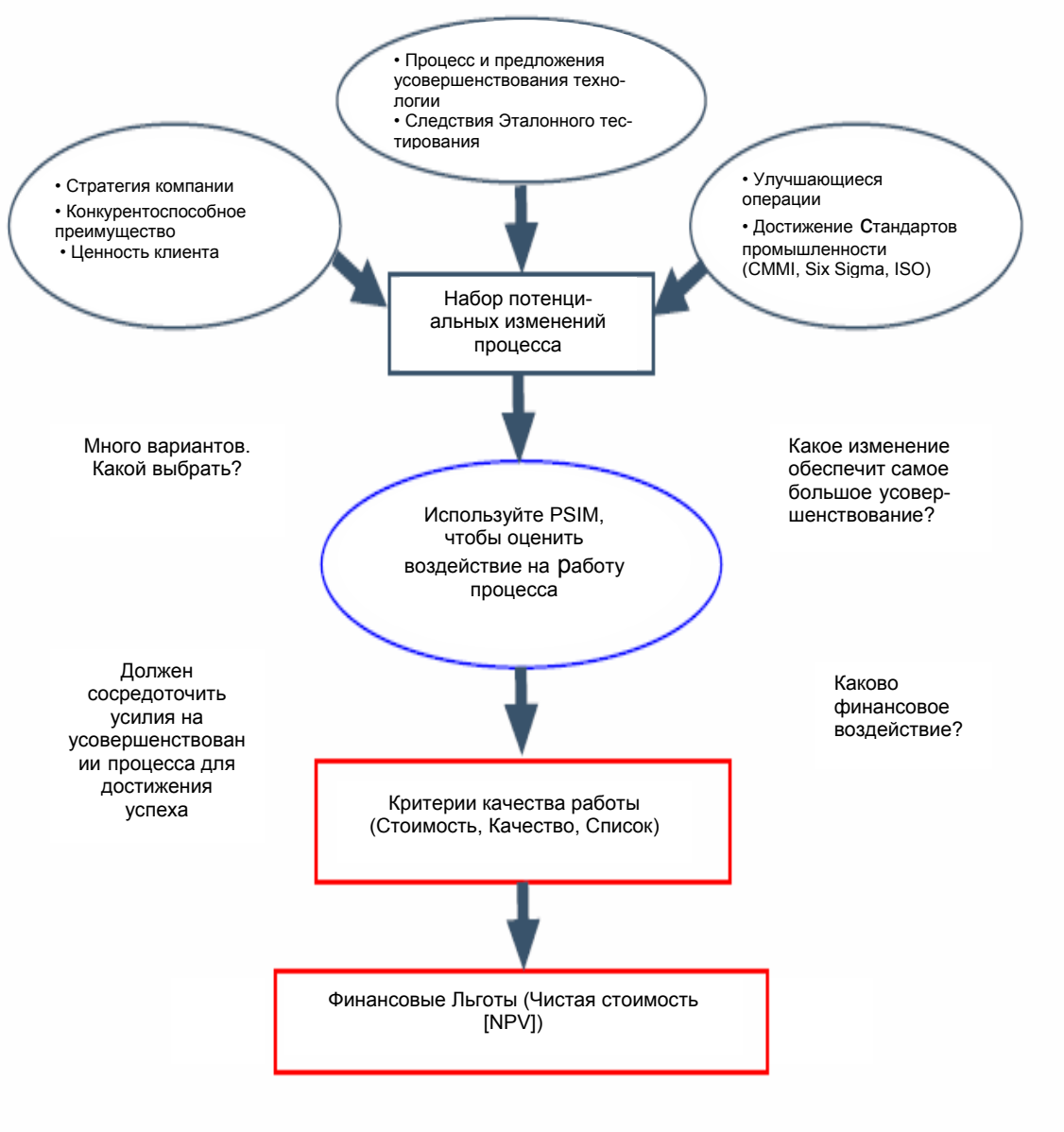


Рис. 5. Центральная роль PSIM в управлении процессом.

В любом случае контекст применения CMM – это рациональная интеграция используемых методик. CMM не является предписанием способа развития организации. CMM описывает производственные характеристики организации для каждого из уровней зрелости, не давая каких-либо инструкций о способах перехода с уровня на уровень. Обычно процесс совершенствования технологии разработок отражается в стратегических планах организации её бизнес-требованиях, её организационной структуре,

используемых методах и технологиях, общей социальной культуре и её системе управления. CMM фокусируется на аспектах процессов, которые можно охарактеризовать как "Суммарные усилия по эффективному управлению".

Последовательное совершенствование процессов организации состоит из последовательности логически связанных простых (относительно) шагов, не являясь чем-то революционным. В CMM эти простые шаги сгруппированы в пять уровней зрелости. Уровни зрелости являются как бы инструментом для измерения степени зрелости процессов и их эффективности. Они также помогают правильно выстраивать приоритеты усилий по совершенствованию всей организации.

Уровни со второго по пятый могут характеризоваться через операции, направленные на стандартизацию и/или модернизацию процессов создания ПО, и через операции, составляющие сами процессы его создания. При этом первый уровень является базой, фундаментом для сравнительного анализа верхних уровней.

PSIM может помочь пользователям увеличивать зрелость процессов возможностей. На каждом уровне CMMI PSIM помогает выполнить или строго поддерживать многие ключевые области процессов. PSIM может использоваться организациями, чтобы позволить, или выполнить CMMI.

2.7. Управление изменениями требований

Требования меняются и изменения в требованиях могут возникнуть в любой момент на протяжении жизни проекта (и даже после его закрытия). Чем позже меняются требования, тем большее влияние это оказывает на проект. Вместо того чтобы желать стабильности требований или надеяться, что по какой-либо причине начальные требования окажутся "такими хорошими", что изменения не понадобятся, лучше приготовиться к их непостоянству, чтобы справиться с заявками на изменения, когда они поступят [19]. Неконтролируемые изменения в требованиях могут неблагоприятно повлиять на стоимость, график и качество проекта. Изменения в требованиях могут составлять до 40% общей стоимости проекта.

Процесс управления изменениями.

В ходе планирования проекта его менеджер решает, какой процесс соблюдать для обработки заявок на изменения. Запланированный процесс обсуждается с заказчиком, поскольку заказчик и изготовитель должны прийти к согласию по вопросам управления изменениями. Обычно процесс определяет, как подавать заявки на изменения, когда требуется официальное утверждение, и т.п. При поступлении заявки на изменение требований должен выполняться процесс управления изменениями требований.

Поскольку заявки на изменения влияют на стоимость, необходимо достичь полного согласия по оплате. Часто с одобрения заказчиков в проектах предусматривается резерв для реализации заявок на изменения (обычно он составляет небольшой процент от общей трудоемкости по проекту). Наличие такого резерва в бюджете упрощает административные аспекты реализации утвержденных заявок.

Используемый в большинстве случаев процесс управления изменениями включает следующие шаги:

1. Регистрация изменений
2. Анализ их влияния на рабочие продукты
3. Оценка трудоемкости, необходимой для выполнения заявок на изменения
4. Повторная оценка графика поставки
5. Анализ влияния на накопленную стоимость
6. Обсуждение этого влияния с вышестоящим руководством, если превышены пороговые значения
7. Получение подтверждения изменения от заказчика
8. Исправление рабочих продуктов

Для отслеживания заявок на изменения ведется специальный журнал. Каждая запись в журнале состоит из номера заявки, краткого описания изменения, состояния заявки на изменение и основных дат. Эффект изменения оценивается путем анализа его влияния. Анализ влияния включает: определение рабочих продуктов, которые потребуется изменить; оценку объема изменений в каждом из них; повторную оценку рисков проекта через пересмотр плана управления рисками; оценивание общих воздействий изменений на оценки трудоемкости и сроков. Результат анализа

рассматривается и утверждается заказчиком. Заявки на изменения включаются в документ спецификации требований — обычно в виде приложений. Иногда затронутые части документа также корректируются, чтобы отразить изменения. Наблюдение за утвержденными заявками на изменения и обеспечение их надлежащей реализации происходит в процессе управления конфигурацией.

Изменение можно классифицировать как незначительное, если общая трудоемкость его реализации не превышает предопределенного значения — например, двух человеко-дней. Незначительные изменения обычно входят в трудоемкость проекта, используя запланированный резерв. Значительные изменения, как правило, оказывают более существенное влияние на трудоемкость и график и должны быть официально утверждены заказчиком. Вышестоящее руководство может наблюдать за изменениями посредством отчетов о состоянии и отчетов контрольных точек.

2.8. Адаптация процесса разработки программного обеспечения

Ни один определенный процесс — будь то стандартный процесс организации или процесс, использованный в предыдущем проекте, - невозможно применить ко всем ситуациям и всем проектам. Определенный процесс нужно адаптировать, чтобы он соответствовал потребностям реального проекта.

Адаптация — это приспособление ранее определенного процесса организации, которое позволяет получить процесс, соответствующий конкретной предметной области или техническим потребностям проекта. Адаптацию можно рассматривать как такое добавление, удаление или изменение действий процесса, которое создает результирующий процесс, лучше приспособленный для достижения целей проекта, чем изначальный. Бесконтрольная адаптация фактически означает создание процесса с чистого листа. Для реального повторного применения определенных ранее процессов используются правила адаптации. В них задаются условия и типы изменений, которые следует вносить в стандартный процесс [19]. По существу в них

определен набор разрешенных отклонений от стандартного процесса, который позволит получить оптимальный для проекта процесс. На рис. 6. наглядно представлена роль инструкций по адаптации.

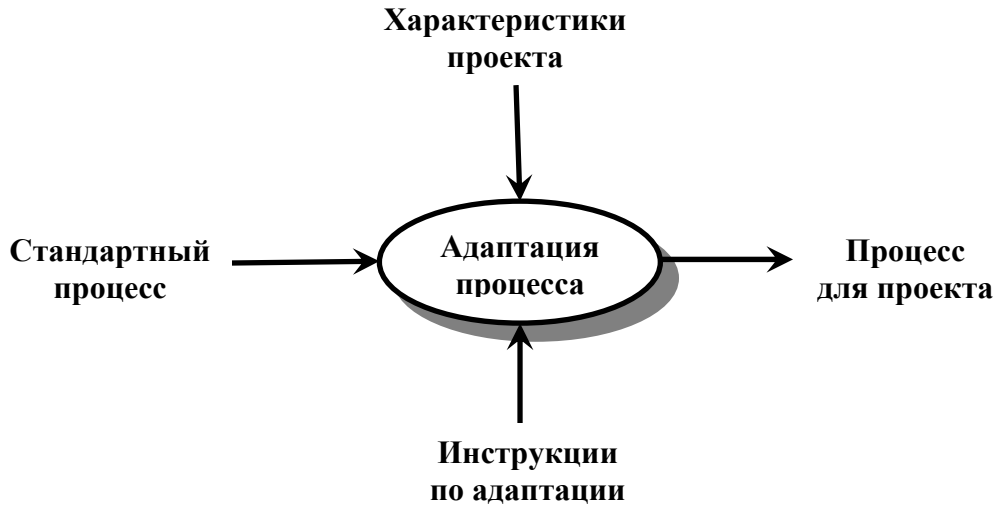


Рис. 6. Адаптация процесса

Для того чтобы проиллюстрировать необходимость адаптации, возьмем одно действие из фазы создания, входящей в процесс разработки, — "проведение экспертизы кода". Во многих случаях экспертиза кода начительно увеличивает ценность последнего, но иногда добавленная ценность не соизмерима с вынужденной трудоемкостью. Кроме того, экспертиза может выполняться группой (в соответствии с процедурой групповой экспертизы) или одним лицом. В стандартном процессе разработки не определяется, как следует выполнять экспертизу кода. Инструкции по адаптации могут помочь менеджеру проекта, если в них будет сказано, что действие "проведение экспертизы кода" относится только к определенным типам программ (сложным программам или внешним интерфейсам), и будет предложена форма экспертизы (групповая или одиночная).

Адаптация на суммарном уровне. В этом случае для адаптации стандартного процесса менеджер проекта применяет общие инструкции, опираясь на характеристики проекта. Иначе говоря, на этом уровне предоставляются некоторые общие правила, касающиеся отдельных типов детализированных действий. По отношению к проектам разработки для адаптации используются следующие характеристики:

• Уровень опыта и квалификация команды и менеджера проекта

• Максимальная численность команды

• Ясность требований

• Продолжительность проекта

• Критичность приложения

Уровень опыта команды считается высоким, если большинство членов команды имеют более чем двухлетнюю практику работы с технологией, используемой в проекте; в противном случае он считается низким. Критичность приложения считается высокой, если приложения оказывает на деятельность заказчика значительное влияние. Продолжительность проекта определяется как особенно короткая, если проект должен быть поставлен менее чем через три месяца. Инструкции суммарной адаптации предоставляются для различных значений этих характеристик. Суммарные инструкции обычно связаны с экспертизами, трудоемкостью, графиком работ, ресурсами или формализмом. Инструкции, касающиеся экспертиз, обычно указывают, когда и какую экспертизу следует выполнять. Аналогично, инструкции относительно трудоемкости предлагают шаги, которые нужно выполнить для проекта, чтобы повлиять на его трудоемкость. Эти общие инструкции устанавливают контекст для детальной адаптации процесса и выбора подходящего процесса для проекта.

Детальная адаптация. Детальная адаптация охватывает выполнение действий, их экспертизы и их потребности в документировании. Инструкции детальной адаптации могут определять действие как необязательное, и в этом случае менеджер проекта может решить, выполнять его или не выполнять. Аналогично подготовка некоторых документов может быть необязательной, и тогда менеджер проекта сам решает, нужен ли данный документ проекту. Для экспертизы существуют общие варианты: "провести групповую экспертизу", "провести одиночную экспертизу" и "не проводить экспертизу". Кроме того, менеджер проекта может добавить новые действия или повторить некоторые из них. В ходе детальной адаптации устанавливается последовательность выполняемых для проекта действий, которая затем используется для того, чтобы спланировать действия, составить для них график и сформировать основу для выполнения проекта. Проведение адаптации особо выделяется в плане проекта, поэтому когда

пересматривается план, определение и адаптация процесса также пересматриваются.

2.9. Выводы по второй главе.

1. В работе предложено процессы обучения описывать лабиринтной формальной комбинаторной моделью, конкретизация которой дает конкретные задачи обучения под разными углами. Для внешнего представления знаний предложено использование объектно ориентированного, предусмотренного для анализа и проектов, универсального языка моделирования UML. Для внутреннего представления знаний рассматриваются контекстно управляемые семантические сети, которые могут быть описаны с использованием XML-языка. В конкретных задачах обучения для преобразования структуризованных знаний предложен метод модульных трансформаций представлений.

2. Чем более структуризованно опишутся знания, тем больше шанс успешного завершения процессов обучения и использования знаний.

3. Предложено комплексное использование CMMI и PSIM в когнитивной системе, позволяющее сосредотачиться на динамике программного обеспечения, а также развитии и ведении систем. Рассмотрены способы вероятностной оценки риска и модели оценки трудоемкости проектов.

3. Инженерные аспекты реализации

3.1. Характеристики проекта

Проект – это временное предприятие, предназначенное для создания уникальных продуктов, услуг или результатов.

Термин "временное" означает, что у любого проекта есть четкое начало и четкое завершение. Завершение наступает, когда достигнуты цели проекта; или осознано, что цели проекта не будут или не могут быть достигнуты; или исчезла необходимость в проекте, и он прекращается.

В результате проекта получаются *уникальные* результаты поставки, представляющие собой продукты, услуги или результаты. В результате проекта могут получиться:

- Продукт и производимое изделие, которое можно измерить и которое может быть как конечным звеном производственной цепи, так и элементом.
- Способность предоставить услуги, такие как практические функции, способствующие производству или дистрибуции
- Результаты, такие как последствия или документы. Например, исследовательский проект получает данные, которые можно использовать для определения наличия тенденции или пользы нового процесса для общества.

Уникальность является важной характеристикой результатов поставки проекта.

Последовательная разработка – это свойство проектов, наравне с понятиями временности и уникальности. Последовательная разработка означает развитие по этапам и протекание по шагам. Например, содержание проекта формулируется в общих чертах на ранних стадиях проекта и впоследствии детализируется и конкретизируется по мере того как команда проекта разрабатывает более ясное и полное представление о целях проекта и результатах поставки.

Последовательную разработку спецификаций проекта необходимо тщательно согласовывать с правильным определением содержания проекта, особенно в случае выполнения проекта по контракту. Если содержание

проекта (т. е. состав работ, которые необходимо выполнить) определено правильно, то оно должно контролироваться по мере постепенного уточнения спецификаций продукта и проекта.

Управление проектами – это приложение знаний, навыков, инструментов и методов к операциям проекта для удовлетворения требований, предъявляемых к проекту[20]. Управление проектами выполняется с помощью применения и интеграции процессов управления проектами: инициации, планирования, исполнения, мониторинга и управления, завершения. Менеджер проекта – это лицо, ответственное за достижение целей проекта.

В управление проектом входит:

- Определение требований
- Установка четких и достижимых целей
- Уравновешивание противоречащих требований по качеству, содержанию, времени и стоимости
- Коррекция характеристик, планов и подхода в соответствии с мнением и ожиданиями различных участников проекта.

Многие знания, инструменты и методы, используемые в управлении проектами, применяются исключительно в этой области. К их числу относятся иерархические структуры работ, анализ критического пути и управление освоенным объемом. Однако одного только понимания и применения знаний, навыков, инструментов и методов, которые обычно считаются хорошей практикой, недостаточно для эффективного управления проектами. Для эффективного управления проектами необходимо, чтобы команда управления проектами понимала и использовала знания и навыки как минимум пяти экспертных областей:

- Свод знаний по управлению проектами
- Знания, стандарты и нормативные акты, относящиеся к данной области приложения
- Понимание окружения проекта
- Знания и навыки в области общего менеджмента
- Навыки межличностных отношений.

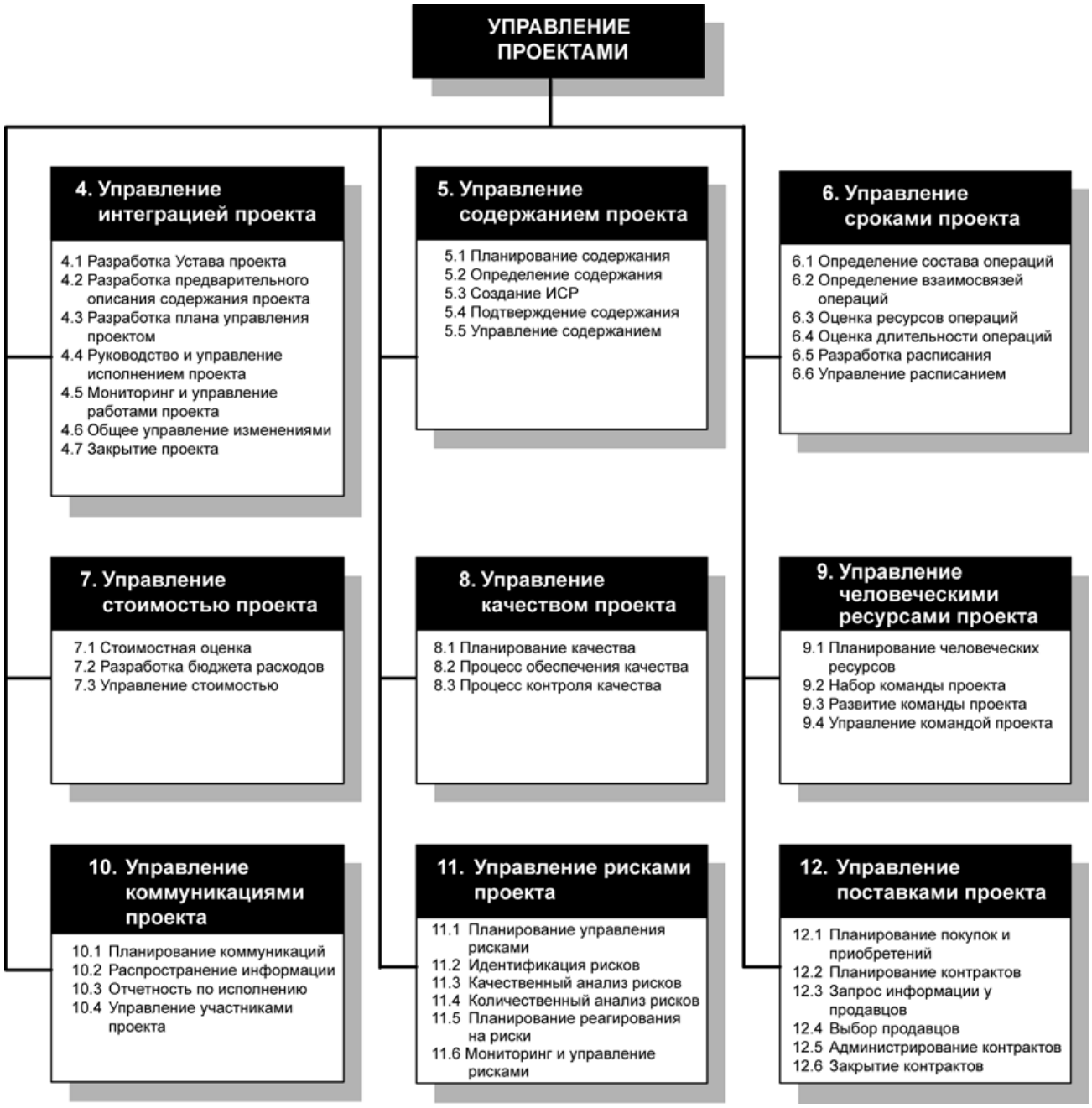


Таблица 4. Обзор областей знаний по управлению проектами и процессов управления проектами.

На рис. 7 изображены отношения между этими пятью экспертными областями. Хотя они выглядят как отдельные элементы, обычно они перекрываются и не могут существовать независимо. Эффективные команды проекта включают их во все аспекты проекта. Каждый из членов команды проекта не обязан быть экспертом во всех пяти областях. Кроме того,

маловероятно, чтобы кто-либо один обладал всеми знаниями и навыками, необходимыми для проекта.

Знания по управлению проектами, описанные в *Руководстве PMBOK®* включают в себя следующие элементы [20]:

- Определение жизненного цикла проекта
- Пять групп процессов управления проектом
- Девять областей знаний.

Менеджеры проекта или организация могу разделить проект на фазы, чтобы обеспечить более качественное управление с соответствующими отсылками на текущие операции исполняющей организации. Совокупность этих фаз составляет жизненный цикл проекта. Многие организации во всех своих проектах используют определенный набор жизненных циклов.

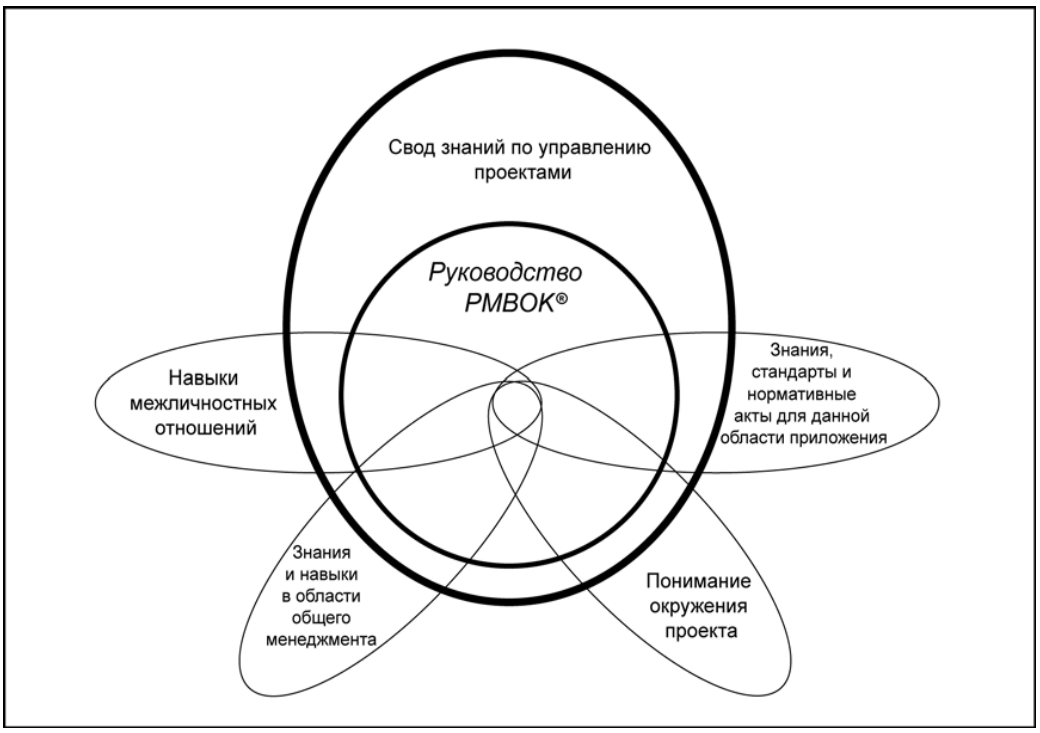


Рис. 7. Отношения между пятью экспертными областями.

Характеристики жизненного цикла проекта

Жизненный цикл проекта определяет фазы, которые связывают начало проекта с его завершением. Например, когда организация обнаруживает благоприятную возможность, которую она хотела бы использовать, она часто

авторизует анализ осуществимости, чтобы решить, следует ли браться за выполнение проекта.

Определение жизненного цикла проекта может помочь менеджеру проекта решить, следует ли считать анализ осуществимости первой фазой проекта или выделить его в отдельный проект. Когда результат этого предварительного анализа не очевиден, лучше выделять его в отдельный проект. Фазы жизненного цикла проекта не совпадают с группами процессов управления проектом.

3.2. База данных процессов

База данных процессов (PDB) представляет собой постоянное хранилище данных о производительности процессов из проектов, которые могут использоваться для планирования и оценки проекта, анализа продуктивности и качества, а также в других целях. PDB состоит из данных, полученных в завершенных проектах, и каждый проект предоставляет одну запись данных. Как нетрудно сообразить, для заполнения PDB необходимо собрать данные, проанализировать их, а затем сформировать элемент в базе данных.

Содержимое базы данных процессов

Менеджеры проектов, используя при планировании сведения из PDB, особенно полезной находят информацию об аналогичных проектах. Для того чтобы учесть возможность проверок на аналогии, в PDB необходимо зафиксировать общую информацию о проекте, в том числе указать используемые языки программирования, платформы, базы данных, инструменты, размер ПО и трудоемкость. Имея такую информацию, менеджер проекта может найти данные по всем проектам, которые, к примеру, относятся к определенной предметной области, используют конкретную систему управления базами данных (СУБД), конкретный язык программирования или ориентированы на определенную платформу.

Вы поможете планированию проекта, если будете фиксировать данные о трудоемкости, ошибках, графиках работ, рисках и т. д. Если известны общая трудоемкость проекта, его размер и распределение

трудоемкости по фазам, эти сведения могут оказаться полезными для оценивания трудоемкости в новом проекте.

Таким образом, данные, записываемые в PDB, можно классифицировать следующим образом:

• Характеристики проекта

• График работ по проекту

• Трудоемкость проекта

• Размер ПО

• Ошибки

Характеристики проекта включают его название, имена менеджера проекта и лидеров модулей (благодаря этому с ними можно связаться и получить дополнительную и уточняющую информацию), подразделение (это позволяет проводить анализ по подразделениям), размещенный процесс (дает возможность анализировать процессы по отдельности), предметную область, аппаратную платформу, используемые языки и СУБД, краткое изложение целей проекта, информацию о его рисках, продолжительность выполнения проекта и численность команды.

Данные графика работ — это в первую очередь предполагавшиеся и фактические даты начала и конца проекта. Данные о трудоемкости проекта включают информацию о первоначальной оценке трудоемкости и об общей фактической трудоемкости, а также распределение фактической трудоемкости по таким стадиям, как запуск проекта, управление требованиями, проектирование, создание, тестирование элементов и другим. Размер разработанного ПО можно выразить числом строк кода (lines of code, LOC), количеством простых, средних и сложных программ или сочетанием этих показателей. Даже если число функциональных точек (FP) не оценивалось, можно вычислить унифицированный показатель продуктивности, представив окончательный размер в виде числа функциональных точек, которое, как правило, можно получить преобразованием полученного размера ПО (в LOC) по опубликованным таблицам преобразования. Размер окончательной системы, выраженный числом функциональных точек, также фиксируется. Данные об ошибках включают число ошибок, найденных разными методами их обнаружения, и число ошибок, внесенных на разных стадиях. Следовательно, фиксироваться должно число ошибок разного вида,

обнаруженных при экспертизе требований, экспертизе кода, тестировании элементов и в других фазах.

Кроме того, записываются замечания, в том числе по оцениванию (например, критерий, использованный для разделения программ на простые, средние и сложные), и замечания по управлению рисками (допустим, как восприятие риска менялось в ходе выполнения проекта).

Имущество процесса и система совокупности знаний

Как говорилось, процесс включает опыт организации в форме "рецептов" успеха. Однако описание процессов обычно содержит последовательность необходимых действий, список тех, кто их выполняет, критерии на входе и на выходе для основных шагов и т.д. Использование процесса облегчается инструкциями, контрольными перечнями и шаблонами. Вместе все эти материалы называются имуществом процесса.

Инструкции обычно устанавливают правила и процедуры выполнения шага. Например, один из шагов процесса планирования проекта — оценка трудоемкости. Чтобы выполнить это действие, менеджеру проекта нужны инструкции: контрольные перечни действий и контрольные перечни экспертизы. Как следует из названия, контрольный перечень действия представляет собой список действий, составляющих шаг процесса. Цель перечня экспертизы — привлечь внимание экспертов к ошибкам, которые с большой вероятностью могут обнаружится в конечном продукте. Шаблоны обычно предоставляют структуру документа, в котором можно зафиксировать результат процесса или шага. На рис. 8 показаны взаимосвязи между процессом и его имуществом.

Рис. 8. Имущество процессов

Главное назначение имущества процесса состоит в облегчении использования процессов через экономию затрат труда и, следовательно, в улучшении продуктивности. Например, при создании документа гораздо проще и быстрее применить шаблон, чем разрабатывать документ "с чистого листа". Имущество процесса также помогает повысить качество проекта: сначала предоставляя надлежащие инструкции и контрольные перечни действий, а затем — помогая экспертам как можно раньше обнаружить ошибки.

Короче говоря, для того чтобы в полной мере воспользоваться преимуществами подхода к выполнению проекта, ориентированного на процессы, важно накапливать и использовать имущество процесса.

Обычно собирается и становится доступным через отдельную систему следующее имущество:

• План управления проектом

• План управления конфигурацией

• График работ

• Стандарты, контрольные перечни, инструкции, шаблоны и другие пособия

• Разработанные инструменты и замечания по ним

• Обучающие материалы

• Другие документы, которые могут повторно использоваться в будущих проектах

Хотя в имуществе процесса — контрольных перечнях, шаблонах и т.п. —должен отражаться опыт организации, эти документы не всегда могут зафиксировать разнообразные формы знания, приобретенного при выполнении проектов. Для фиксирования разных форм знания и его повторного использования требуется надлежащее управление знанием, которое стало важным в таких базирующихся на информации организациях, как поставщики решений и консалтинговые компании.

3.3. Коллективная разработка с использованием Visual Studio Team Foundation Server (TFS)

Коллективная разработка. Введение в среду для коллективной разработки

Успех групповых проектов разработки ПО обеспечивает сочетание многих элементов, процессов и ролей. Основное внимание следует обратить на:

- Процесс разработки
- Процесс сборки
- Процесс управления проектом

Следующая диаграмма иллюстрирует отношения между типовыми процессами коллективной разработки ПО и тем, как может использоваться Team Foundation Server для обеспечения единообразной фундаментальной поддержки этих инициатив [24].

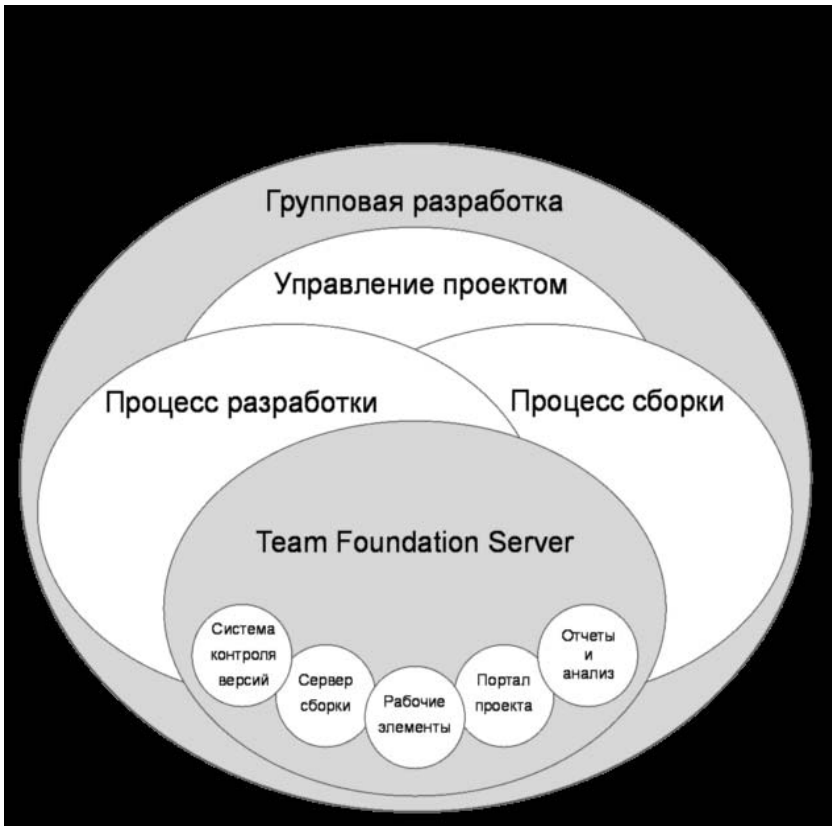


Рис. 9. Отношения между типовыми процессами коллективной разработки ПО.

На следующей диаграмме представлен пример логической реализации Team Foundation Server с точки зрения наиболее типичных ролей в разработке ПО и жизненном цикле разработки.

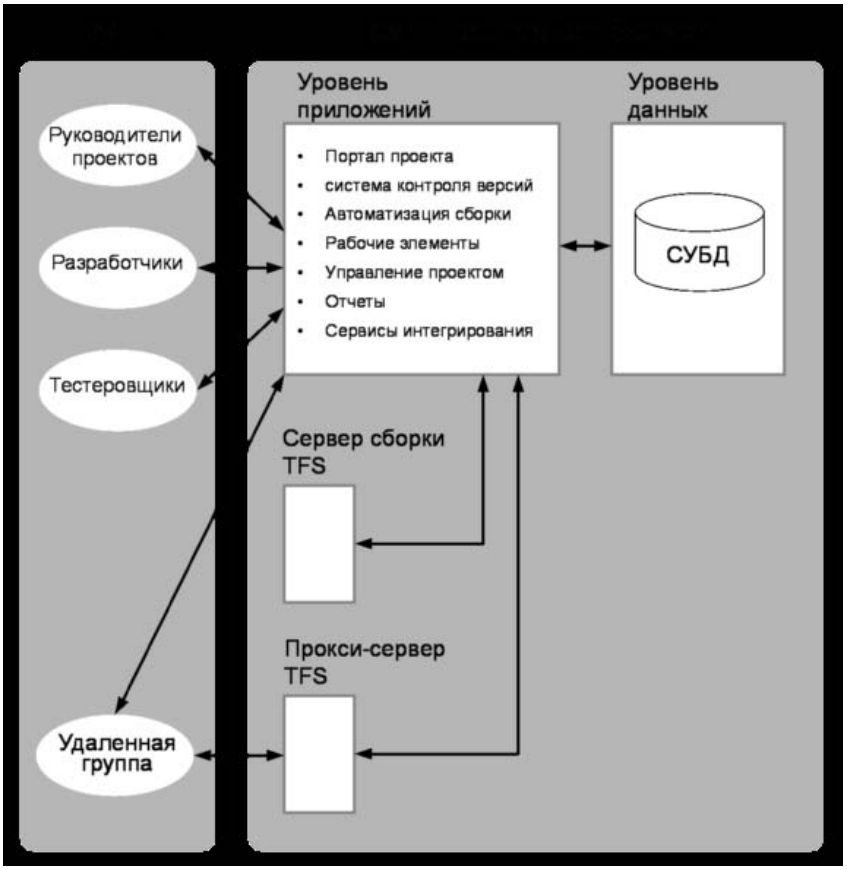


Рис. 10. Пример логической реализации Team Foundation Server с точки зрения наиболее типичных ролей.

В данной главе описано использование Team Foundation Server (TFS) и Microsoft Visual Studio Team System (VSTS) в условиях коллективной разработки программного обеспечения. Здесь представлены основные характеристики TFS и VSTS и взаимодействие групп разработки и тестирования при разработке программного обеспечения. TFS интегрирует в себе системы контроля версий, отслеживания процесса работы над проектом, создания и отображения отчетов, управления проектом и автоматизированный процесс сборки, и, следовательно, повышает эффективность работы группы разработки.

Коллективная разработка ПО включает множество процессов, и лишь их правильное сочетание обеспечивает эффективные условия работы. Выделим основные процессы:

- Разработка
- Тестирование

- Сборка
- Развертывание
- Выпуск

Процесс разработки программного обеспечения с использованием Team Foundation Server

С помощью TFS группа разработки может хранить исходный код в централизованно управляемом хранилище. Тогда есть возможность создавать сборки с использованием сервера сборки и исходных файлов из этого хранилища и затем передавать их группе тестирования.

На рис. 11 показан процесс коллективной разработки ПО с использованием TFS и взаимосвязь сред разработки и тестирования.

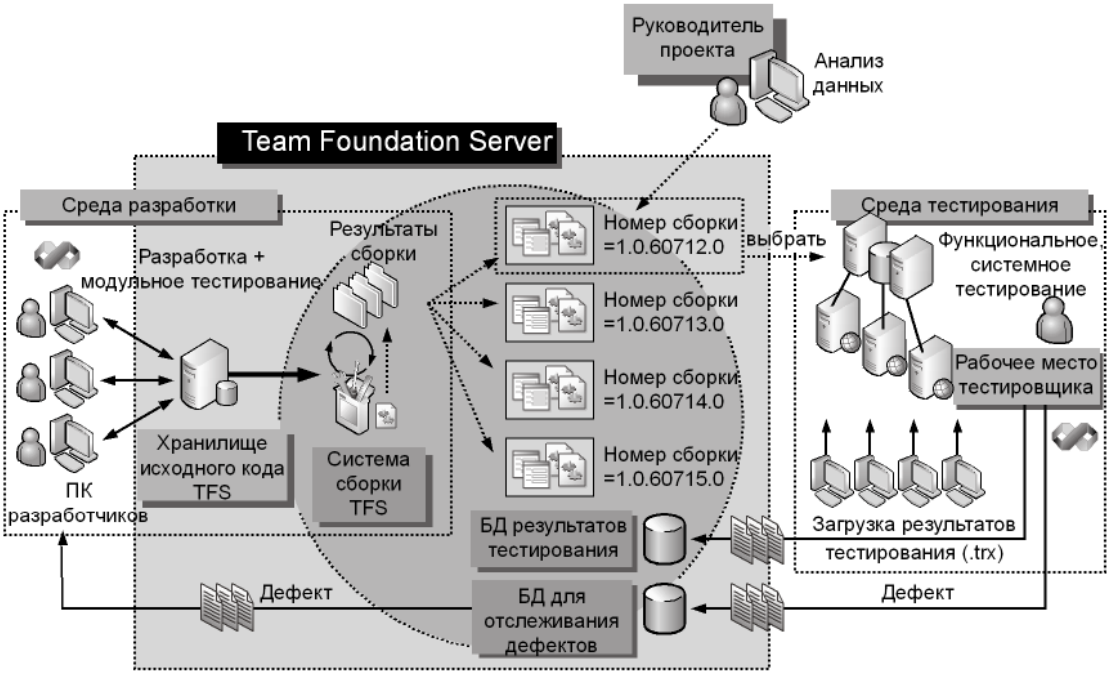


Рис. 11. Процесс коллективной разработки ПО с использованием Team Foundation Server.

Группа тестирования берет версии сборок приложений из места публикации результатов сборки и выполняет их в своей среде тестирования, сочетая ручное и автоматизированное тестирование. TFS сохраняет результаты тестирования и использует их для обеспечения обратной связи по качеству сборки. Группа тестирования также может создавать рабочие элементы или дефекты (особый тип рабочих элементов), по которым группа разработки должна предпринять некоторые действия. Эти рабочие элементы позволяют группе тестирования отслеживать работу группы разработки.

В больших организациях с несколькими группами разработки каждая группа использует отдельный TFS, включая хранилища исходного кода и сервера сборки. На рис. 12 показан пример взаимодействия двух групп разработки, передающих сборки приложений группе тестирования.

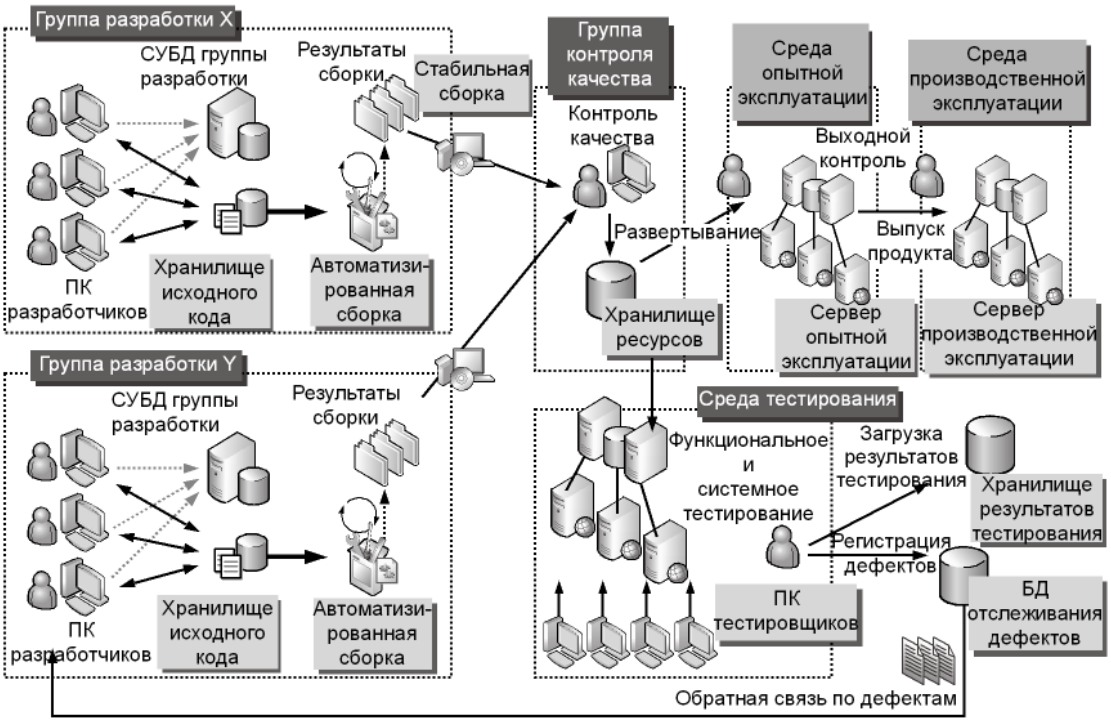


Рис. 12. Взаимодействие двух групп разработки и группы тестирования.

Каждая группа разработки помещает плановые сборки в место публикации результатов сборки. Группа тестирования выполняет тестирование этих сборок и определяет их качество. Пройдя контроль качества, приложения развертываются на сервере опытной эксплуатации для окончательной проверки и одобрения пользователями. После этого они поступают на сервер производственной эксплуатации.

В ходе процесса разработки ПО можно выделить ряд ключевых взаимодействий разработчиков с TFS. Например, как разработчик вы взаимодействуете с TFS следующим образом:

- Осуществляете доступ к дефектам и задачам, находящимся в TFS, и таким образом выясняете, что необходимо сделать. Например, рабочие элементы могут быть назначены руководителем проекта, другим разработчиком или группой тестирования.

- Используете VSTS Source Control Explorer для доступа к хранилищу исходного кода в TFS и загрузки последней версии исходного кода в локальное рабочее пространство или свой рабочий компьютер.
- Выполнив задачи, оговоренные рабочим элементом, возвращаете свой код в хранилище исходного кода.
- Check-in2-событие (событие регистрации изменений) может запустить процесс непрерывной интеграции, который использует Team Build.
- Если сборку создать не удалось, создается новый рабочий элемент для отслеживания неудачного создания сборки.

Член группы тестирования взаимодействует с TFS следующим образом:

- Загружает плановую сборку из места публикации результатов сборки.
- С помощью различных инструментов VSTS выполняет ручное и автоматизированное тестирование, включая тестирование безопасности, производительности и Web-тестирование.
- Загружает результаты тестирования в базу данных TFS Test Result для дальнейшего использования.
- Регистрирует в TFS дефекты, выявленные при тестировании, как рабочие элементы.
- Объявляет открытые дефекты исправленными после тестирования последней версии сборки.

Физические среды разработки и тестирования

Размер сред разработки и тестирования и количество компьютеров в них могут быть различными в зависимости от размера группы и проекта. На рис. 13 показана типичная физическая среда разработки и тестирования.

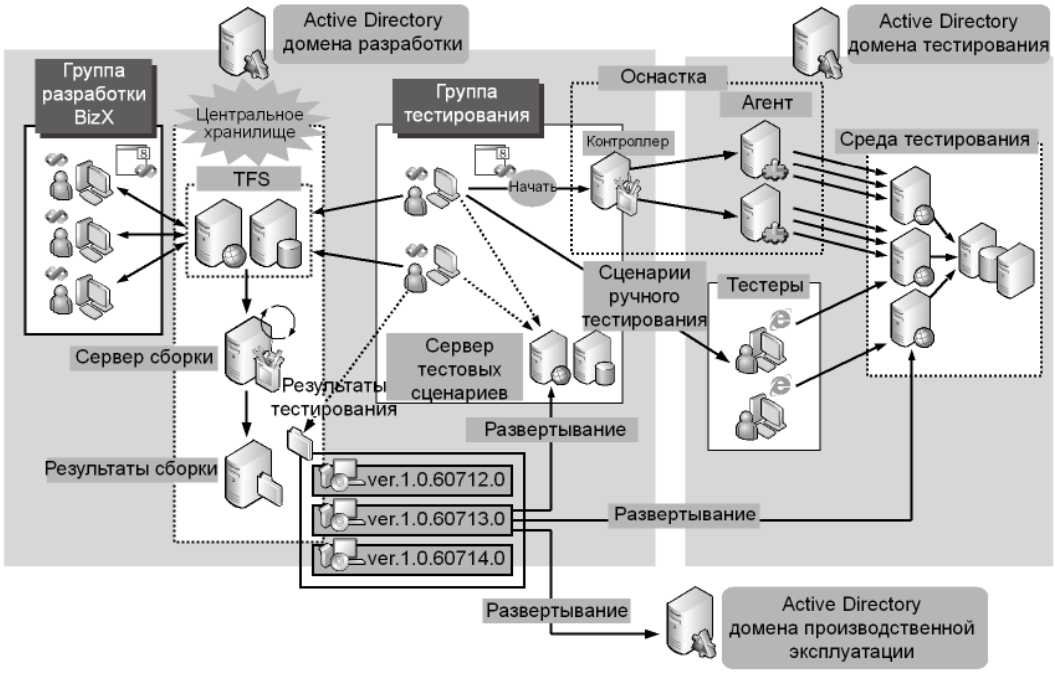


Рис. 13. Физическая среда разработки и тестирования.

Среда разработки обеспечивает процессы разработки и сборки продукта. Среда разработки включает следующие компьютеры:

- Team Foundation Server
- Сервер сборки
- Сервер для хранения результатов сборки
- Рабочие станции разработчиков

Если группа разработки работает с TFS удаленно или если эта группа особенно велика, что обусловливает проблемы с производительностью центрального сервера TFS, для повышения производительности можно настроить прокси-TFS.

Среда тестирования состоит из одной или более тестовых рабочих станций, на которых установлена Visual Studio Team Edition for Software Testers. Они используются для управления процессом тестирования и проведения функционального тестирования, системного тестирования, тестирования безопасности, производительности и Веб-тестирования. Члены группы используют TFS для управления, дефектами и другими рабочими элементами, а также результатами тестирования.

Среда тестирования также может включать Visual Studio Team Test Load для нагрузочного тестирования.

Архитектура Team Foundation Server

TFS использует трехуровневую архитектуру, которая включает клиентский уровень, уровень приложений и уровень данных. Клиенты TFS взаимодействуют с уровнем приложений посредством различных Веб-сервисов; уровень приложений, в свою очередь, поддерживается различными базами данных уровня данных. На рис. 14 показаны компоненты и взаимодействия всех уровней TFS.

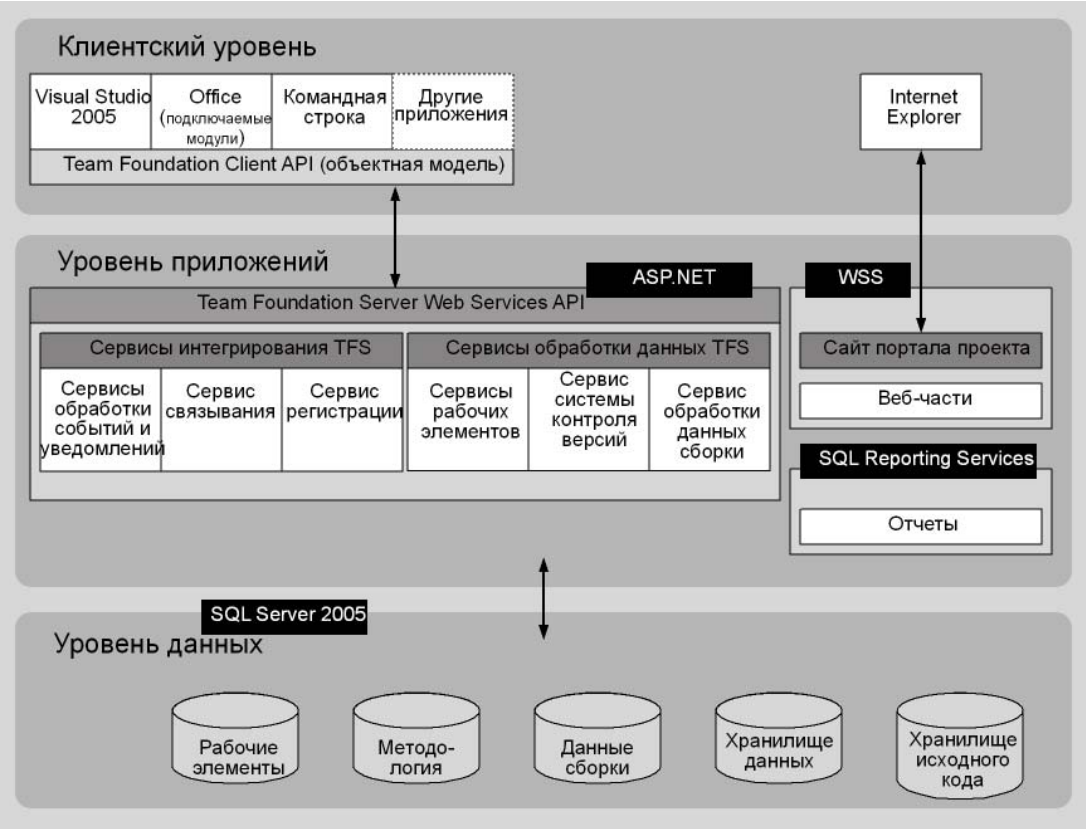


Рис. 14. Компоненты и уровни TFS.

Клиентский уровень

Клиентский уровень включает следующие компоненты:

- *Объектная модель Team Foundation Server.* Это открытый API, используемый для взаимодействия с TFS. Объектная модель может использоваться для создания клиентских приложений, взаимодействующих с TFS.
- *Компоненты Visual Studio Industry Partners (VSIP).* Это инструментальные средства, надстройки и языки программирования сторонних производителей для Visual Studio.

- *Интеграция с Microsoft Office.* Включает ряд надстроек Microsoft Office Excel® и Microsoft Office Project, которые обеспечивают возможность запрашивать и обновлять рабочие элементы в базе данных TFS Work Item Tracking. Особенно полезна для руководителей проектов, которые уже активно используют эти приложения.
- *Инструментальные средства командной строки.* Это инструментальные средства, обеспечивающие возможность взаимодействия с TFS из командной строки. Преимущественное большинство этих инструментов предоставляют функциональность контроля версий и используются для автоматизации повторяющихся и выполняющихся по расписанию задач.
- *Инфраструктура политик регистрации изменений файла в системе контроля версий.* Поддерживает политики регистрации изменений, которые являются расширяемым механизмом проверок исходного кода в процессе регистрации изменений.

Уровень приложений

Уровень приложений предоставляет Веб-сервисы ASP.NET, с которыми взаимодействует клиентский уровень. Эти Веб-сервисы не предназначены для использования в сторонних приложениях, но приведены здесь для полноты картины. Веб-сервисы сгруппированы в следующие наборы:

- *Сервисы обработки данных TFS* (Team Foundation Data Services)
- *Сервисы интегрирования TFS* (Team Foundation Integration Services)

Сервисы обработки данных TFS

Эти Веб-сервисы преимущественно обеспечивают взаимодействие с уровнем данных. К этим сервисам относятся:

- *Веб-сервис контроля версий.* Клиентский уровень использует этот Веб-сервис для выполнения различных операций контроля версий и взаимодействия с базой данных исходного кода.
- *Веб-сервис для отслеживания рабочих элементов.* Клиентский уровень использует этот Веб-сервис для создания, обновления рабочих элементов и запросов к базе данных отслеживания рабочих элементов.
- *Веб-сервис сборки.* Клиентский уровень и инфраструктура MSBuild используют этот Веб-сервис для выполнения процессов сборки приложений.

Сервисы интегрирования TFS

Данный набор Веб-сервисов обеспечивает функциональность интегрирования и автоматизации. Эти сервисы не взаимодействуют с уровнем данных. К сервисам интегрирования TFS относятся:

- *Веб-сервис регистрации.* Этот сервис используется для регистрации других сервисов TFS. Он обслуживает информацию регистрационной базы данных. Эта информация используется сервисами для определения способа взаимодействия друг с другом.
- *Веб-сервис безопасности.* Этот сервис состоит из Сервиса групповой безопасности (Group Security Service) и Сервиса авторизации (Authorization Service). Сервис групповой безопасности используется для управления всеми пользователями и группами TFS. Сервис авторизации предоставляет TFS систему управления доступом.
- *Веб-сервис связывания.* Этот сервис обеспечивает возможность устанавливать отношения слабой связи (или просто «связи») между элементами данных инструментальных средств. Например, соответствие между рабочим элементом (например, дефектом), и исходным кодом, который был изменен с целью исправления этого дефекта, устанавливается в TFS при помощи связи.
- *Веб-сервис обработки событий.* Этот сервис обеспечивает возможность инструментальному средству или сервису регистрировать типы событий. Пользователи могут подписываться на эти события и получать уведомления по электронной почте или посредством вызова Веб-сервиса. Например, событие регистрации изменений может использоваться для запуска процесса непрерывной интеграции.
- *Веб-сервис классификации.* Этот сервис совместно с Веб-сервисом связывания обеспечивает классификацию артефактов TFS соответственно установленным систематикам. Это обеспечивает возможность поддерживать создание перекрестных отчетов даже для артефактов, не использующих общую систематику для организации своих данных. Например, если обычно рабочие элементы группируются по проектным группам, а тесты – по компонентам, то с помощью классификации можно организовать тесты по проектным группам, так они смогут быть включены в тот же отчет, что и рабочие элементы.

Уровень данных

TFS не поддерживает прямого доступа клиентских приложений к уровню данных. Все запросы к данным осуществляются через Веб-сервисы на уровне приложений. Уровень данных TFS состоит из следующих хранилищ данных, соответствующих сервисам обработки данных уровня приложений.

- *Отслеживание рабочих элементов.* Здесь хранятся все данные, касающиеся рабочих элементов.
- *Контроль версий.* Здесь хранятся все данные, касающиеся контроля версий.
- *Сборка.* Содержит всю информацию, касающуюся инструмента TFS Team Build.
- *Хранилище отчетов.* Хранит информацию, касающуюся всех инструментов и функций TFS. Хранилище отчетов упрощает создание отчетов, сочетающих в себе данные нескольких инструментальных средств.

Схема развертывания

Развертывание TFS можно осуществлять, используя различные схемы, начиная от установки на один сервер, заканчивая более сложными многосерверными топологиями. Независимо от используемой схемы должен быть учтен ряд ключевых требований.

3.4. Структурирование проектов и решений в системе контроля версий Team Foundation Server (TFS)

Стратегии структурирования решений и проектов

Чаще всего используются три стратегии структурирования файлов решений и проектов:

- Одиночное решение. При разработке небольшой системы, создается одиночное решение, в котором размещаются все проекты.

- Сегментированное решение. При разработке большой системы взаимосвязанные проекты группируются в разные решения. Для каждой логической группы проектов, с которой разработчик, вероятнее всего, будет работать как с совокупностью, создается отдельное решение. Затем все эти решения объединяются в одно главное решение, которое будет содержать все проекты. При таком подходе сокращается количество данных, извлекаемых из системы контроля версий, поскольку работа ведется только над определенными проектами.
- Несколько решений. При создании очень большой системы, требующей десятков и более проектов, следует работать с подсистемами. Для отображения зависимостей и из соображений производительности не нужно создавать главное решение, содержащее все проекты.

В общем, следует:

- Использовать стратегию одиночного решения, если размер получаемого в результате решения не слишком велик и не приводит к проблемам загрузки в Visual Studio.
- Использовать несколько решений для создания отдельных представлений подсистем приложения.
- Использовать несколько решений для сокращения времени загрузки решения и сокращения времени сборки для разработчиков.

При разработке структуры проектов и решений следует иметь в виду следующее:

- Каждый проект во время компиляции создает отдельную сборку (assembly). Необходимо начать с определения того, какие сборки потребуется создать, и затем, исходя из этого, принимайте решение о необходимых проектах. На основании этого распределите код по проектам.

- Необходимо начать с самой простой структуры одиночного решения. Усложнять структуру только в том случае, когда это действительно необходимо.
- При проектировании структуры с множеством решений:
  - ✓ Рассмотреть зависимости проекта. Пытаться сгруппировать взаимосвязанные проекты в одно решение. Это позволит использовать в решении ссылки на проекты, а не на файлы, что обеспечивает возможность Visual Studio синхронизировать конфигурации сборки (отладка/ версия для выпуска) и, отслеживая версии, определять, когда необходимо повторно собрать проект. Пытаться свести к минимуму количество перекрестных ссылок на проекты между решениями.
  - ✓ Рассмотреть возможность совместного использования исходного кода. Поместить проекты, использующие один и тот же исходный код, в одно решение.
  - ✓ Учесть структуру группы. Решения должны быть структурированы таким образом, чтобы упростить группам работу с набором взаимосвязанных проектов.
- Придерживаться плоской структуры проекта. Это облегчит задачу по группировке проектов в решения без необходимости внесения изменений в структуру каталогов или папку системы контроля версий.

Одиночное решение

При работе над небольшой системой рекомендуется размещать все проекты в одном решении Visual Studio. Такая структура упрощает разработку, потому что при открытии решения доступен весь исходный код. При такой стратегии также очень легко работать со ссылками, потому что все они являются ссылками на проекты одного решения. Но все-таки, возможно, придется использовать ссылки на файлы сборок сторонних

производителей, например на купленные компоненты, находящиеся вне решения. На рис. 15 показан подход с использованием одиночного решения.

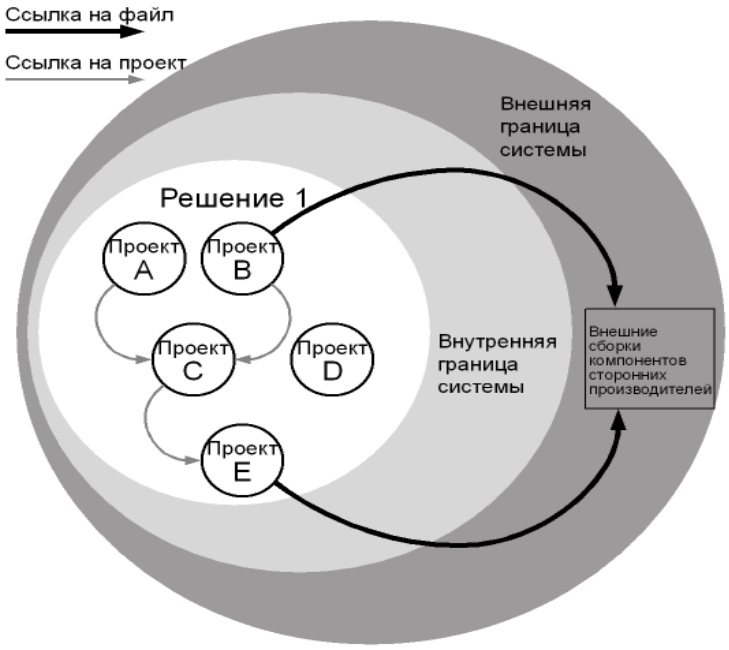


Рис. 15. Подход с использованием одиночного решения.

Главными причинами выбора данной структуры являются:

- Простые сценарии сборки.
- В рамках решения можно без труда отображать зависимости между проектами.

Такая структура должна использоваться, если все разработчики работают с одним и тем же решением и располагают одним и тем же набором проектов. Это может быть проблематичным для больших систем, где требуется организовать проекты по подсистемам или по функциональным возможностям.

Сегментированное решение

При работе над большой системой рекомендуется использовать несколько решений, каждое из которых будет представлять некую подсистему приложения. Разработчики могут использовать эти решения и работать над отдельными небольшими частями системы, в этом случае им не придется загружать код всех проектов. Структура решения должна быть

спроектирована таким образом, чтобы все взаимосвязанные проекты располагались в одной группе. Это позволит использовать ссылки на проекты, а не на файлы. Также можно создать один файл главного решения, содержащий все проекты. Этот файл используется для сборки всего приложения. В отличие от предыдущих версий, Visual Studio 2005 полагается в сборке проектов на MSBuild. Начиная с Visual Studio 2005, появилась возможность создавать структуры решений, не включающие в себя все проекты, на которые имеются ссылки, и такие решения все равно будут собираться без ошибок. Поскольку главное решение собирается первым, формируя результирующие двоичные файлы каждого проекта, MSBuild может прослеживать ссылки на проекты вне границ решения и успешно выполнять сборку. Но это возможно только в случае использования ссылок на проекты, а не ссылок на файлы. Созданные таким образом решения можно успешно собирать из командной строки Visual Studio и из IDE, но не с помощью Team Build. Чтобы успешно создать сборку с помощью Team Build, необходимо использовать главное решение, включающее все проекты и зависимости. На рис. 16 показан подход с использованием сегментированного решения.

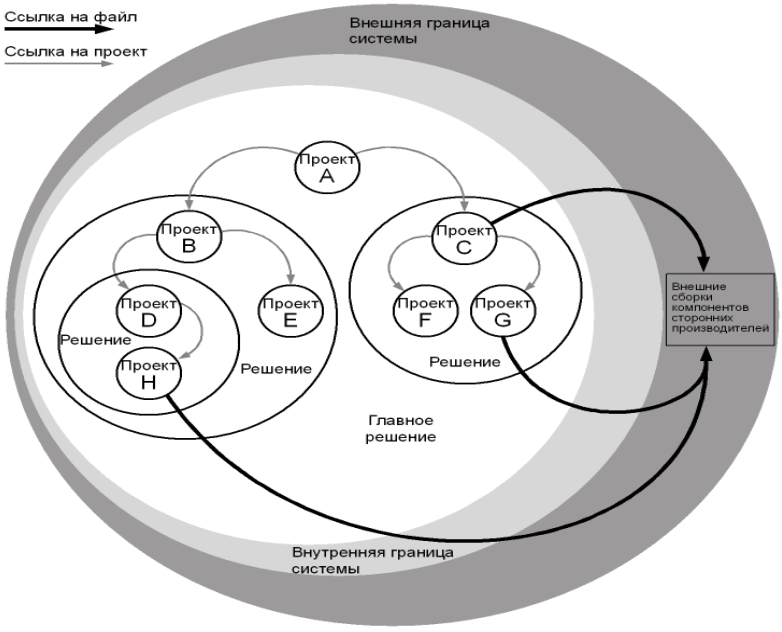


Рис. 16. Подход с использованием сегментированного решения.

При работе с несколькими решениями все проекты должны иметь плоскую структуру. Типичный пример – приложение, включающее проект Microsoft Windows® Forms, проект ASP.NET, службу Windows и ряд библиотек классов, которые совместно используются некоторыми или всеми перечисленными проектами. Для всех проектов может использоваться следующая плоская структура:

```
/Source
        /WinFormsProject
        /WebProject
        /WindowsServiceProject
        /ClassLibrary1
        /ClassLibrary2
        /ClassLibrary3
        Web.sln
        Service.sln
        All.sln
```

Плоская структура обеспечивает большую гибкость и возможность использования решения для создания разных представлений проектов. Физическую структуру каталогов решения менять очень трудно, особенно если используется библиотека классов из другого решения.

Основания использования этой структуры:

- Повышение производительности при загрузке и сборке составляющих решений.
- Возможность использования составляющих решений для создания представлений наборов проектов, созданных определенной подгруппой, или на основании границ совместного использования кода.
- Возможность использования главного решения для сборки всего приложения.
- Возможность без труда отображать зависимости между проектами в каждом составляющем решении.

- Упрощение системы в целом, если решения выделены логично. Например, если решение выделено соответственно технологическим или функциональным характеристикам, новым разработчикам намного проще понять, над каким из решений работать.

Основная причина не использовать эту структуру:

- Повышение затрат на обслуживание решения. Введение нового проекта может повлечь за собой изменение файлов многих решений.

Несколько решений

При работе над очень большим решением, включающим десятки проектов, может возникнуть проблема с масштабируемостью решения. При таком сценарии следует разбить приложение на несколько решений, но при этом не создавать главного решения для всего приложения, потому что все ссылки внутри решений являются ссылками на проекты. Ссылки на проекты вне решений (например, на библиотеки сторонних производителей или проекты из другого составляющего решения) – это ссылки на файлы, т.е. без «главного» решения можно обойтись. Вместо главного решения необходимо использовать сценарий, который понимает порядок сборки решений. Одна из главных задач при обслуживании структуры с несколькими решениями – гарантировать невозможность создания циклических ссылок между решениями. Эта структура требует сложных сценариев сборки и явного отображения зависимостей. При такой структуре невозможно создать сборку всего приложения в Visual Studio. Это делается непосредственно из TFS Team Build или MSBuild. На рис. 17 показан подход с использованием нескольких решений.

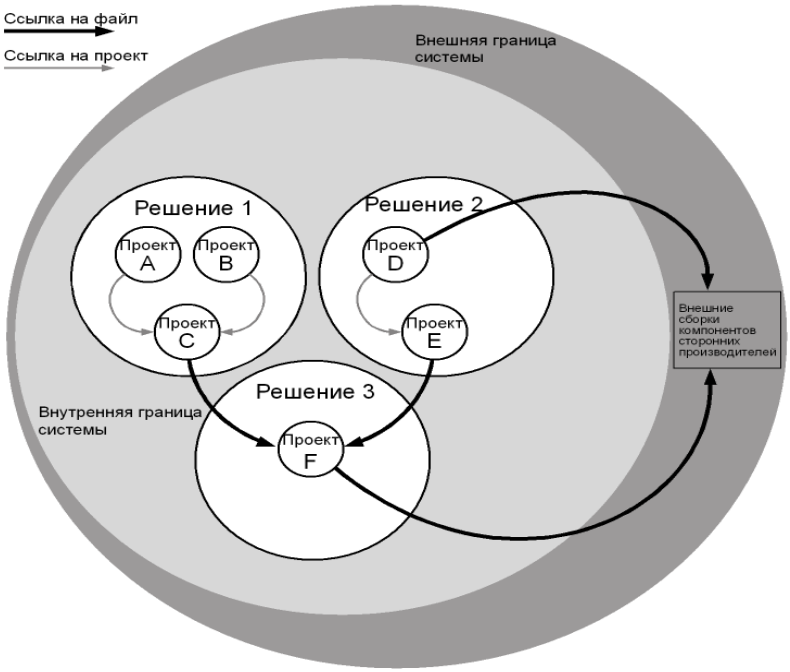


Рис.17. Подход с использованием нескольких решений.

Такая структура должна использоваться для очень больших приложений. Она поможет решить проблемы с производительностью и масштабируемостью Visual Studio IDE.

3.5. Выводы по третьей главе.

1. В данной главе дается обзор областей знаний по управлению проектами и процессов управления проектами.
2. Менеджер проекта может лучше спланировать свой проект, если имеет доступ к опыту, накопленному в завершенных проектах. Инфрастуктура планирования проекта помогает эффективно собирать данные и полученные уроки и делать их доступными для менеджеров проектов.
3. База данных процессов содержит данные по производительности завершенных проектов. Она включает данные по рискам, трудоемкости и ее распределению, ошибкам и их распределению, размеру ПО и другим характеристикам проектов.

4. Базовая линия устойчивости процесса суммирует производительность процесса в проектах, определяя тем самым диапазон результатов, который можно ожидать при соблюдении процесса. Базовая линия устойчивости процесса содержит такие показатели, как качество, продуктивность, эффективность устранения ошибок, распределение трудоемкости и ошибок.
5. Имущество процесса — это такие документы, как контрольные перечни, шаблоны, методологии и инструкции. Они повышают продуктивность, сокращая трудоемкость некоторых задач, и улучшают качество, позволяя снизить число ошибок или обнаружить их раньше.
6. Рассматривается процесс коллективной разработки с использованием Team Foundation Server и структурирование проектов и решений в системе контроля версий TFS .

4. Практическая реализация основных положений диссертационой работы

4.1. Архитектура системы обучения методам коллективной разработки программного обеспечения

С целью облегчения процесса обучения коллективным методам разработки программного обеспечения предлагается использование интеллектуальной обучающей/когнитивной системы.

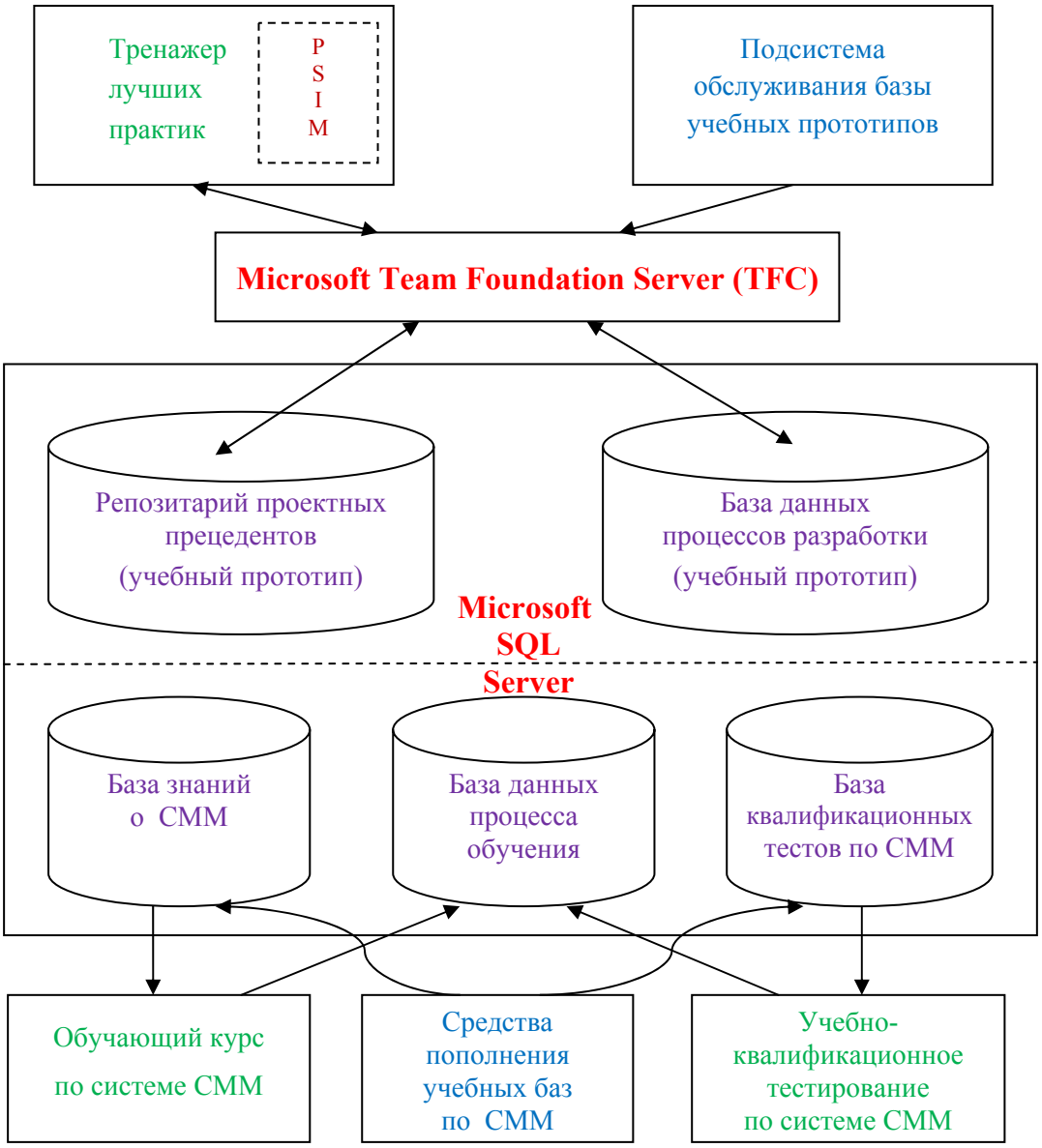


Рис.18. Архитектура системы обучения коллективным методам разработки программного обеспечения.

Правильная работа системы в значительной степени определяется одним из основных ее компонентов – информационной базы системы. Это сложный компонент, содержащий знания о проектных прецедентах (лучших практиках), процессах, квалификационных тестах. База данных процессов содержит данные по производительности завершенных проектов. Она включает данные по рискам, трудоемкости, ошибкам и их распределению, размеру ПО и другим характеристикам проектов. Менеджер проекта может лучше спланировать свой проект, если имеет доступ к опыту, накопленному в завершенных проектах. Для этого существует репозитарий проектных прецендентов. В качестве языка представлений базисных знаний выбран предусмотренный для анализа и проектирования объектно-ориентированный универсальный язык моделирования UML. Соответственно разработаны механизмы объектно – ориентированного представления знаний с помощью диаграмм UML и методы и алгоритмы перевода их в семантические сети. Представление в виде семантических сетей осуществлено с использованием языка XML. Компонента обучения на базе модели CMM предназначена для обучения основным принципам правильной организации процесса разработки программного обеспечения; компонента тестирования определяет уровень зрелости разработчиков.

Кроме того, предусмотрена компонента, осуществляющая тренаж на основе лучших практик завершенных проектов (проектных прецендентов). Прецедент - это описание проблемы или ситуации в совокупности с подробным указанием действий, предпринимаемых в данной ситуации или для решения данной проблемы. Вывод на основе прецедентов - это метод принятия решений, в котором используются знания о предыдущих ситуациях или случаях (прецедентах). При рассмотрении новой проблемы (текущего случая) находится похожий прецедент в качестве аналога. Можно попытаться использовать его решение, возможно, адаптировав к текущему случаю, вместо того, чтобы искать решение каждый раз сначала. После того, как текущий случай будет обработан, он вносится в базу прецедентов вместе со своим решением для его возможного последующего использования. Прецедент включает:

1. Описание проблемы.
2. Решение этой проблемы.

3. Результат (обоснованность) применения решения.

Описание проблемы должно содержать всю информацию, необходимую для достижения цели вывода. Описание результата может также включать ссылки на другие прецеденты, текстовую информацию (рис. 19) .

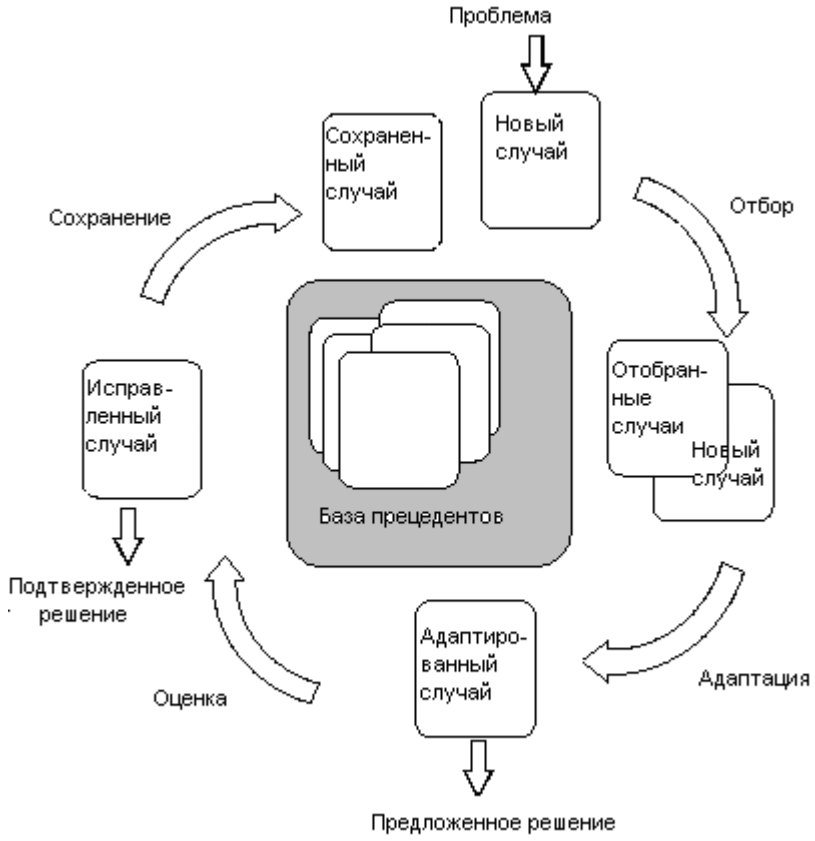


Рис. 19. Выбор решения на основе прецендентов.

Предлагается следующая структура прецедента для адаптивного управления:

1. Состояние до воздействия. Описание объекта (набор признаков, принадлежность к классу состояний).
2. Управляющее воздействие. Описание воздействия (здесь возможна формализация, в частности, классификация управляющих воздействий). Как частный случай, возможно отсутствие воздействия.
3. Состояние после воздействия. Описание объекта (набор признаков, принадлежность к классу состояний).
4. Исход (положительный исход/отрицательный/спорный).

   Наполнение базы прецедентов может происходить как до момента начала управления на основе априорной информации, с помощью

реальных или смоделированных прецедентов, так и в процессе управления, после обработки итога управляющего воздействия.

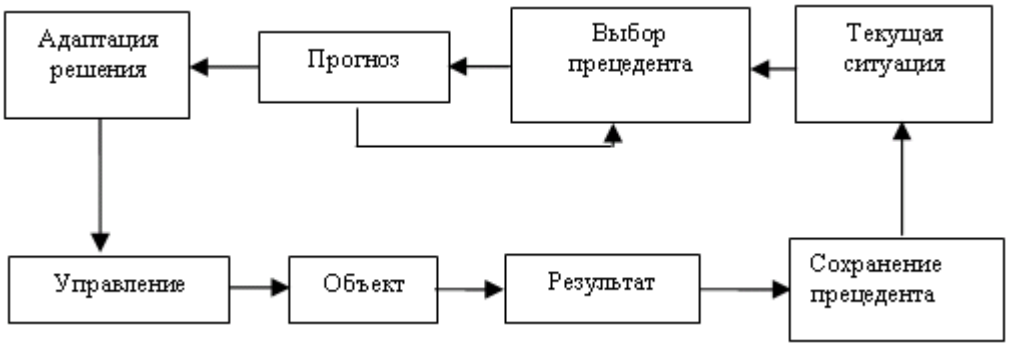


Рис.20. ***Схема адаптивного управления по прецедентам.***

Классификация состояний может производиться с привлечением экспертного знания или путем предварительной кластеризации в этом случае будет выглядеть так (рис. 20). После применения регулирующего воздействия и оценки итога этого воздействия текущая ситуация превращается в прецедент, который заносится в базу прецедентов. Отрицательный результат также является информативным и заносится в базу. Данная модель должна обеспечить решение следующих задач:

1. Формирование обобщенных образов состояний на основе априорной информации (обучение).
2. Идентификация состояния по его выходным параметрам.
3. Определение влияния входных параметров на перевод в различные состояния.

После того, как прецеденты извлечены, нужно выбрать "наиболее подходящий" из них. Это определяется сравнением признаков в текущей ситуации и в выбранных прецедентах. Затем выбирается способ измерения степени близости прецедента и текущего случая по каждому признаку (будь это текстовый, числовой или булевский), который пользователь сочтет полезным для достижения цели. Вводится метрика (расстояние) на пространстве всех признаков, в этом пространстве определяется точка, соответствующая текущему случаю, и в рамках этой метрики находится ближайшая к ней точка из точек, представляющих прецеденты. Каждому признаку назначают вес, учитывающий его относительную ценность. Полностью степень близости прецедента по всем признакам можно вычислить, используя обобщенную формулу типа:

$$\frac{\sum_j w_j * sim(x_{ij}, x_{kj})}{\sum_j w_j}$$

где wj - вес j-го признака, sim - функция подобия (метрика), x_{ij} и x_{kj} - значения признака xj для текущего случая и прецедента, соответственно. После вычисления степеней подобия для всех прецедентов получаем их единый ранжированный список. В управлении важен еще и результат воздействия, то, насколько он приближает к цели. При вводе метрики можно учесть и этот критерий. Считая, что цель должна быть достигнута за конечное число шагов, можно считать более близким прецедент, позволяющий достичь цели за меньшее число шагов. В общем случае, можно представить многоуровневую метрику, где прецеденты сравниваются по:

*1) Состоянию **до** воздействия; 2) Воздействию; 3) Состоянию **после** воздействия.*

При вычислении расстояния каждому прецеденту ставятся в соответствие дескрипторы описания, состоящие из кортежей ({…,Ci,…} типа Ci=<н,з,в,о>, где н – наименование свойства; з – его значение; в – важность или информационный вес свойства; о – ограничение на интервал допустимых значений). Ограничение определяет интервал значений, в рамках которого значение свойства может определять значение меры подобия. Сложность поиска решения и выявления различий между прецедентами в значительной степени зависит от используемых термов индексации. Каждому свойству (или *размерности)* присваивается определенный вес, соответствующий степени "важности" этого свойства. Из базы прецедентов выбирается тот, который "заслужил" самую высокую оценку. Вычисленное значение обычно называется *агрегированной оценкой совпадения.* Если же алгоритм работы системы предполагает и исследование альтернативных прецедентов, то оставшиеся должны быть ранжированы по полученным оценкам. Из всех этих рассуждений вытекает алгоритм сопоставления прецедентов, представленный ниже на псевдокоде:

```
Set:= Ø   {Set – множество близких стереотипов}
FOR Ster:= Ster1 TO SterN DO   {Ster1, …, SterN – все стереотипы в
                                                    базе знаний}
  IF тип цели Ster совпадает с типом текущей цели THEN
    Match1:= частичная строка сопоставления объектов из описания
```

```
                уточняемой  стереотипом цели с реальными
                объектами из текущей цели
    FOR Str:= Str1 TO StrM DO
      {Str1, …, StrM – строки сопоставления из памяти узла Store
      ситуации стереотипа Ster, которых нет в памяти узла End}
      IF Match1 является подстрокой Str THEN
        Node := узел-потомок узла Store
        WHILE тип Node ≠ End DO
          Выполнить проверку в узле Node
          IF проверка неуспешна THEN
            приписать к Str пометку о невыполненном условии
          END IF
          Node := узел-потомок Node
        END WHILE
        построить список недостающих фактов Facts по пометкам Str
        Set:= Set + (Ster, ситуация Ster, Str, Facts)
      END IF
    END FOR
  END IF
END FOR
```

В случае если найденный прецедент не является полным аналогом текущей ситуации, должна выполняться адаптация - модификация решения, которое имеется в выбранном прецеденте и направлено на решение целевой проблемы. Алгоритмы адаптации предполагают наличие зависимости между признаками прецедентов и признаками содержащихся в них решений. Такие зависимости могут задаваться человеком при построении базы прецедентов или обнаруживаться в базе автоматически методами добычи знаний. Процесс модификации решения включает ряд шагов, от простой замены некоторых компонентов в имеющемся решении, корректировки или интерполяции (числовых) признаков или изменения порядка операций до более существенных действий.

4.2. Трансформация UML-моделей

Модельно-ориентированный подход к разработке ПО позволяет решить проблемы, связанные с постоянно увеличивающимся количеством технологических платформ, а так же может ускорить разработку и интеграцию систем. На данный момент разработаны и активно используются множество стандартов, middleware-технологий и платформ (CORBA, DCOM, Dot-Net, WebServices, JAVA-технологии и т.д.).

Существует несколько способов описания и выполнения трансформаций UML-моделей [34]. Простейший способ - это явное императивное описание процесса трансформации с использованием любого алгоритмического языка. При этом подходе в среду разработки на этапе её создания встраивается набор трансформаций, которые позднее могут быть задействованы пользователем. У этого подхода имеются существенные недостатки. Прежде всего, у пользователя отсутствует возможность добавлять новые описания трансформаций или изменять существующие; он вынужден использовать то, что сделано разработчиками инструмента. Кроме того, из-за отсутствия единого стандарта описания трансформаций разные среды разработки неминуемо будут выполнять трансформации по-разному. И наконец, подобный подход означает, что для каждой среды разработки придётся писать полный набор описаний трансформаций для всех популярных технологий, вместо того чтобы использовать стандартные описания трансформаций, созданные независимыми разработчиками.

Другой подход - использование уже разработанных механизмов трансформаций и преобразований из других областей информатики. В частности, можно представить UML-модель в виде графа и использовать математический аппарат трансформации графов. Главный недостаток такого подхода состоит в том, что в нем используется собственный понятийный аппарат, не имеющий отношения к UML-моделированию. Это значит, что от пользователей такой системы требуется знание не только UML-моделирования, но и теории графов и принципов их трансформации. Кроме того, поскольку UML-модель несёт семантическую нагрузку, отличную от формального графа, правила трансформации, сформулированные для графа, будут трудны для понимания с точки зрения UML-модели.

Ещё один вариант заключается в использовании методик трансформации XML-документов и стандарта XMI [35]. XMI (XML Metadata Interchange) - это стандарт, позволяющий представить UML модель в виде XML документа. Он предназначен главным образом для хранения UML-данных, а так же любых других данных, метамодель которых задана с помощью MOF (Meta Object Facility) и обмена ими между различными инструментами и средами разработки. MOF - это стандарт описания метамоделей, с помощью которого в частности можно описать структуру и

общий вид UML-модели. Для XML существует несколько хорошо развитых методик трансформации, в частности XSLT [36] (EXtensible Stylesheet Language) и XQuery [37]. Для трансформации UML-модели можно преобразовать её в XMI-представление и выполнить трансформацию средствами работы с XML (рис. 21).

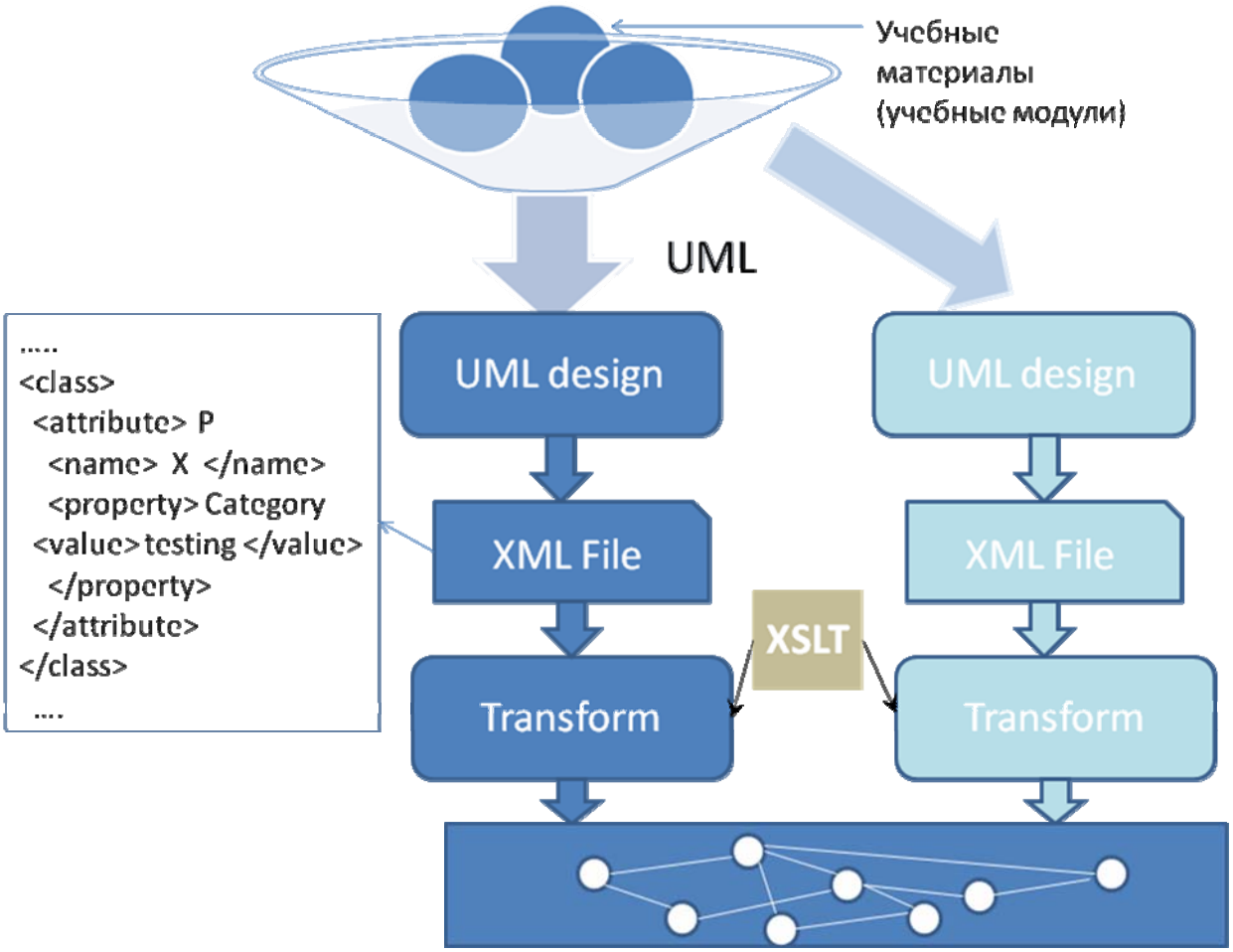


Рис. 21. Схема трансформаций учебных модулей.

Язык моделирования UML считается универсальным, и, конечно же, он содержит средства описания трансформаций. Однако даётся возможность только описывать сам факт того, что между определёнными элементами модели существует отображение, но не содержится развитых средств для задания трансформаций в общем виде (декларативно или императивно). Ещё один стандарт из семейства UML - QVT (Query, View, Transformation) - представляет значительно больший интерес с точки зрения его применения. Но, к сожалению, на данный момент этот стандарт находится на ранних этапах разработки.

Принципиальная схема инструмента трансформации

Инструмент трансформации может быть самостоятельным программным продуктом или компонентом среды разработки.

Входными данными для него являются:

- Одна или несколько *исходных моделей*
- Метамодель для каждой модели, принимающей участие в трансформации.
- Описание трансформации на определённом языке трансформации. Описание трансформации существенно зависит от используемых метамоделей, но по возможности универсально относительно исходных моделей.

Результатом работы инструмента являются:

- Набор исходных моделей с изменениями, внесёнными в процессе трансформации.
- Одна или несколько новых сгенерированных моделей, созданных в процессе трансформации. Наличие и количество таких моделей зависит от используемого описания трансформации. Каждая сгенерированная модель соответствует одной из метамоделей, заданных в качестве исходных данных.
- Информация о связях и отображениях между элементами модели, образованных в процессе трансформации. Такая информация необходима для того, чтобы далее можно было поддерживать соответствие между моделями при их модификации.

Язык описания трансформаций

Описание трансформации представляет собой один или более *блоков трансформации*. Каждый блок имеет уникальное имя и состоит из последовательности *правил трансформации*. В заголовке блока может быть также указан параметр, определяющий порядок выполнения этого блока. Ниже приведены фрагменты синтаксической нотации языка трансформации, заданной с помощью расширенных БНФ (форм Бэкуса-Наура).

```
transformation::= <stage>*;
stage::= stage <name> [<sequence>] { <transformation_rule>* };
sequence::= [reversed] (linear | loop | rollback | rulebyrule);
```

Каждое правило имеет уникальное имя и состоит из секции выборки, определяющей, в каких случаях применимо правило, и секции генерации, в которой задаются действия, совершаемые при применении правила.

```
transformation_rule::= rule <name> {
                         <select_section> <generate_section> };
```

Секция выборки состоит из последовательности *операторов выборки*. Каждый оператор выборки объявляет новую переменную, называемую *переменной выборки*. Имя этой переменной должно быть уникальным в пределах данного правила, а область значений - множество элементов модели, задаваемое с помощью *навигационного выражения*. Кроме того, секция выборки может содержать *уточняющие условия* - логические выражения, в котором могут использоваться переменные выборки, объявленные вышестоящими (в рамках правила) операторами выборки.

```
select_section::= (<select_operator>|<constraint>)*;
select_operator::=forall <name> from <nav_expression>;
constraint::= where <condition>;
```

Навигационное выражение - это последовательность *направлений навигации*, начинающаяся с имени ранее объявленной переменной (назовём её *базовой переменной*). Направление навигации - это имя ассоциации в метамодели UML, соответствующей трансформируемой UML-модели (если в трансформации участвует несколько моделей с разными метамоделями, то метамодель определяется по тому, на элемент какой модели указывает начальная переменная).

```
nav_expression::=<name> iteration_pair(/, <nav_direction>);
```

При вычислении навигационного выражения происходит последовательный переход от одного UML-элемента к другому по ассоциации метамодели, соответствующей очередному направлению навигации, начиная со значения базовой переменной. Формально результат вычисления навигационного выражения можно описать с помощью индукции:

- для выражения, состоящего только из имени переменной, результатом является значение этой переменной;
- процесс вычисления разобьём на шаги: первый шаг - выражение, состоящее только из базовой переменной; на каждом следующем шаге будем добавлять к вычисляемому выражению очередное (слева направо в исходном выражении) направление навигации;

- результатом вычисления очередного шага будет множество элементов модели, достижимых хотя бы из одного элемента, полученного на предыдущем этапе, переходом по текущему (то есть последнему добавленному) направлению навигации;
- если исходное выражение содержит последовательность из N направлений навигации, то на (N+1) шаге будет получен результат вычисления этого выражения.

Секция генерации - это последовательность операторов создания, изменения или удаления элементов модели. Оператор создания элемента позволяет добавить новый элемент модели в последовательность элементов, оператор удаления - исключить элемент из списка. Соответствующие навигационные выражения указывают на изменяемый список и определяют тип добавляемого элемента. Оператор изменения позволяет менять значение того или иного поля данных или атрибута; при этом навигационное выражение указывает на изменяемый элемент.

```
        generate_section::=
(<create_operator> | <update_operator> | <delete_operator> |
 <include_operator> | <exclude_operator> )*;
        create_operator::=make <name> in <nav_expression>; ;
        update_operator::= <nav_expression> = <expression>; ;
        delete_operator::= delete <nav_expression>; ;
include_operator::=include <name> in <nav_expression>; ;
        exclude_operator::= exclude <name> in <nav_expression>; ;
```

Выполнение трансформации

Теперь, когда определён синтаксис языка трансформации, опишем процедуру выполнения трансформации. Инструмент трансформации, получив в качестве входных данных исходную модель (или несколько моделей) и описание трансформации, а также имея информацию об используемой метамодели, может выполнить заданную трансформацию. При этом, в зависимости от описания трансформации, возможно как изменение исходной модели, так и создание новой. Выполнение трансформации заключается в последовательном применении блоков трансформации. Выполнение блока состоит в нахождении в этом блоке правила, которое может быть применено в данный момент, и применении этого правила. Применимость правила определяется по его секции выборки. Каждый оператор выборки объявляет новую переменную и с помощью навигационного выражения определяет множество возможных значений этой

переменной на данной модели. Секция выборки может также содержать уточняющие условия; конъюнкцию всех таких условий назовём обобщённым уточняющим условием. Множество возможных выборок - это декартово произведение множеств значений переменных выборки, ограниченное уточняющим условием, то есть множество всевозможных наборов значений переменных, для которых обобщённое уточняющее условие истинно. Выборкой назовём произвольный элемент этого множества. Правило применимо, если для текущего состояния трансформируемой модели существует такая выборка, для которой это правило не было применено ранее. Применение правила к выборке состоит в последовательном выполнении всех операторов секции генерации. При этом значениями переменных выборки являются соответствующие компоненты выборки, для которой применяется правило. Каждый раз, когда применяется правило, создаётся новый экземпляр трансформационной связи, порождённой этим правилом. Если правило применимо более чем к одной выборке, то для выполнения случайным образом выбирается одна из них. Далее, в зависимости от порядка применения правил в блоке, правило может быть применено к другим выборкам, или может начаться выполнение другого правила.

Порядок применения правил определяется заданной при создании описания трансформации *последовательностью применения правил* в блоке. Она задаётся с помощью параметра <*sequence*> в заголовке блока. Если значением параметра является "*linear*", то блок выполняется линейно сверху вниз, то есть поиск следующего применимого правила осуществляется в порядке объявления правил в описании трансформации, начиная с последнего выполненного правила. Когда процесс доходит до последнего правила в блоке, выполнение блока завершается. При значении параметра "*loop*" процесс выполнения также происходит сверху вниз по описанию трансформации, но после достижения последнего правила выполнение блока не заканчивается, а продолжается с первого правила в описании блока. Если значением параметра является "*rollback*", то после каждого применения правила поиск следующего применимого правила начинается с первого правила в описании блока, а не с последнего применённого правила, как в предыдущих вариантах. При значении параметра "*rulebyrule*" поиск применимого правила также начинается с первого правила в описании блока, но

после нахождения применимого правила оно выполняется для всех возможных значений выборки, и только после того, как все возможности применения этого правила исчерпаны, происходит поиск другого применимого правила (начиная с первого правила в описании блока). Также возможно применение описанных выше параметров совместно с ключевым словом "*reversed*". В этом случае процесс поиска применимого правила происходит не в порядке объявления правил в описании блока, а в обратном порядке. Если ни одно правило из блока трансформации больше не может быть применено, или если достигнут конец описания блока при линейном порядке выполнения, то этот блок трансформации считается завершённым и начинает выполняться следующий. Трансформация считается завершенной, когда выполнен последний блок.

Механизмы уточнений правил и шаблоны

Для улучшения структуры языка трансформации в него добавлено понятие уточнения правил. При описании любого правила можно объявить, что оно *уточняет* одно из ранее объявленных правил в рамках того же блока трансформации. Для этого в заголовке правила после его имени указывается имя уточняемого правила. Возможно создание многоярусных древовидных иерархий за счёт дальнейшего уточнения уточняющих правил и создания нескольких уточняющих правил для одного уточняемого.

```
transformation_rule::= rule <name> [ : <name>] {
                          <select_section> ; <generate_section> };
```

Уточняющее правило применимо в тех случаях, когда выполняются все условия из следующего списка:

- применимо уточняемое правило;
- найдена выборка, удовлетворяющая секции выборки уточняющего правила;
- это уточняющее правило ранее не применялось к данному применению уточняемого правила и данной выборке.

Применение уточняющего правила заключается в выполнении операторов секции генерации. После выполнения секции генерации происходит замена трансформационной связи, порождённой применением уточняемого правила, на расширенную трансформационную связь, включающую как старые переменные, так и переменные, объявленные

уточняющим правилом. *Шаблон* - это специальная разновидность правила, описание которого начинается с ключевого слова "*abstract*". Сам по себе шаблон не применим никогда, вне зависимости от его секции выборки. Но на основе шаблона за счёт механизма уточнения можно создавать новые правила, которые уже могут быть применены при соответствии выборки и трансформируемой модели. Как и при применении обычных уточнений, использование шаблонов позволяет структурировать совокупность трансформационных связей для её последующего использования в других правилах трансформации.

4.3. Метод тестирования возможностей разработчиков в процессе создания программного обеспечения на основе модели CMM.

Назначением когнитивной системы является *обучение* методам коллективной работы, а также *оценка возможностей* коллектива разработчиков – исполнителей, определяя наиболее приоритетные шаги для улучшения возможностей "программирующей организации" [22]. Назначение системы состоит в содействии целям и согласованным оценкам возможностей потенциальных исполнителей разработки ПО в соответствии с методами современной *технологии программирования* и *обучение* их этим методам. Такие оценки должны проводится либо в процессе квалификации предложений о работах, либо в процессе выбора формальных исполнителей, либо в обоих этих случаях. Система предназначена для оценивании возможностей субподрядчиков/исполнителей в области *технологии программирования* (ТП) а также для их обучения, она также может быть значима в оценке возможностей в области разработки программных систем для конкретных коллективов исполнителей. С другой стороны, она может быть использован как помощь для организаций, разрабатывающих ПО, при их внутренней оценке их собственных возможностей в этой области. Так как понимание подходящей практики ТП сейчас только образуется,

стандартных, широко признанных методов измерений в этой области еще не существует.

Методология оценивания основана на принципе, в соответствии с которым предварительный опыт есть лучшая основа для предсказания будущих характеристик. Процесс оценивания фокусируется на определении и прояснении положительных атрибутов хорошего опыта в ТП. Программа обучения предусматривает несколько занятий по обзору оценочных вопросов в деталях и обсуждению материалов и поддерживающих средств, которые доступны для демонстрации характеристик каждого вопроса.

Вопросы для оценивания основаны на следующих предпосылках:

- Качество ПО зависит, по большей части, от качества процесса его создания.
- Процесс разработки ПО есть процесс планируемый, управляемый, измеряемый и постоянно совершенствуемый.
- Качество процесса разработки ПО (равно как и результатов этого процесса) определяющим образом зависит от технологии, используемой для его поддержки.
- Уровень технологии, используемый в конкретной разработке, будет соответствовать уровню зрелости этого процесса.

Чтобы обеспечить структуру для оценивания, рассматриваются пять уровней зрелости модели CMM процесса разработки ПО и две стадии технологического совершенствования.

Стадии Технологии Разработки

A. *Непроизводительная:* Доступны множество реализаций отдельных методов и приемов, и имеющийся опыт может быть широко используемым, но технология не слишком эффективна, наиболее важные методы ТП не практичны в больших, сложных разработках.

B. *Базовая:* Доступны множество реализаций отдельных методов и приемов, и они подтверждают свою эффективность. Организации, исполь-

зующие базовые технологии приемлемо устойчивы в своих характеристиках.

Поместив полученный уровень зрелости процесса разработки организации и оценку ее технологической стадии в двухмерную матрицу, общую оценку можно найти, комбинируя оба измерения. Рисунок 22 представляет уровни процесса по оси x, а технологические стадии - по оси y. Кроме того, на рисунке представлена область, которую можно считать целевой, поскольку именно в эту область будет стремиться перейти организация при совершенствовании своих возможностей.

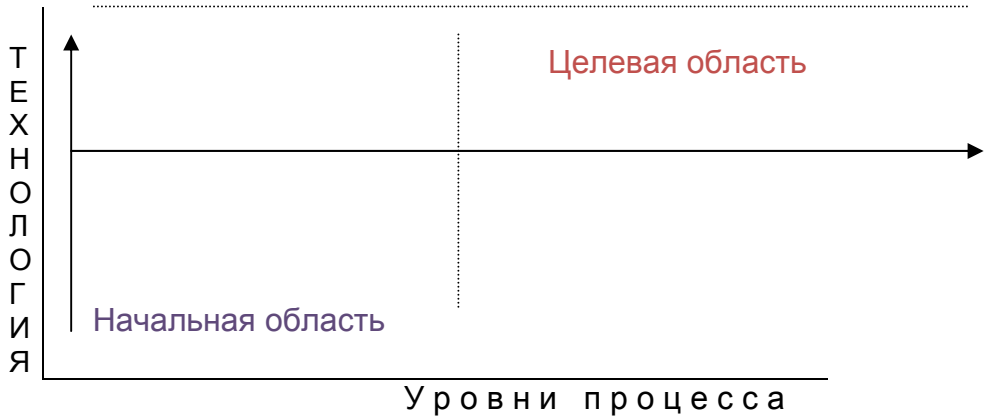


Рис. 22. Матрица Процесс/Технология

Как было замечено ранее, практика технологии программирования является не только сложной, но еще и эволюционирующей.

Уровни Зрелости.

Каждый уровень зрелости (кроме уровня 1) характеризуется ключевыми областями процесса (key process areas, *KPA*). Ключевые области процесса — это области, на которые организация должна обращать особое нимание, чтобы поднять свои процессы до данного уровня зрелости.

Чтобы организация достигла некоторого уровня зрелости, она должна удовлетворять всем KPA на этом уровне зрелости и всем KPA на всех более низких уровнях.

На рис.23 изобрежены **ключевые области** уровней зрелости **CMM**.

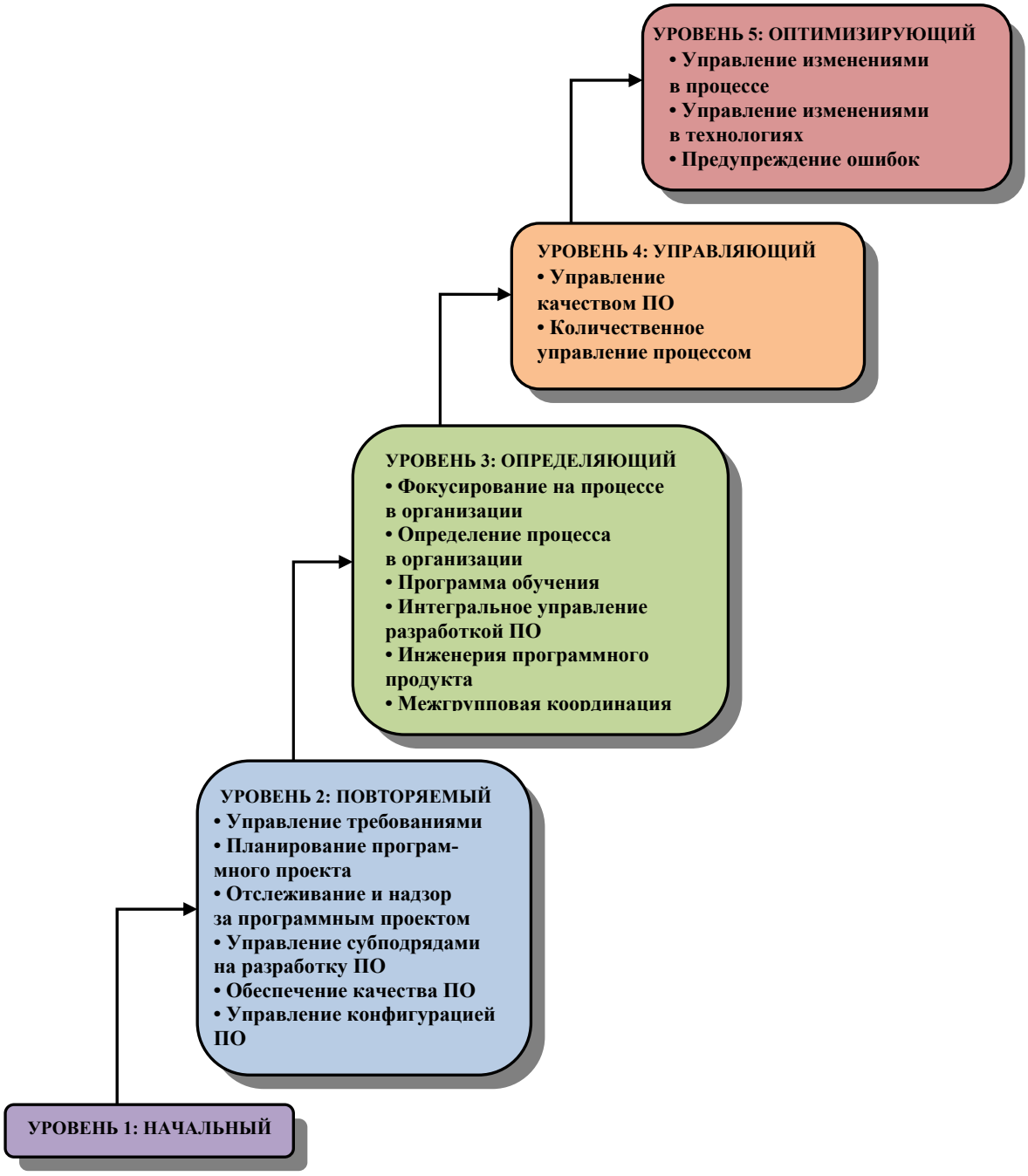


Рис. 23. Ключевые области уровней зрелости CMM.

Каждая KPA задает цели, которым процессы в организации должны соответствовать, чтобы удовлетворять этой области процесса. Кроме того, каждая KPA задает группу действий, называемую ключевыми практиками (*key practices*), которые вместе удовлетворяют целям этой KPA. Во многих отношениях цели KPA фиксируют ее сущность: цели, установленные CMM для процессов этой ключевой области.

В таблице 5 перечислены все цели для КРА уровня 2.

КРА	Цели
Управление требованиями	• Для определения базовой линии разработки ПО и управления разработкой требования к ПО контролируются • В соответствии с требованиями поддерживаются планы разработки ПО, продукты и действия
Планирование программного проекта	• Для использования в процессах планирования и отслеживания проекта оценки документируются • Планируются и документируются действия и обязательства по проекту • Участвующие группы и отдельные лица принимают на себя обязательства, связанные с проектом
Отслеживание и надзор за программным проектом	• Фактические результаты и характеристики сравниваются с планами разработки ПО • Если фактические результаты существенно отклоняются от планов разработки, вносятся поправки, и управление этим процессом не прекращается до его закрытия • Изменения в обязательствах согласовываются с заинтересованными группами и отдельными лицами
Управление субподрядами на разработку ПО	• Главный подрядчик и субподрядчик договариваются о взаимных обязательствах • Главный подрядчик отслеживает фактические результаты деятельности субподрядчика в соответствии с его обязательствами • Главный подрядчик и субподрядчик поддерживают постоянное взаимодействие • Главный подрядчик отслеживает фактическую продуктивность субподрядчика в соответствии с его обязательствами
Обеспечение качества ПО	• Планируются действия, обеспечивающие качество ПО • Проверяется строгое соответствие программных продуктов и действий применяемым стандартам, процедурам и требованиям • Заинтересованные группы и отдельные лица информируются о действиях и результатах, обеспечивающих качество ПО • Вопросы несоответствия, которые не могут быть разрешены внутри проекта, направляются старшим менеджерам
Управление конфигурацией ПО	• Планируется деятельность по управлению конфигурацией ПО • Определяются, контролируются и делаются доступными избранные программные рабочие продукты • Контролируются изменения в определенных программных рабочих продуктах • Заинтересованные группы и отдельные лица информируются о состоянии и содержании базовых линий ПО

Таблица 5. Цели КРА уровня 2 (повторяемый уровень)

В таблице 6 приведены цели трех из семи КРА уровня 3. Другие КРА нацелены на организационные вопросы и управление процессом.

КРА	Цели
Интегральное управление разработкой ПО	• Определенный для проекта процесс разработки ПО является специализированной версией стандартного процесса разработки ПО в организации • Проект планируется и управляется в соответствии с определенным для него процессом разработки ПО
Межгрупповая координация	• Требования заказчика согласовываются всеми заинтересованными группами • Все группы согласовывают взаимные обязательства • Группы определяют, отслеживают и разрешают межгрупповые вопросы
Экспертизы специалистов	• Планируется проведение экспертиз • Определяются и устраняются ошибки в программных рабочих продуктах

Таблица 6. Цели трех КРА уровня 3 (определенный уровень)

На уровне 4 (таблица 7) устойчивость процесса организации понимается в количественном выражении. Устойчивость процесса используется для установки количественных целей проекта. Данные по производительности процесса постоянно собираются и сравниваются со статистическими данными по производительности в завершенных проектах. При значительных отклонениях применяются корректирующие действия, которые позволяют восстановить контроль над проектом.

КРА	Цели
Количественное управление процессом	• Планируются действия по количественному управлению процессом • Производительность процесса, определенного для проекта, контролируется в количественном отношении • Устойчивость стандартного процесса в организации известна в количественном выражении
Управпение качеством ПО	• Планируются действия по управлению качеством ПО в проекте • Определяются измеряемые цели качества программного продукта и их приоритеты • Измеряется фактическое достижение целей качества для программных продуктов и предпринимаются регулирующие действия

Таблица 7. Цели КРА уровня 4 (управляющий уровень)

Три ключевые области процесса на уровне 5 нацелены на улучшение устойчивости процесса. Из них предупреждение ошибок в наибольшей степени затрагивает управление проектом.

Как было сказано, каждая ключевая область описывается в терминах ключевых практик, необходимых для достижения целей этой ключевой области. *Ключевые Практики* описывают, *ЧТО* должно быть сделано, но не обязаны описывать, *КАК* должны достигаться цели ключевой области. Некоторые ключевые области содержат десятки ключевых практик. Работать к таким большим числом практик неудобно. Для удобства все ключевые практики, принадлежащие каждой из ключевых областей, разбиты на типовые группы, имющие общие характерные черты (common feature).

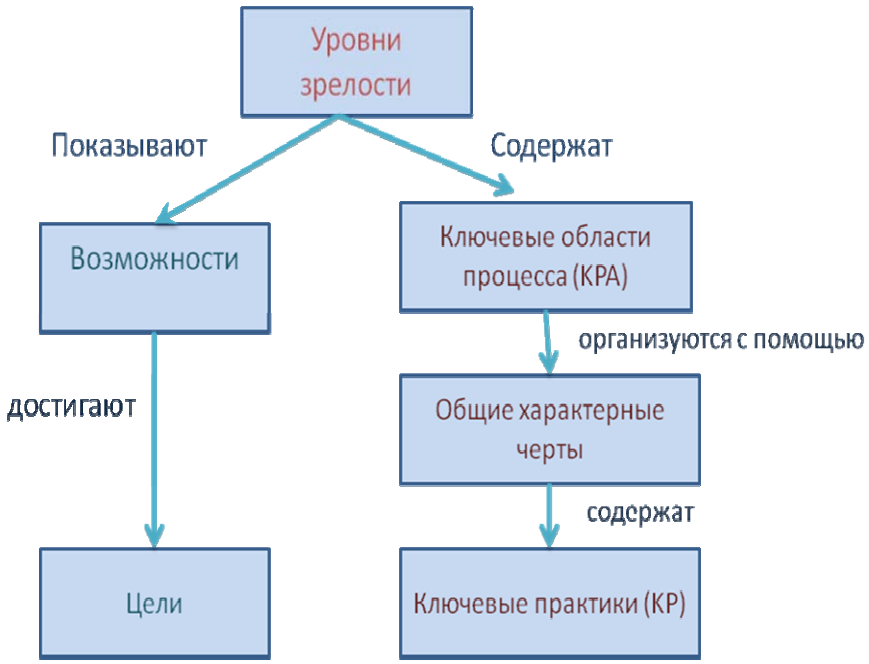


Рис. 24. Структура Модели Зрелости Возможностей

Уровень 1. "Начальный"

Каждый конкретный процесс создания ПО является уникальным, реализуемым во всех своих деталях только в конкретной разработке (следующая разработка, даже если она во многом подобна предыдущей, реализуется по-иному). По фиксированным (утвержденным в организации)

методикам выполняется только небольшое число элементарных действий процесса разработки. В большинстве случаев для совершенствования характеристик своей работы используются технологически-ориентированные средства (например, CASE-среды) или новые методы разработки, но при этом никогда не обсуждается готовность к их применению.

Уровень 2. "Повторяемый"

В организациях, относящихся к уровню 2, отработаны базовые процессы управления каждым проектом в целях отслеживания фактической стоимости, план-графика и получающегося качества результатов разработки. Достигнута необходимая дисциплина процесса, позволяющая устойчиво повторять основные характеристики разработки в новых (схожих) проектах. Для достижения стабильности в работе в организациях уровня 2 обычно добиваются следующего: *прием заказов на ПО реализуется рациональной, формализованной процедурой*; *реализован более строгий и формальный подход к планированию разработки и оценке ее стоимости*; *осуществляется управление конфигурациями*; *определен подход к гарантии качества.*

Организация, находящаяся на уровне 2, использует стандартные методы и практики для управления работами по созданию ПО, такие как оценки стоимости, управление изменениями требований, управление изменениями кода, обзоры состояния разработки и пр. Такая организация обеспечит положительные ответы на большинство ***следующих вопросов:***

1. Существует ли для каждого проекта, включая программные разработки, специально назначенный менеджер?

2. Менеджер программного проекта подотчетен непосредственно менеджеру проекта (проектной разработки), если проект содержит и программную часть, и аппаратную?

3. Имеет ли Служба Гарантии Качества ПО канал административной отчетности, независимый от администрации проекта разработки ПО?

4. Существует ли функция управления конфигурацией ПО для каждого проекта, который включает разработку ПО?

5. Существует ли обязательная программа обучения для всех вновь назначенных менеджеров разработки, созданная специально для ознакомления их с практикой управления программными разработками?

6. Существует ли механизм для сопровождения знаний о состояниях в технологии разработки ПО?

7. Существует ли формальная процедура, используемая в административном обзоре каждой разработки ПО перед принятием решений о контрактных соглашениях?

8. Существует ли формальная процедура для оценки административных обзоров состояния каждого проекта разработки ПО?

9. Существует ли механизм для оценивания (положения дел) соисполнителей (внешних организаций), и, если таковые есть, подчиняются ли они общей (принятой) дисциплине процесса разработки?

10. Существует ли для каждого проекта независимый аудит, проводимый для каждого шага процесса разработки ПО?

11. Применяются ли стандарты на кодирование к каждому проекту разработки ПО?

12. Существует ли формальная процедура, используемая для оценки размера разрабатываемого ПО?

13. Существует ли формальная процедура, используемая при создании планов конкретных работ по созданию ПО?

14. Существует ли формальная процедура, применяемая при оценке стоимости разработки ПО?

15. Существует ли механизм, используемый для того, чтобы получить серьезную гарантию того, что каждая из команд разработчиков верно понимает каждое из требований к разрабатываемому ими ПО?

16. Существует ли соответствие между планируемыми профилями используемого персонала и фактически используемыми?

17. Сопровождаются ли профили размеров ПО для каждого объекта программной конфигурации все время разработки?

18. Оценивается и прослеживается ли использование памяти целевого компьютера?

19. Оценивается и прослеживается ли использование производительности целевого компьютера?

20. Прослеживается ли использование каналов В/В целевого компьютера?

21. Составляются ли отчеты по аварийным завершениям программ по результатам тестирования, прослеживаемым до его завершения?

22. Прослеживается ли прогресс тестирования в сданных программных компонентах и сравнивается ли с плановым?

23. Сопровождаются ли профили содержания версий/редакций по всему времени разработки?

24. Имеет ли высшая администрация механизм для регулярных обзоров статуса проектов разработки ПО?

25. Существует ли механизм, используемый для регулярных технических обменов информацией с заказчиком?

26. Умалчивают ли ведущие менеджеры программных разработок о своих оценках плана и стоимости?

27. Существует ли механизм, используемый для контроля изменений в требованиях к ПО?

28. Существует ли механизм, используемый для управления изменениями кода? (Кто может внести изменение и при каких условиях?)

29. Существует ли механизм для гарантии того, что регрессионное тестирование реализуется в обязательном порядке?

Уровень 3. "Определенный"

Процесс создания ПО как с точки зрения управления им, так и с точки зрения производственной технологии, продокументирован, стандартизован и интегрирован в стандартный (для данной организации) процесс. Все проекты используют утвержденную версию процесса для разработки и сопровождения ПО. Описание процесса образует необходимую платформу для его усовершенствования. При этом условии становятся эффективными эволюционные переходы к передовым технологиям программирования.

Для перехода на уровень 3 необходимо: *ввести формальные стандарты* на всю создаваемую в ходе разработки документацию; *ввести обязательные инспекции* - инспекции становятся механизмом для реализации гарантии качества; *реализовать формальную политику тестирования*; *принять совершенные автоматизированные формы управления конфигурациями*; *управлять качеством начальных работ* (анализ, проектирование); *регулярно проводить учет и аудит* полных конфигураций текущих результатов разработки; *построить формальную модель применяемой технологии программирования*. Организации, претендующие на уровень 3, будут отвечать "да" на большинство ***следующих вопросов:***

1. Существует специально назначенный специалист или команда, ответственный за управление программным интерфейсом?

2. Системная технология ПО представлена команде проектирования систем?

3. Существует ли функция группы процесса ТП?

4. Имеет ли каждый разработчик ПО частный терминал или рабочую станцию?

5. Существует ли требуемая программа обучения ТП для программных разработчиков?

6. Существует ли требуемая программа обучения ТП для ведущих инспекторов (супервизоров) программных разработок?

7. Существует ли формальная программа обучения, требуемая для ведущих обзоров проекта и кода?

8. Существует ли механизм, используемый для оценивания применяемых в организации технологий по отношению к тем, которые общедоступны?

9. Использует ли данная программирующая организация стандартизованный и документированный процесс разработки ПО в каждом из своих проектов?

10. Существует ли стандартизованная документация процесса разработки ПО, описывающая использование средств разработки и тестирования?

11. Существуют ли стандарты, используемые при формировании содержания описывающих ход разработки ПО файлов/журналов?

12. Применяются ли какие-либо стандарты при подготовке тестовых вариантов компонент?

13. Поддерживаются ли стандарты на сопровождаемость кода ПО?

14. Применяются ли стандарты на человеко-машинный интерфейс в каждой из соответствующих программных разработок?

15. Собирается ли статистика по ошибкам в проектах ПО?

16. Прослеживаются ли завершения действий, вытекающих из результатов проектных обзоров?

17. Прослеживаются ли завершения действий, вытекающих из результатов кодовых обзоров?

18. Существует ли механизм, используемый для идентификации и разрешения проблем технологии систем, которые влияют на создаваемое ПО?

19. Существует ли механизм, используемый для проблем независимо вызываемых интеграций и тестирования, привлекших внимание проектного менеджера?

20. Существует ли механизм, используемый для достижения гарантированной согласованности со стандартами ТП?

21. Существует ли механизм, используемый для достижения трассируемости между требованиями к ПО и высокоуровневым проектом?

22. Существует ли механизм, используемый для достижения трассируемости между высокоуровневым и детальным проектами ПО?

23. Проводятся ли внутренние обзоры проекта ПО?

24. Существует ли механизм, используемый для управления изменениями проекта ПО?

25. Существует ли механизм, используемый для достижения трассируемости между детальным проектом ПО и кодом?

26. Поддерживается ли последовательность формальных версий объекта (модуля) по ходу разработки?

27. Проводятся ли обзоры кода ПО?

28. Существует ли механизм, используемый для управления конфигурацией программных средств, используемых в процессе разработки?

29. Существует ли механизм, используемый для верификации того, что образцы, проверенные Службой Гарантии Качества, действительно представляют проделанную работу?

30. Существует ли механизм, используемый для оценки адекватности регрессионного тестирования?

31. Проводятся ли формальные обзоры тестовых вариантов?

Уровень 4. "Управляемый"

Организация уровня 4 имеет внедренной систему мер, охватывающую измерения не только характеристик стоимости и планов/сроков/производительности, но и характеристик качества создаваемого продукта. Вводится программа мер по внедрению системы метрик ПО. Первичными метриками явятся те, которые в том или ином виде позволяют измерить объем проделанной работы (объем кода, прошедшего тестирование, объем согласованных спецификаций и пр.) Организации уровня 4 измеряют характеристики процесса разработки совместно с характеристиками создаваемого ПО (затраты времени на каждый шаг разработки, замеченные дефекты по шагам, на которых их внесли в результаты разработки). В организациях уровня 4 всегда есть группа Гарантии Качества Продукта. В дополнение к вопросам уровней 2 и 3, организация, претендующая на уровень 4, ответит положительно на большее число ***следующих вопросов***:

1. Существует ли механизм, используемый для следует внедрять новые технологические компоненты в процесс разработки?

2. Существует ли механизм, используемый для управления и поддержки внедрения новых технологических компонент?

3. Применяются ли стандарты на внутренние проектные обзоры?

4. Применяются ли стандарты на внутренние обзоры кода?

5. Проектируются и сравниваются ли с фактическими проектные ошибки?

6. Проектируются и сравниваются ли с фактическими ошибки в кодах и тестах?

7. Измеряются и записываются ли покрытия обзоров проекта и кода?

8. Измеряются и записываются ли покрытия тестов на каждой фазе функционального тестирования?

9. Поддерживается ли сопровождаемая и контролируемая база данных о процессе в течение всего процесса?

10. Собираются ли данные обзоров в ходе анализируемых проектных обзоров?

11. Анализируются ли данные об ошибках из обзоров кода и теста для определения вероятного распределения и характеристик ошибок, оставшихся в разработке?

12. Анализируются ли ошибки для определения их причин, связанных с процессом разработки?

13. Анализируется ли эффективность обзоров для каждой разработки?

14. Анализируется ли производительность разработчиков для каждого шага основного процесса?

15. Существует ли механизм, используемый для периодического оценивания процесса разработки и реализации найденных улучшений?

16. Существует ли формальный административный процесс для определения того, является ли прототипирование функций ПО соответствующей частью процесса разработки?

Уровень 5. "Оптимизация"

Здесь управление процесса разработки обеспечивают обратную связь для совершенствования характеристик в любом направлении, так как существует фундамент для этого: внедренная система метрик, охватывающая все необходимые характеристики процесса разработки и результата разработки. Основное внимание руководства сосредоточено на непрерывном текущем совершенствовании технологического процесса, основанном на метриках. Имеется возможность постоянного и целенаправленного улучшения (оптимизации) процесса разработки ПО. Основные ***вопросы*** следующие:

1. Существует ли механизм, используемый для идентификации и замены устаревших технологических компонент?

2. Существует ли механизм, используемый для анализа причин появления ошибок?

3. Рассматриваются ли причины появления ошибок для определения изменений процесса ТП, необходимого для предотвращения именно таких ошибок?

4. Существует ли механизм, используемый для инициализации действий, направленных на предотвращение ошибок?

На данном уровне используются более формальные анализы данных об ошибках и стоимости, собранных во время хода разработок, равно как и введение исчерпывающего анализа причин появления ошибок и исследования способов их предотвращения.

4.4. Описание процессов разработки

Базовые и общие виды деятельности

Модель описывает процессы, которые организация использует для приобретения, поставки, разработки, использования, развития и сопровождения ПО, и виды деятельности, характеризующие степень зрелости этих процессов. Виды деятельности делятся на базовые и общие виды.

Базовые виды деятельности являются обязательными и сгруппированы в пять категорий.

- **Контрактная категория** (Customer – Supplier) состоит из видов деятельности, непосредственно влияющих на заказчика, поддерживают процесс разработки и передачи ПО заказчику и обеспечивают возможность его корректного использования.
- **Инженерная категория** (Engineering) состоит из видов деятельности, которые непосредственно определяют, реализуют и/или поддерживают ПО и документацию для него.
- **Управленческая категория** (Management) состоит из видов деятельности, которые определяют все аспекты управления проектом и используются для координации расхода ресурсов и управления ими, для управления производством ПО или предоставления услуги, удовлетворяющей заказчика.
- **Вспомогательная категория** (Support) состоит из видов деятельности, которые обеспечивают саму возможность эффективной реализации основных процессов проекта и поддерживают их производительность.
- **Организационная категория** (Organization) состоит из видов деятельности, которые определяют цели организации и формируют методы управления, необходимые для повышения качества использования ресурсов и создаваемого ПО. Общие виды деятельности приложимы к

любому процессу и нужны для управления им и улучшения характеристик его выполнения. Общие виды деятельности объединены в группы и уровни зрелости.

Реализация только базовых видов деятельности в проекте может представлять собой только первый шаг в обеспечении зрелости, хотя они и обеспечивают необходимую функциональность для выполнения проекта.

Повышение зрелости процессов выражается в терминах уровней их зрелости и наличия общих видов деятельности. Уровень зрелости обеспечивается набором групп общих видов деятельности, определяющих качество реализации процессов совместно с базовыми видами деятельности. Уровни зрелости – рациональный путь прогрессивного улучшения всего процесса.

Каждому виду деятельности присваивается идентификатор. Для базовых видов этот идентификатор имеет формат **КП.НП.НВ**, где

- КП – категория процесса,
- НП – номер процесса внутри категории,
- НВ – номер вида деятельности внутри категории или уровня.

Для общего вида деятельности идентификатор имеет формат вида: **НУ.НЧ.НВ**, где

- НУ – номер уровня зрелости,
- НЧ – номер группы для определённого уровня,
- НВ – номер вида деятельности внутри категории или уровня.
- **Базовые виды деятельности**

Базовые виды деятельности делятся на пять категорий:

- CUS – Контрактная
- ENG – Инженерная
- MAN – Управленческая
- SUP – Вспомогательная
- ORG – Организационная

Описание каждой из категорий процессов состоит из общей характеристики и кратких аннотаций видов деятельности процессов этой категории, состоящих из сведений о назначении каждого вида, описания характера его взаимодействия с другими видами деятельности и сведений об условиях начала и завершения его выполнения.

- **Контрактная категория процессов (CUS)**

Эта категория состоит из процессов, связанных с заказчиком непосредственно, поддерживающих процесс разработки и передачи ПО заказчику и обеспечивающих возможность его корректного использования.

К данной категории относятся следующие процессы:

- **CUS.1 Приобретение продукта или услуг**

Целью приобретения продукта или услуг является выполнение видов деятельности, осуществляемых заказчиком для получения продукта или услуги. Виды деятельности, относящиеся к данному процессу, включают в себя определение необходимости (и/или целесообразности) получения продукта или услуги, рассмотрение предложений, выбор поставщика и приёмку продукта/услуги.

CUS.1.1 Идентификация потребности. Идентифицировать потребность в приобретении, разработке или совершенствовании ПО. Соотнести с целесообразностью и целями организации.

CUS.1.2 Определение требований. Подготовить требования к системе и ПО, удовлетворяющие установленной потребности.

CUS.1.3 Определение стратегии приобретения. Разработать стратегию приобретения продукта, включающую:

- Анализ рисков покупки/разработки (готовый продукт, самостоятельная разработка, разработка по контракту, совершенствование существующей системы).
- Стратегия приобретения.

CUS.1.4 Подготовка запроса на предложение. Подготовить запрос к тендерным предложениям, включающим требования по приобретению и план-график проекта.

CUS.1.5 Выбор поставщика. Выбрать поставщика продукта/услуги на основе оценки тендерных предложений (с учётом опыта прошлых взаимодействий с известными поставщиками), возможностей и прочих факторов, характерных для проекта.

После выбора поставщика заключается контракт (см. CUS.2). **CUS.2 Заключение контракта.**

Целью процессов этой категории является разработка контракта, точно и однозначно выражающего ожидания и ответственность сторон.

CUS.2.1 Рассмотрение контракта перед подписанием. Рассмотреть контракт перед его подписанием, *что обычно включает:*

- *Оценить смысл контракта и требования в нём.*

- *Оценить возможные риски.*
- *Оценить соответствие контракта стратегии организации.*
- *Оценить защиту конфиденциальной информации.*
- *Оценить требования, отличные от первоначально выставленных.*
- *Оценить способность исполнителя выполнить требования контракта.*
- *Установить ответственность за работу субподрядчиков.*
- *Определить терминологию.*
- *Оценить способность заказчика выполнить обязательства по контракту*.

CUS.2.2 Заключение контракта, который обычно включает: *план поставки продукта; порядок оплаты; критерии готовности продукта к сдаче; процедуры внесения изменений в требования заказчика; процедуры рассмотрения требований заказчика по мониторингу качества работ/продукта; процедуры рассмотрения проблем, выявленных заказчиком; роль заказчика в разработке и сопровождении продукта; ресурсы, предоставляемые заказчиком; определяющие стандарты и руководящие документы; требования по сервису и сопровождению.*

CUS.2.3 Определение интерфейсов с независимыми агентами. Определить отношения заказчика и поставщика к независимой верификации, сертификации и тестированию, и документировать их в контракте.

CUS.2.4 Определение интерфейсов с субподрядчиками. Определить отношения заказчика и поставщика к третьим лицам, таким как субподрядчики.

- **CUS.3 Идентификация требований заказчика.**

Целью идентификации требований заказчика является управление сбором, обработкой и отслеживанием требований и запросов заказчика для лучшего понимания того, что именно его (заказчика) удовлетворит.

CUS.3.1 Получение требований и запросов заказчика. Получить требования и запросы заказчика путём прямого взаимодействия с ним и опроса будущих пользователей, а также анализа бизнес-процессов заказчика и его бизнес-предложения на проектирование и разработку требуемой ему системы и прочих документов, несущих в себе требования заказчика (стандартов его и его предметной области,…)

CUS.3.2 Выяснение ожиданий заказчика. Для уточнения нужд и ожиданий будущих пользователей сделать обзор их требований и запросов.

CUS.3.3 Информирование заказчика о ходе работ. Информировать заказчика о состоянии работ по реализации его требований и запросов.

- **CUS.4 Осуществление совместной оценки деятельности**

Целью этого вида деятельности является обеспечение единого с заказчиком понимания состояния выполнения проекта и того, что должно быть сделано для создания такого ПО, которое удовлетворяет его требованиям.

CUS.4.1 Проведение совместного аудита и контроля. Определить, какие виды и когда (даты или условия/события) будут осуществляться.

CUS.4.2 Подготовка к аудиту и контролю заказчиком. Подготовиться к аудиту и контролю заказчиком выполнения проекта. Для этого надо сделать следующее:

- *Уточнить назначение контроля.*
- *Уточнить список проверяемых работ.*
- *Уточнить желаемый результат проверки.*
- *Установить и согласовать план-график проверок.*
- *Составить список участников проверок от заказчика и от исполнителя.*
- *Установить подходы к выявлению и решению проблем.*
- *Согласовать необходимые условия проведения проверок.*

CUS.4.3 Проведение совместного контроля, осуществляемого руководством организации. Проводить регулярные совместные мероприятия по обеспечению и выполнению контроля, осуществляемого руководством в целях аттестации:

- *соответствия предложений требованиям,*
- *соответствия состояния проекта плану-графику,*
- *выполнения плана-графика,*
- *управления рисками,*
- *соответствия стандартам выполняемых работ,*
- *готовности к следующим стадиям разработки.*

CUS.4.4 Проведение совместного технического контроля. Проводить регулярные совместные мероприятия технического контроля для аттестации технического состояния разрабатываемого ПО и его соответствия документированным требованиям заказчика.

CUS.4.5 Обеспечение приёмки работ заказчиком. Обеспечивать заказчика необходимыми материалами по оценке продукта, своевременно предоставляя свидетельства того, что ПО соответствует стандартам и спецификациям и удовлетворяет критериям приёмки, оговоренным в контракте. *Этот вид*

деятельности может включать в себя тестирование заказчиком разрабатываемого ПО.

- **CUS.5 Поставка и инсталляция разработанного ПО**

Целью этого вида деятельности является оформление, поставка и инсталляция ПО на объекте заказчика и обеспечение его эффективного использования и хранения. Базовые виды деятельности в данной категории процессов важны для поддержания требуемого качества ПО.

CUS.5.1 Формирование требований к инсталляции. Сформировать требования к оформлению, поставке и инсталляции ПО, включая в них по необходимости следующее: *сведения о носителе ПО; перечень сопровождающей документации; данные об авторских правах и лицензировании; порядок хранения основной и запасных копий; условия копирования; наиболее важные требования по безопасности и защите информации.*

CUS.5.2 Подготовка объекта к инсталляции. Подготовить объект заказчика к инсталляции ПО.

CUS.5.3 Оформление ПО. Подготовить ПО и сопровождающую документацию (включая формуляр и список изменений) к отправке заказчику. *ПО создаётся с помощью вида деятельности SUP.2.6.*

CUS.5.4 Поставка ПО. Передать заказчику ПО в составе, определённом в ТЗ на выполнение проекта.

CUS.5.5 Верификация получения заказчиком ПО. Удостовериться в корректном и полном получении заказчиком поставленного ему ПО, включая переданную упаковку, инструкции по поставке и сопровождающую документацию.

CUS.5.6 Инсталляция ПО. Инсталлировать ПО на объекте заказчика, документируя предпринятые шаги и их результаты.

CUS.5.7 Регламентация и выполнение процедур хранения ПО. Определить и обеспечить выполнение процедур дальнейшего использования и хранения ПО и сопровождающей его документации, включая

- *Передачу в архив первых копий кода и документации на случай их утраты.*
- *Восстановление ПО и документации в случае их утраты/некорректной передачи.*
- *Меры безопасности и защиты информации.*

- **CUS.6 Сопровождение функционирования ПО**

Целью этого вида деятельности является обеспечение эффективной эксплуатации ПО в течении обусловленного договором/контрактом срока. При этом подразумевается, что ПО уже было инсталлировано у пользователей (см. CUS.5) и

функционирует как часть общей системы. Также подразумевается, что за рабочее состояние системы отвечает специально назначенный системный оператор, и сама система эксплуатируется персоналом заказчика. Цель управления корректировкой ПО в процессе его эксплуатации определена в ENG.7. Входными данными этого процесса могут быть требования к качеству ПО, определённые видом деятельности ENG.2.1.

CUS.6.1 Определение эксплуатационных рисков. Определять и предотвращать риски, относящиеся к эксплуатации системы, такие как природные явления, отказы аппаратуры или ПО или сети.

CUS.6.2 Операционное тестирование. Осуществить операционное тестирование каждой версии ПО, доказывая его соответствие специфицированным критериям.

CUS.6.3 Эксплуатация ПО. Эксплуатировать ПО в соответствии с его назначением и специфицированным образом.

CUS.6.4 Решение операционных проблем. Выявлять, документировать и разрешать проблемы, возникающие при сопровождении функционирующего ПО (проблемы, возникающие у оператора, а не у пользователей).

CUS.6.5 Реакция на запросы пользователей. Принимать запросы пользователей, относящиеся к ПО, документировать их и помогать в разрешении возникающих у пользователей проблем.

CUS.6.6 Разработка и документирование временных решений. Разрабатывать временные решения в случаях, когда постоянное решение проблемы не может быть предложено своевременно, документировать их и доводить до сведения пользователей.

CUS.6.7 Мониторинг работоспособности системы. Обеспечить возможность регулярного мониторинга работоспособности системы.

- **CUS.7 Поддержка пользователей**

Целью этого вида деятельности является предоставление необходимых (или даже просто желательных) консультаций и помощи пользователям в реальном масштабе времени в процессе эксплуатации ими ПО.

CUS.7.1 Обучение пользователей. Обеспечить пользователей необходимыми им учебными пособиями и провести курс по их обучению с тем, чтобы они смогли эффективно эксплуатировать ПО.

CUS.7.2 Обеспечение сопровождения ПО. Обеспечить условия, при которых пользователь смог бы эффективно выявлять проблемы, возникающие у него в ходе эксплуатации и получать помощь в их разрешении.

CUS.7.3 Мониторинг производительности. Осуществлять мониторинг производительности ПО с целью выявления слабых мест в его функционировании.

CUS.7.4 Инсталляция новых версий. При необходимости планировать, тестировать и инсталлировать новые версии и выпуски ПО.

- **CUS.8 Определение удовлетворённости пользователей**

Целью этого вида деятельности является определение уровня удовлетворённости пользователей полученным ПО и услугами по его сопровождению.

CUS.8.1 Определение уровня удовлетворённости пользователя. Определить этот уровень полученным ПО и услугами по его сопровождению путём анализа данных о результатах его эксплуатации, а при необходимости – путём опросов и исследований.

CUS.8.2 Сравнение с конкурентами. Сравнивать уровень выполнения требований пользователей с получаемым у конкурентов.

CUS.8.3 Публикация сведений о выполнении требований пользователей. Распространить информацию о выполнении требований пользователей в организации.

- **Инженерная категория процессов (ENG)**

Эта категория состоит из процессов, которые непосредственно определяют, реализуют или поддерживают систему и ПО с документацией. Процессы, описанные ниже, могут выполняться последовательно, параллельно или циклически. Входные данные этой категории процессов – контракт или соглашение о характере и составе выполняемой работы и план её выполнения (см. процессы CUS.2 и ENG.2).

К инженерной категории относятся следующие процессы:

- **ENG.1 Определение требований к системе и её проектированию**

Целью этого процесса является определение системных требований к элементам системы и ПО. Этот процесс также должен определить деление системы и ПО на версии.

Этот процесс обычно реализуется специальной группой аналитиков, часть которой должна иметь опыт разработки ПО. Результатом этого процесса должны быть техническое задание и пояснительная записка к нему в соответствии с государственным стандартом или на его основе.

ENG.1.1 Определение системных требований. Составить техническое задание на систему в соответствии с действующими стандартами или на его основе. В CUS.3 приведен перечень требований заказчика, на основе которого проводится анализ требований к системе.

ENG.1.2 Описание архитектуры системы. Описать обобщённую архитектуру системы. Описание должно содержать следующие сведения:

- Описание оборудования.
- Описание ПО.
- Описание состава и назначения ручных операций.

ENG.1.3 Распределение требований. Распределить требования к системе между элементами описанной архитектуры.

Результатом выполнения процессов ENG.1.2 и ENG.1.3 является документированное описание конфигурации системы, определяющее положение каждого элемента в структуре системы, его назначение, характер взаимодействия с другими элементами системы и требования, которые он реализует.

ENG.1.4 Определение стратегии сдачи. Разработать приоритеты реализации требований к системе и определить версии и выпуски системы, в которых они будут реализованы.

В некоторых случаях может оказаться полезным разработку ПО и его предъявление заказчику производить поэтапно. В этом случае надо точно определить, какие из установленных заказчиком требований должны быть реализованы на каждом из этапов разработки и сдачи. Входными данными для этого процесса может служить модель ЖЦ, разработанная процессом MAN.1.

- **ENG.2 Определение требований к ПО**

Целью этого процесса является разработка, анализ и уточнение требований к ПО.

ENG.2.1 Определение требований к ПО. Составить техническое задание на ПО в соответствии с релевантными государственными стандартами.

ENG.2.2 Анализ требований к ПО. Проанализировать корректность требований к ПО. При анализе требований необходимо обращать внимание на: *полноту , понятность, тестируемость, верифицируемость, выполнимость, значимость, согласованность, адекватность.*

В зависимости от выбранного ЖЦ ПО может оказаться желательным реализация только ограниченного согласованного круга требований, оставляя остальные на следующие итерации процесса (см. ENG.2.5).

ENG.2.3 Определение влияния на операционную среду. Определить влияние требований к ПО на операционную среду.

ENG.2.4 Согласование требований с заказчиком. Согласовать техническое задание с заказчиком и по результатам согласования при необходимости скорректировать его. *Может оказаться, что для согласования требований будет*

необходимо разработать прототипы ПО или его модели. Этот вид деятельности может совпадать с CUS.3.2.

ENG.2.5 Корректировка требований для следующей итерации. После завершения итерации проектирования, кодирования и тестирования ПО проанализировать рекламации, замечания и предложения пользователя и заказчика по его модификации и скорректировать на основе них требования.

- **ENG.3 Проектирование ПО**

Целью этого процесса является проектирование ПО в соответствии с требованиями к нему. На верхнем уровне определяются основные компоненты ПО и уточняется состав и назначение элементов ПО (подлежащих кодированию и тестированию) более низкого уровня.

ENG.3.1 Разработка архитектуры ПО. На основе требований к ПО разработать архитектуру ПО, описывающую её структуру на верхнем уровне и определяющую состав и назначение её основных компонент.

ENG.3.2 Определение интерфейсов на верхнем уровне. Разработать и документировать внешние и внутренние интерфейсы верхнего уровня.

ENG.3.3 Разработка детального проекта ПО. На основе архитектуры ПО верхнего уровня разработать детальные описания каждого программного компонента. Эти описания детализируются до самого низкого уровня, на котором они могут быть закодированы, оттранслированы и протестированы. *Подробное проектирование включает в себя разработку спецификаций внешних и внутренних интерфейсов между программными компонентами.* Результатом этого процесса является документ, подробно описывающий положение каждой программной единицы в архитектуре, а также её функциональные, количественные и качественные характеристики.

ENG.3.4 Установление соответствия. Установить соответствие между требованиями к ПО и элементами архитектуры.

- **ENG.4 Реализация проекта ПО**

Целью этого процесса является создание исполняемых, либо встраиваемых и независимо тестируемых компонент ПО, реализующих элементы разработанной архитектуры.

ENG.4.1 Разработка компонент ПО. Разработать и документировать каждый компонент ПО. Информация о каждом компоненте должна включать в себя:

- *код программного компонента,*
- *описание структур данных,*
- *описание баз данных.*

ENG.4.2 Разработка процедуры верификации каждого компонента ПО. Разработать процедуры верификации для каждого компонента на соответствие требованиям проекта.

ENG.4.3 Верификация программных компонент. Верифицировать соответствие каждой компоненты ПО требованиям дизайна и документировать результаты. *Обычно процедура верификации состоит из разработки набора тестов и исходных данных тестирования, проведение самого тестирования и анализа результатов.*

- **ENG.5 Интеграция и тестирование ПО**

Целью этого процесса является интеграция программных компонент в единый комплекс, удовлетворяющий документированным требованиям к ПО.

Процесс состоит в агрегации компонент и тестировании этих агрегатов и в проведении завершающего тестирования интегрированного ПО на соответствие предъявляемым к нему требованиям. *Обычно в тестировании интегрированного ПО его разработчики участия не принимают.*

Разработка планов тестирования должна начинаться заблаговременно, например, одновременно с разработкой требований к ПО и его проектированием. Это необходимо в связи с большой трудоёмкостью подготовки необходимых исходных данных по тестированию.

ENG.5.1 Определение стратегии регрессионных тестов. Определить условия, при которых агрегаты ПО нужно тестировать заново при корректировке отдельных компонент.

ENG.5.2 Формирование агрегатов из компонент ПО. Определить состав агрегатов программных компонент и последовательность их тестирования. *Обычно этот вид деятельности выполняется на основе архитектуры ПО и/или стратегии сдачи ПО.*

ENG.5.3 Разработка тестов для агрегатов. Разработать тесты агрегатов ПО, их данные и критерии соответствия.

ENG.5.4 Тестирование программных агрегатов. Протестировать каждый агрегат в соответствии с разработанной методикой и документировать результаты.

ENG.5.5 Разработка тестов для ПО. Разработать тесты для ПО в целом, включая состав и методики проверок, тестовые данные и критерии соответствия. *Тесты могут быть разработаны в ходе процессов ENG.2, ENG.3 и ENG.4. Разработка тестов должна завершиться до начала интеграции ПО.* Набор тестов должен обеспечить проверку ПО на соответствие требованиям к нему и полностью охватывать его внутреннюю структуру.

ENG.5.6 Тестирование ПО в целом. Протестировать ПО в целом в соответствии с разработанной методикой и документировать результаты.

- **ENG.6 Интеграция и тестирование системы**

Целью интеграции и тестирования системы является интеграция ПО с операциями, выполняемыми вручную, и аппаратурой с тем, чтобы получить систему, удовлетворяющую требованиям заказчика к системе.

ENG.6.1 Построение агрегатов системы. Определить состав и назначение агрегатов элементов системы и последовательность их тестирования. *Обычно этот процесс выполняется на основе архитектуры и/или стратегии сдачи системы.*

ENG.6.2 Разработка тестов для агрегатов. Разработать тесты агрегатов, тестовые данные, методики тестирования и критерии соответствия.

ENG.6.3 Тестирование системных агрегатов. Протестировать каждый агрегат в соответствии с разработанной методикой и документировать результаты.

ENG.6.4 Разработка тестов для системы в целом. Разработать тесты для системы в целом, включая проверяемые требования, тестовые данные, методики проверки и критерии соответствия. *Тесты могут быть разработаны в ходе процесса ENG.1. Разработка тестов должна закончится до начала процессов ENG.6.1..6.3.*

ENG.6.5 Тестирование системы в целом. Протестировать систему в целом в соответствии с разработанной методикой и документировать результаты.

- **ENG.7 Сопровождение системы и ПО**

Целью этого процесса является модификация системы, её аппаратного, программного и сетевого обеспечений и соответствующей документации на основе рекламаций, запросов и пожеланий пользователей, сохраняя при этом её целостность и функциональность.

Причинами необходимости в модификации системы или её ПО могут быть:

- *выявление ошибок,*
- *операционные проблемы с эксплуатацией ПО или системы,*
- *изменения в требованиях и пожеланиях заказчика.*

Этот процесс тесно взаимодействует с несколькими процессами контрактной категории (CUS.6 и CUS.7, например) и может частично перекрываться с ними.

ENG.7.1 Формирование требований по сопровождению ПО. Сформировать требования по сопровождению системы, включая набор сопровождаемых элементов системы и требуемые улучшения.

ENG.7.2 Анализ проблем и пожеланий пользователей. Анализировать проблемы и пожелания пользователей, оценивая возможное влияние различных вариантов модификации системы на ПО и системные интерфейсы.

ENG.7.3 Определение состава изменений для следующей версии. На основе упомянутого анализа определить, какие изменения должны быть произведены при следующем обновлении системы и её ПО, документируя, какие элементы ПО и иные элементы системы (включая документацию) должны быть изменены и какие тесты должны быть проведены.

ENG.7.4 Реализация и тестирование изменений. Выполнить нужные процессы инженерной категории для реализации и тестированию выбранных модификаций, после чего протестировать новую версию (выпуск).

ENG.7.5 Установка новой версии (выпуска) у пользователя. Установить у пользователя новую версию системы и/или ПО. При необходимости обеспечить:

- *параллельное функционирование старой и новой версий,*
- *переподготовку персонала пользователя,*
- *необходимые замены в предыдущей системе.*

- **Управленческая категория процессов (MAN)**

Управленческая категория состоит из процессов, обеспечивающих выполнение проекта и координирующих расход его ресурсов с целью производства продукта или предоставления услуг в соответствии с требованиями заказчика.

Входными данными этой категории процессов является соглашение или контракт (см. CUS.2). Целью процессов этой категории является эффективное использование ресурсов (времени, специалистов, труда, денег) в ходе выполнения проекта.

К управленческой категории относятся следующие процессы:

- **MAN.1 Планирование жизненного цикла (ЖЦ) проекта.**

Целью этого процесса является разработка модели ЖЦ программного продукта. Входными данными этого процесса являются соглашение или контракт на разработку ПО, которые устанавливают цели проекта в зависимости от требований к его качеству, стоимости и/или срокам выпуска.

Результатом выполнения этого процесса является модель ЖЦ ПО с описанием видов деятельности, задач в рамках проекта и методов управления проектом (управленческие и технические ревизии и пр.).

MAN.1.1 Оценка вариантов разработки ПО. Оценить варианты разработки ПО, выявляя риски, связанные с каждым из них.

Примерами вариантов разработки ПО являются:

- *Использование исключительно внутренних ресурсов.*
- *Субподряд.*
- *Использование стандартных программных продуктов.*
- *Использование программных продуктов, предоставленных заказчиком.*
- *Комбинация предыдущих пунктов.*

MAN.1.2 Выбор модели ЖЦ ПО. Выбрать модель ЖЦ, соответствующую рамкам, величине и сложности проекта, а также стратегии организации. *ЖЦ может быть определён заказчиком и задан в контракте.*

MAN.1.3 Описание видов деятельности и задач. Описать каждый из используемых видов деятельности и соответствующие им задачи. Указать назначение каждого элемента описания.

MAN.1.4 Определение структуры задач (работ). Определить структуру задач в пределах выбранного ЖЦ, особо выделяя назначение и периодичность проведения:

- *управленческих и технических ревизий,*
- *аудита ПО,*
- *взаимных проверок текущего состояния проекта.*

MAN.1.5 Документирование видов деятельности. Определить и документировать используемые виды деятельности и задачи по разработке ПО, включая их входы, выходы и взаимодействие друг с другом.

- **MAN.2 Разработка плана проекта.**

Целью этого процесса является создание плана разработки ПО и формирование базы управления проектом. Как правило, в плане отражаются следующие данные: *назначение и цели проекта; что именно должно быть разработано; модель ЖЦ разрабатываемого ПО;смета проекта;риски проекта и планы по их уменьшению или устранению; план-график работ, ресурсы, распределённые по видам деятельности проекта.*

Входные данные этого процесса могут включать в себя описание модели ЖЦ ПО, разработанное процессом MAN.1.

MAN.2.1 Разработка структуры подразделений. Разработать структуру подразделений применительно к составу и последовательности задач, которые

нужно выполнить, и с учётом имеющихся ресурсов, требуемых в процессе выполнения.

MAN.2.2 Определение проектных стандартов. Определить используемые в ходе разработки стандарты.

MAN.2.3 Определение специального оборудования. Определить состав и назначение дополнительных специализированных инструментальных средств и оборудования помимо стандартно используемых, которые будут нужны в процессе выполнения проектных работ.

Такими дополнительными инструментальными средствами могут быть: аппаратура, для которой создаётся ПО, эмуляторы, специализированное тестовое оборудование, тестовые лаборатории, дополнительные средства безопасности.

MAN.2.4 Определить стратегию повторного использования. Проанализировать возможности повторного использования компонент ПО и определить стратегию этого использования.

Этот процесс использует результаты процесса ORG.5.

MAN.2.5 Оценка затрат проекта. Оценить затраты, необходимые для реализации требований к ПО на протяжении всего его ЖЦ. По-моему, это должно быть сделано при заключении контракта.

При оценке должны быть учтены:

- *размер ПО,*
- *трудоёмкость его проектирования и разработки,*
- *сроки проектирования и разработки,*
- *требуемые ресурсы.*

MAN.2.6 Предварительное определение начальных рисков проекта. Определить, оценить и документировать предварительный (исходный) набор рисков проекта и планы по их уменьшению.

Риски программного проекта включают в себя:

- *риск превышения предусмотренных бюджетом размера ПО, стоимости, трудоёмкости и сроков,*
- *риск нехватки ресурсов,*
- *технические риски.*

MAN.2.7 Определение критериев оценки текущего состояния проекта. Определить основной набор критериев оценки текущего состояния проектных работ и степени адекватности результатов их выполнения заданным требованиям.

MAN.2.8 Составление графика проекта. Составить план-график проекта на основании модели ЖЦ, структуры подразделений команды проекта, смет, планов управления рисками и контрольных точек из контракта.

MAN.2.9 Принятие обязательств проекта. Обязательства следовать планам и сметам принимаются всеми задействованными в проекте подразделениями и группами.

MAN.2.10 Документирование планов проекта. Документировать планы выполнения проектных работ. Принято включать в план разработки как составные элементы планы для управления конфигурациями, управления качеством, взаимодействия с сертифицирующим органом, верификации (делается другим подразделением) и управления субподрядчиками.

Сюда же включаются документирование модели ЖЦ ПО.

- **MAN.3 Формирование команды проекта.**

Целью этого процесса является комплектация подразделений, участвующих в выполнении проекта, квалифицированными сотрудниками, способными качественно выполнять свои обязанности. Термин "подразделение" служит для обозначения различных по типу и долговечности групп, включая:

- смешанные подразделения, состоящие из представителей заказчика и поставщиков,
- аналитиков, проектировщиков, кодировщиков, тестировщиков, технологов и пр.,
- подразделения, создаваемые на короткий период для решения определённой задачи, а также долговременные подразделения, являющиеся частью организационной структуры разработчика.

Входными данными для выполнения этого процесса могут служить проектные планы, разработанные в ходе процесса MAN.2.

MAN.3.1 Определение состава ролей выполнения проекта. Определить, выполнение каких ролей потребуется для выполнения работ по проекту, их распределение по подразделениям, выполняемым задачам, права и обязанности.

MAN.3.2 Организация выполнения проектных работ. Обеспечить необходимые условия выполнения проектных работ, включая:

- разъяснение персоналу целей его работы,
- создание атмосферы общности взглядов и общности интересов,
- формирование соответствующих механизмов и представление средств для общения и работы,
- обеспечение управления выполняемыми работами.

MAN.3.3 Поддержка взаимодействия ролей в процессе выполнения проекта. Разработать соглашение по правилам взаимодействия между ролями и обеспечивать их поддержку в процессе выполнения проектных работ.

Соглашения могут быть направлены на

- *распределение ответственности между ролями,*
- *распределение обязанностей между ролями,*
- *правила взаимодействия,*
- *способы взаимодействия,*
- *способы разрешения конфликтов.*

MAN.3.4 Улаживание разногласий между ролями. Выявлять, отслеживать и разрешать разногласия между ролями, которые влияют на ход работы или угрожают срыву выполнения проекта.

- **MAN.4 Управление требованиями.**

Целью этого процесса является формирование базового уровня требований к ПО, которые служат основой для проектных работ, продуктов и видов деятельности, а также управление изменениями в этом базовом уровне.

Основным различием между этим процессом и процессом CUS.3 является то, что в CUS.3 управляемые требования не обязательно ограничиваются этим конкретным проектом или относятся только к ПО. Здесь требования относятся только к этому проекту и только к ПО.

См. также процессы CUS.2 и ENG.2, которые также относятся к управлению требованиями.

MAN.4.1 Согласование требований. Согласовать требования заказчика со всеми ролями, которые участвуют в их реализации.

MAN.4.2 Определение базового уровня требований заказчика. Установить базовый уровень требований заказчика, руководствуясь им в процессе выполнения проекта. Документировать все требования, поступающие от заказчика.

MAN.4.3 Сопровождение требований заказчика. Сопровождать изменения в требованиях заказчика, оценивать их влияние на ход выполнения проекта (и на сами проектные решения), трудоёмкость их реализации и возникающие при этом риски. При необходимости доводить эту информацию до сведения ролей, которые будут участвовать в их реализации.

MAN.4.4 Обработка требований заказчика. На основании требований заказчика выполнять следующие работы:

- Корректировку планов программного проекта,
- Разработку спецификаций требований заказчика,

- Определение необходимого инструментария и видов деятельности для реализации этих требований.

MAN.4.5 Сопровождение инструментария. Определить и поддерживать необходимый состав инструментальных средств на протяжении всего ЖЦ ПО.

- **MAN.5 Управление качеством.**

Целью этого процесса является обеспечение необходимого качества создаваемого ПО и услуг по его сопровождению. Управление качеством включает в себя формирование необходимых критериев качества и его конкретных показателей в соответствии с выбранными критериями, выполнение работ по достижению заданных значений показателей и демонстрацию того, что это качество достигнуто.

Входные данные процесса – требования заказчика и план проекта (см. процесс MAN.2). Необходимый состав работ по управлению качеством должен быть включён в план проектных работ.

MAN.5.1 Определение целей по обеспечению качества. Основываясь на требованиях заказчика, определить цели по достижению требуемого качества ПО для каждого из этапов его ЖЦ.

MAN.5.2 Определение метрик качества. Определить метрики, измеряющие результаты проекта для оценки того, достигнуты или нет соответствующие цели по обеспечению качества.

MAN.5.3 Определение состава работ по обеспечению качества. Для каждой цели по обеспечению качества определить состав работ, выполнение которых необходимо для её достижения, и отразить эти работы в модели ЖЦ ПО.

MAN.5.4 Выполнение работ по обеспечению качества. Выполнять запланированные работы.

MAN.5.5 Оценка качества. По завершению этапов ЖЦ создаваемого ПО оценивать, достигнуты ли соответствующие цели по обеспечению качества в соответствии с заданными метриками качества.

MAN.5.6 Принятие корректирующих мер. В случае, если цели по обеспечению качества не достигнуты, определить, запланировать и выполнить соответствующие меры. *Корректирующие меры могут включать в себя исправление продукта, полученного в результате определённой работы в рамках проекта и/или изменение запланированного состава работ.*

- **MAN.6 Управление рисками.**

Целью этого процесса является постоянное выявление и минимизация рисков в проекте на протяжении всего ЖЦ ПО. Управление рисками включает в себя постоянное выявление новых рисков, работы по их эффективному уменьшению и оценку успешности усилий по минимизации рисков.

MAN.6.1 Определение области управления рисками. Определить область, в которой будет осуществляться управление рисками, включая их серьёзность, вероятность возникновения и типы, которые будут выявляться и минимизироваться.

MAN.6.2 Выявление рисков. Выявлять риски в проекте по мере их возникновения. Риски могут влиять на превышение стоимости проекта, сроков, увеличение трудоёмкости выполнения работ, нехватку ресурсов и пр.

MAN.6.3 Анализ рисков и расстановка их приоритетов. Оценить вероятность возникновения, возможные воздействия, временные рамки, причины и взаимозависимости рисков для определения их приоритетов, в соответствии следует выделять ресурсы для их минимизации.

MAN.6.4 Разработка стратегии минимизации рисков. Определить соответствующие стратегии по минимизации риска или группы рисков.

MAN.6.5 Определение метрик рисков. Для каждого риска или группы рисков определить метрики, отражающие изменения в состоянии риска (вероятность, воздействие, временные рамки) и ход работ по его уменьшению.

MAN.6.6 Минимизация рисков в соответствии с определёнными стратегиями. Минимизировать риски по определённым стратегиям.

MAN.6.7 Оценка результатов минимизации рисков. По завершению этапов проекта оценивать ход и эффективность выполнения стратегии по уменьшению рисков, используя определённые ранее метрики.

MAN.6.8 Принятие корректирующих мер. В случае, если ожидаемый результат не достигнут, определить и принять соответствующие меры. *Корректирующие меры могут включать в себя разработку и реализацию новых стратегий по уменьшению рисков или модификацию существующих.*

- **MAN.7 Управление ресурсами и графиком работ.**

Целью этого процесса является координация ресурсов и управление ими в течении ЖЦ проекта. Входными данными являются планы проекта, разработанные в ходе процесса MAN.2. В ходе данного процесса они могут корректироваться.

MAN.7.1 Распределение ресурсов. Выделять и распределять ресурсы, необходимые для выполнения работ в рамках проекта, включая технические и управленческие ресурсы.

MAN.7.2 Мониторинг хода выполнения проекта. Регулярно проверять соблюдение планов в ходе выполнения проекта.

Внимание должно быть обращено в особенности на следующие стороны хода выполнения проекта: размер, усилия, стоимость, сроки, ресурсы, риски.

MAN.7.3 Проведение проверок. Регулярно и по завершению основных этапов проводить проверки для анализа и оценки состояния:

- соблюдения планов и графика проекта,
- состояния рисков проекта,
- соответствия соответствующим стандартам,
- готовности к очередному этапу разработки.

MAN.7.4 Проведение технических проверок. Регулярно и по завершению основных этапов проводить технические проверки для обсуждения и оценки:

- технического соответствия проектных работ,
- наличия не разрешённых технических вопросов,
- степени технической готовности к очередному этапу разработки.

MAN.7.5 Управление обязательствами. Контролировать выполнение персоналом, участвующим в проекте, смет и планов, принимая при необходимости соответствующие меры.

- **MAN.8 Управление субподрядчиками.**

Целью этого процесса является выбор квалифицированных субподрядчиков и управление их деятельностью.

Базовые виды деятельности помогают добиться того, что поставщик и его субподрядчики имеют одно понимание целей проекта и требований заказчика и того, как они будут совместно работать для успешного достижения целей и реализации всех требований.

MAN.8.1 Постановление о субподряде. Разработать постановление о составе, назначении и порядке проведения работ, которые должны быть выполнены по субподряду.

MAN.8.2 Определение квалификации потенциальных субподрядчиков. Квалифицировать потенциальных субподрядчиков путём оценки их способностей выполнять поручаемые им работы.

MAN.8.3 Выбор субподрядчика. Выбрать наиболее квалифицированного субподрядчика для выполнения возлагаемых на него работ. Вообще, выбор должен быть основан на истории отношений с этим субподрядчиком.

MAN.8.4 Определение обязательств субподрядчика и осуществление контроля за их выполнением. Определить и контролировать выполнение обязательств субподрядчика и обязательств перед ним.

MAN.8.5 Поддержка взаимодействия с субподрядчиком. Для достижения общего понимания выполняемой работы регулярно обмениваться с субподрядчиком технической информацией

MAN.8.6 Оценка соответствия. Оценивать, насколько субподрядчик следует предусмотренным договором стандартам и процедурам.

MAN.8.7 Оценка качества работ, исполняемых субподрядчиком. Оценивать качество работ, продуктов и услуг, производимых субподрядчиком, для обеспечения полноты, корректности и соответствия стандартам и спецификациям.

- **Вспомогательная категория процессов (SUP)**

Вспомогательная категория процессов состоит из процессов, используемых исключительно в рамках других процессов, включая и иные вспомогательные процессы. Они могут использоваться на протяжении всего ЖЦ во всех организациях, участвующих в выполнения проекта по созданию или сопровождению ПО. Для выполнения вспомогательного процесса в рамках другого процесса может потребоваться некоторая его адаптация. Например, особенности конфигурационного управления зависят от используемого инструментария, стабильности результатов выполнения процессов и технологии управления проектом.

При аттестации конкретного вспомогательного процесса следует уделить внимание тому, насколько широко он применяется применительно к уровню зрелости. От эксперта по оценке процессов такого типа потребуется заключение о том, насколько формально и строго реализация такого процесса соответствует требованиям конкретной ситуации.

К вспомогательной категории относятся следующие процессы:

- **SUP.1 Разработка документации**

Целью этого процесса является разработка состава, содержания и структуры документации, необходимой менеджерам, специалистам, пользователям и заказчикам системы или ПО.

Этот процесс охватывает разработку таких категорий документов, как:

полноту; документацию по управлению проектом (например, планы), конструкторскую документацию (например, обоснование технического проекта), входную и выходную документацию процессов (например, документы взаимных проверок), документацию конечного пользователя.

SUP.1.1 Определение требований к документации. Определить требования к документации, включая: требования к титульным листам*;* аудиторию, для которой документация предназначена; назначение; цели документа; обзор содержания; требования к носителю и требования распространения; требования к безопасности. График разработки проектной документации должен быть разработан и включён в планы проекта.

SUP.1.2 Разработка документов. Разработать нужные документы в соответствии с требований к ним.

SUP.1.3 Проверка документов. Проверить разработанные документы на соответствие требований с привлечением, по необходимости, тех, для кого она предназначается (экспертов, в предметной области, экспертов по документации, …). Выверить документы.

Степень о необходимости этого вида деятельности варьируется в зависимости от назначения документов. В случае документации пользователя этот вид деятельности особенно важен (адекватное и актуальное описание интерфейса).

SUP.1.4 Распространение документов. Оформить и передать документ в соответствующем формате соответствующим группам. Определить способ доступа к каждому документу.

SUP.1.5 Сопровождение документации. Поддерживать документы в актуальном состоянии, внося в них текущие изменения.

Если документ является частью базовой версии ПО, он должен модифицироваться и распространяться в соответствии с процессом SUP.2. Если документ является частью базовой версии продукта, подлежащей поддержке, эта поддержка выполняется процессом MAN.7.

- **SUP.2 Конфигурационное управление**

Целью этого процесса является поддержка целостности ПО проекта на протяжении всего ЖЦ.

SUP.2.1 Создание библиотечной системы конфигурационного управления. Создать и поддерживать репозиторий с разграничением доступа, который представляет:

- складирование и восстановление элементов конфигурации (версий и выпусков),
- распределение элементов конфигурации между участниками проекта,
- восстановление архивных версий элементов конфигурации,
- формирование версий и выпусков на основе данных репозитория.

SUP.2.2 Идентификация элементов конфигурации. Идентифицировать каждый элемент, подлежащий конфигурационному управлению.

Элементами конфигурации могут быть, например:

- технические требования, технический проект, код ПО, тесты,
- другие элементы ПО (например, документация пользователя),
- планы проекта,
- стандарты и процедуры,

- переписка с заказчиком.

SUP.2.3 Ведение описаний элементов конфигурации. Разработать и поддерживать описания всех элементов конфигурации, включая:

- их разбиение на компоненты конфигурации более низкого уровня,
- указание ответственных за каждый элемент,
- условия приёмки под конфигурационное управление каждого элемента.

SUP.2.4 Управление запросами на изменение. Документировать, анализировать, утверждать и сопровождать все запросы на изменение и сообщения о проблемных ситуациях для всех элементов конфигурационного управления.

SUP.2.5 Контроль изменений. Обеспечить разграничение доступа в целях обеспечения корректности и целостности программных элементов в репозитории конфигурационного управления.

SUP.2.6 Сборка версий и выпусков ПО. Собирать только санкционированные версии и релизы ПО и только из элементов конфигурации, хранящихся в репозитории.

SUP.2.7 Ведение истории изменений элементов конфигурации. Вести историю изменений каждого элемента конфигурации, записывая действия конфигурационного управления по отношению к элементу с подробностью, необходимой для восстановления предыдущих версий.

SUP.2.8 Формирование отчётов о состоянии конфигурации. Формировать регулярные отчёты о результатах выполнения описанной выше деятельности и о текущем состоянии каждого конфигурационного элемента.

- **SUP.3 Обеспечение качества**

Целью этого процесса является обеспечение соответствия промежуточных и конечных результатов выполнения проекта установленным стандартам, процедурам и требованиям. Ключевым требованием здесь является формирование объективной картины состояния качества выполнения процессов и их результатов. Обеспечение качества может быть реализовано различными способами.

Похожие функции выполняются в процессе MAN.5. Однако в этом процессе акцент делается на выяснении того, что необходимо сделать, чтобы создавать качественное ПО, и как организовать контроль за выполнением этого требования.

SUP.3.1 Определение проектных стандартов. Участвовать в создании планов проекта, оценивая полноту планов и помогая выбирать стандарты и методы, используемые в проекте.

SUP.3.2 Проверка деятельности по созданию ПО. Проверять каждую деятельность по созданию ПО на соответствие планам, стандартам и методам.

SUP.3.3 Ревизии промежуточных результатов выполнения проекта. Проводить ревизии промежуточных результатов проекта на предмет выяснения их соответствия принятым стандартам и правилам.

SUP.3.4 Разработка отчётности о результатах выполнения проекта. Отчитываться о результатах описанной выше деятельности перед сотрудниками и управляющими звеньями.

SUP.3.5 Устранение отклонений. Отклонения анализируются соответствующим звеном управления и, при необходимости, верхним звеном и принимаются меры по их устранению.

- **SUP.4 Разрешение проблем**

Целью этого процесса является обнаружение проблем, их анализ, выявление тенденций их возникновения и их устранение.

Проблемы, которые здесь рассматриваются, это любые обнаруженные проблемы вне зависимости от их источника и происхождения. Источниками проблем могут быть следующие процессы:

CUS.4.3…4.5 (проблемы, выявленные в ходе проверок заказчиком), CUS.6.4 (проблемы, связанные с функционированием ПО), CUS.6.5 (проблемы, связанные с использованием программ), CUS.7.3 (проблемы, выявленные при мониторинге производительности), ENG.5.4, 5.6 (проблемы, выявленные в ходе тестирования ПО),ENG.6.3, 6.5 (проблемы, выявленные в ходе тестирования системы),ENG.7.2 (проблемы пользователей, выявленные в ходе сопровождения),SUP.3.5 (отклонения от требований, стандартов или методов, SUP.5.7 (проблемы, выявленные в ходе взаимных проверок).

SUP.4.1 Подготовка отчёта о проблеме. Подготовить отчёт с описанием происхождения проблемы сразу после её обнаружения.

Эта работа может быть сделана пользователем или заказчиком.

SUP.4.2 Контроль хода разрешения проблемы. Контролировать ход разрешения проблемы по отчёту об изменениях.

SUP.4.3 Расстановка приоритетов проблем. Устанавливать приоритеты разрешения проблем.

SUP.4.4 Определить решение. Проанализировать проблему, определить и документировать её решение

SUP.4.5 Исправление дефекта. Устранить дефект в ПО.

SUP.4.6 Тиражирование исправлений. Тиражировать исправленное ПО.

- **SUP.5 Проведение взаимных проверок**

Целью этого процесса является эффективный поиск и устранение дефектов из промежуточных и конечных результатов проектных работ.

Если в проверках участвует заказчик или руководство проектом (обычно в конце работ), они называются «техническими проверками». Для наиболее трудоёмких промежуточных результатов важно проводить проверки на ранних стадиях их создания, проверяя при этом их техническую корректность. Для таких проверок привлекают "чужих" разработчиков, называя эти проверки "взаимными". Выявленные в ходе таких проверок дефекты и рекомендации являются входными данными к процессу SUP.4.

SUP.5.1 Выбор рабочих продуктов. Определить промежуточные результаты выполнения проекта, которые должны подлежать взаимным проверкам. Это – один из аспектов планирования взаимных проверок, который относится к общей деятельности.

SUP.5.2 Определение стандартов для проверок. Определить стандарты (включая список контрольных вопросов), которые должны использоваться для взаимных проверок.

SUP.5.3 Определение критериев успешного завершения проверки. Определить эти критерии.

SUP.5.5 Распространение материалов проверки. Регулярно распространять материалы проверок по мере их проведения.

SUP.5.6 Выполнение взаимной проверки. Выполнять взаимные проверки для установления качества промежуточных результатов проекта.

SUP.5.7 Документирование выявленных дефектов. Документировать необходимые доработки, выявленные в результате взаимных проверок.

SUP.5.8 Контроль хода доработок. Контролировать ход доработок, выявленных в процессе взаимных проверок.

- **Организационная категория процессов (ORG)**

Эта категория состоит из процессов, предназначенных для определения целей функционирования организации и создания активов процессов, продуктов и ресурсов, которые, будучи использованными в проектах, обеспечивают их выполнение.

К категории организационных процессов принадлежат:

- **ORG.1 Проектирование технологии разработок.**

Целью этого процесса является формирование у сотрудников организации и участников проектов единого видения целей и способов их достижения и корпоративной культуры, которые способствовали бы их эффективному функционированию. Целью реорганизации технологии организации является не только

реорганизация процессов создания ПО. Выполнение этого процесса благотворно влияет на все аспекты деятельности организации.

ORG.1.1 Формирование видения стратегии. Сформировать видение стратегии, определяющей в организации особенности и культуру производства ПО.

ORG.1.2 Распространение видения стратегии. Объяснять видение стратегии организации всем её сотрудникам.

ORG.1.3 Определение корпоративной культуры. Разработать принципы корпоративной культуры, обеспечивающей необходимое качество создаваемого ПО в соответствии с требованиями заказчика.

ORG.1.4 Создание комплексных подразделений. Сформировать подразделения с учётом перспективы развития выпускаемой продукции, способные создавать ПО требуемого качества.

ORG.1.5 Стимулирование. Стимулировать коллективную работу для достижения общих целей персонала, занятого в выполнении проектных работ. *Стимулы могут включать:*

- *установление коллективной ответственности за производительность выполнения работ, начиная с постановки задач и заканчивая реализацией решений,*
- *учёт мнения коллектива при оценке производительности,*
- *вознаграждение персонала за производительный труд.*

ORG.1.6 Планирование профессионального роста. Разработать планы профессионального роста сотрудников организации. В случае небольшой организации с ограниченным потенциалом роста допускаются планы индивидуальной учёбы работников.

- **ORG.2 Определение процессов**

Целью этого процесса является создание многократно используемой библиотеки элементов процессов (включая стандарты, методы и модели) в целях поддержки стабильного и производительного выполнения всех процессов, описанных в этом документе.

ORG.2.1 Определение целей. Определить цели выполнения процессов. Входные данные этого процесса содержат видение стратегии организации. При определении целей необходимо формулировать их так, чтобы их достижение можно было бы определить с достаточной долей объективности (см. ORG.2.9).

ORG.2.2 Выявление текущих видов деятельности, ролей и ответственности. Выявить виды деятельности, составляющие процесс в виде, в котором он выполняется или должен выполняться. Выявить роли и распределение ответственности в рамках этих видов деятельности.

ORG.2.3 Выявление входных и выходных данных процессов. Выявить эти данные.

ORG.2.4 Определение критериев качества входных и выходных данных процессов. Определить эти критерии качества.

ORG.2.5 Определение контрольных точек процессов. Определить контрольные точки процессов, в которых должны проводиться основные проверки и приниматься решения.

ORG.2.6 Выявление внешних интерфейсов. Выявить интерфейсы со связанными процессами.

Базовый вид деятельности ORG.2.3 выявляет промежуточные результаты, являющиеся входными и выходными данными процессов. Этот вид деятельности выявляет связь с другими процессами, с которыми этот процесс может взаимодействовать (например, процессы управления, процессы заказчика/потребителя) и входными/выходными данными, которыми они обмениваются между собой.

ORG.2.7 Выявление внутренних интерфейсов. Выявить интерфейсы между видами деятельности каждого процесса.

ORG.2.8 Разработка состава и назначения записей качества. Разработать состав и назначение записей о качестве промежуточных результатов и методику их ведения.

ORG.2.9 Определение метрик процесса. Определить метрики процесса, необходимые для фиксации его успешного завершения.

В состав метрик должны быть включены метрики эффективности и качества выполнения процесса.

ORG.2.10 Документирование стандартного процесса. Документировать стандарты, методы и модели выполнения процесса, а также состав и структуру его выходных данных.

Стандарты могут распространяться на:

спецификации требований; методы проектирования; стиль кодирования; языки программирования; тестирование, безопасность; человеческие факторы; планы по управлению проектом; планы обеспечения качества; планы управления конфигурациями.

ORG.2.11 Формирование политики. Сформировать и документировать корпоративную политику по выполнению процесса.

ORG.2.12 Определение ожидаемой производительности. Определить ожидаемую производительность процесса при применении разработанных

стандартов предприятия. *Вся ожидаемая производительность обычно выражается количественно.*

ORG.2.13 Развёртывание процесса. Сделать семейство корпоративных стандартных процессов доступным для использования повсеместно внутри организации. Развёртывание корпоративных стандартных процессов включает в себя и обучение персонала.

- **ORG.3 Модернизация процессов**

Целью этого процесса является постоянное наращивание эффективности и производительности процессов, используемых в организации в соответствии с принятой технологией.

Этот процесс обычно сосредоточен на семействе корпоративных стандартных процессов, построенных в соответствии с общим видом деятельности. Именно поэтому данный процесс рассматривается как организационный: накопленные организацией активы стандартов и процессов являются основой для постоянного совершенствования технологии.

ORG.3.1 Выявление возможностей совершенствования. *Выявить возможности для совершенствования процессов, регулярно анализируя:*

- *результаты измерения качества и производительности ПО, возможные изменения процессов и технологий,*
- *результаты сравнения внутренних и внешних данных,*
- *изменения контрактных требований и требований к ПО.*

ORG.3.2 Определение рамок деятельности по совершенствованию. Определить назначение, цели, рамки и приоритеты совершенствования процессов в соответствии с целями организации.

ORG.3.3 Аттестация процессов. Аттестовать процессы, чтобы понять их сильные и слабые стороны.

ORG.3.4 Выявление направлений усовершенствований. Выявлять места, где требуются совершенствование процессов для того, чтобы они эффективнее достигали своих целей. Сведения о целях процессов формируются в процессе ORG.2.1.

ORG.3.5 Расстановка приоритетов усовершенствований. Расставить приоритеты потенциальных усовершенствований процессов, основываясь на анализе их влияния на достижение целей процесса.

ORG.3.6 Определение метрик результатов. Определять метрики, которые могут быть использованы для определения влияния изменений в процессе на достижение им своих целей.

ORG.3.7 Изменение процесса. Изменить процесс с целью его совершенствования.

ORG.3.8 Подтверждение усовершенствования. На основе анализа соответствующих данных провести предварительное тестирование изменений процесса, чтобы подтвердить, что они действительно улучшают процесс.

ORG.3.9 Развёртывание усовершенствования. Сделать усовершенствованные процессы доступными для использования внутри организации там, где это необходимо или полезно.

- **ORG.4 Обучение**

Целью этого процесса является обеспечение организации и проектов сотрудниками, обладающими необходимыми знаниями и навыками для эффективного выполнения своих функций.

ORG.4.1 Выявление потребности в обучении. Выявлять общие потребности в обучении для повышения профессионального уровня знаний и умений у персонала.

ORG.4.2 Разработка или заказ нужных курсов обучения. Разработать или заказать нужные курсы обучения, направленные на удовлетворение общих потребностей в обучении.

ORG.4.3 Обучение персонала. Проводить обучение персонала в соответствии с нужными учебными курсами.

ORG.4.4 Ведение записей об обучении. Вести соответствующие записи о прохождении обучения персоналом.

- **ORG.5 Обеспечение возможности повторного использования**

Целью этого процесса является максимизация повторного использования системных и программных компонент для уменьшения стоимости (и сроков) разработки и сопровождения и повышения качества создаваемого ПО.

ORG.5.1 Определение корпоративной стратегии повторного использования. Определить состав повторно используемых компонент. Когда архитектура создаваемого ПО практически неизменна от проекта к проекту, возможно повторное использование высокоуровневых компонент.

ORG.5.2 Выявление повторно используемых компонент. Выявить программные и системные компоненты, которые могут быть использованы при разработке другого ПО. Для повторного использования подходят только высококачественные компоненты.

ORG.5.3 Разработка повторно используемых компонент. Разрабатывать системные и программные компоненты, предназначенные для повторного использования.

ORG.5.4 Создание библиотеки компонент. Создавать библиотеку повторно используемых компонент, содержащую данные для их идентификации и извлечения.

ORG.5.5 Сертификация повторно используемых компонент. Сертифицировать компоненты из библиотеки на их пригодность к повторному использованию.

ORG.5.6 Интеграция повторно используемых компонент в ЖЦ. Модифицировать ЖЦ ПО и/или стандартные процессы в целях включения в них необходимой поддержки повторно используемых компонент.

ORG.5.7 Публикация изменений в повторно используемых компонентах. Перед тем, как делать глобальные изменения в библиотеке повторно используемых компонент, оценить влияние этих изменений на системы и программы, где эти компоненты уже используются.

ORG.6 Создание условий для производства ПО. Целью этого процесса является создание необходимой инфраструктуры в соответствии с составом и особенностями принятых в организации стандартных процессов.

ORG.6.1 Определение требований к условиям производства ПО. Определить требования к условиям производства ПО. Эти требования должны включать сведения о:

- *составе и назначении ролей и видов деятельности для каждого из процессов,*
- *обеспечении безопасности,*
- *требованиях к пропускной способности общих серверов и разделению данных,*
- *резервировании и восстановлении данных*

ORG.6.2 Обеспечение условий для производства ПО. Обеспечивать необходимые условия для производства ПО, удовлетворяющие предъявляемым требованиям.

ORG.6.3 Предоставление поддержки разработчикам. Предоставлять поддержку разработчикам в процессе выполнения проектных работ. Поддержка включает в себя выявление и разрешение проблем, возникающих при использовании предоставленных средств и оборудования.

ORG.6.4 Обеспечение необходимых условий для производства ПО. Обеспечивать необходимые условия для производства ПО с целью:

- исправления дефектов,
- увеличения производительности,

- модификации условий в соответствии с изменениями в деятельности в рамках процессов и средств, которые их поддерживают,
- управление изменениями для обеспечения при нужде возврата к предыдущему состоянию

➢ **ORG.7 Предоставление рабочего оборудования**

Целью этого процесса является предоставление безопасной и надёжной среды, в которой смогут выполняться проектные работы.

ORG.7.1 Предоставление эффективных рабочих мест. Предоставить персоналу, занятому в проекте, рабочие места, оснащённые всем необходимым для эффективной работы.

ORG.7.2 Обеспечение целостности данных. Предоставить средства, обеспечивающие соответствующее архивирование и защиту от повреждений данных, являющихся результатом деятельности в рамках программного проекта.

ORG.7.3 Обеспечение резервирования данных. Обеспечить регулярное резервирование и архивирование данных, формируемых в процессе выполнения проекта, для предотвращения их потери.

ORG.7.4 Предоставление помещений. Предоставить помещения, включая:

ORG.7.5 Предоставление средств удалённого доступа. Предоставить техническому и управленческому персоналу программного проекта средства доступа к рабочей среде и данных извне, необходимыми для выполнения проектных работ.

Характеристики процессов

Повышение зрелости помогает организации повышать степень предсказуемости результатов выполнения проектов. Проекты в организациях, имеющих 1 уровень зрелости, обычно имеют значительные отклонения от запланированных стоимости, сроков, функциональности и качества.

1. С ростом зрелости отклонения реальных результатов от планируемых уменьшаются.
2. С ростом зрелости улучшаются сами запланированные результаты (себестоимость, время выполнения, производительность и качество).

4.5. Элементы базы данных

В таблице 8 даны общие сведения о проекте, включающие даты начала и конца (предполагавшиеся и фактические), предполагавшуюся трудоемкость (фактическая трудоемкость не помещена в эту таблицу, поскольку она может

быть вычислена из таблицы трудоемкости), максимальный размер команды, информацию о риске, использованные инструменты и другие данные. В таблице может хранится и иная информация, например о клиенте.

Общие характеристики	
Наименование поля	**Значение для проекта**
ProcessCategory (категория процесса)	Разработка
LifeCycle (жизненный цикл)	Полный
BusinessDomain (предметная область)	Маклерство/финансы
ProcessTailoringNotes (замечания по адаптации процесса)	Дополнительная групповая экспертиза документов, обладающих особо сильным влиянием. Для первой программы каждого разработчика проводилась групповая экспертиза
PeakTeamSize (максимальная численность команды)	12
Toolsllsed (используемые инструменты)	MS Visual Studio, MS SQL Server
EstimatedStart (предполагаемое начало)	20 января 2008
EstimatedFinish (предполагаемое окончание)	5 июня 2008
EstimatedEffortHrs (предполагаемая трудоемкость)	3106
EstimationNotes (замечания по оцениванию)	Одним из методов оценки был подход с использованием точек вариантов использования
ActualStart (фактическое начало)	20 января 2008
ActualFinish (фактическое окончание)	5 июня 2008
First Risk (первый риск)	Работа с удаленной базой данных заказчика
Second Risk (второй риск)	Дополнительные требования
Third Risk (третий риск)	Сокращение численности
RiskNotes (замечания по рискам)	Чередование работ; договоренность о модернизации после приемки этого продукта;

Таблица 8. Общие данные о проекте

В таблице 6 зафиксирована информация о трудоемкости. Для разных стадий процесса она включает данные о трудоемкости действия и трудоемкости доработки после выполнения задачи. Объемы доработки фиксировались, поскольку это помогает подсчитать и понять стоимость качества. В таблице 9 показана трудоемкость работ для проекта в человеко-часах. Также представлены оценки трудоемкости.

Трудоемкость по стадиям			
Стадия	Выполнение задачи	Экспертиза	Оценка
Анализ требований	0	0	0
Проектирование	414	32	367
Программирование	1147	76	1182
Независимое тестирование элементов	156	74	269
Комплексное тестирование	251	30	180
Приемочные испытания и установка	183	0	175
Управление проектом	237	8	357
Управление конфигурацией	30	3	38
Обучение для проекта	200	0	218
Другие	332	0	226

Таблица 9. Данные о трудоемкости

Таблица 10 содержит информацию об ошибках. Желательно не только знать, когда была обнаружена ошибка, но также когда она была зарегистрирована. Следовательно, необходимо фиксировать число обнаруженных ошибок и для каждой стадии внесения, и для каждой стадии обнаружения. Стадии обнаружения — это разнообразные экспертизы и тестирование; стадии внесения ошибок — это анализ требований, проектирование и программирование. Эта информация может быть полезной для определения потенциальных областей усовершенствования.

Ошибки						
	Экспертиза требований	Экспертиза проектирования	Экспертиза кода	Тестирование элементов	Системное тестирование	Приемочные испытания
Анализ требований	0	0	0	1	1	0
Проектирование		14	3	1	0	0
Программирование			21	48	17	6

Таблица 10. Данные об ошибках проекта

Следующая таблица содержит информацию о размере проекта. В проекте могут использоваться различные языки, поэтому здесь может присутствовать несколько элементов. Кроме того, может использоваться несколько единиц измерения размера, поэтому в таблице фиксируется единица измерения.

Размер					
Язык кода	ОС кода	СУБД кода	Аппаратная платформа кода	Единица измерения	Фактический размер кода
C#	Windows NT	MS SQL Server	PC	LOC	8081
Visual C++	Windows NT	MS SQL Server	PC	LOC	12185

Таблица 11. Данные о размере проекта

Теперь мы проиллюстрируем подход оценивания. Если в проекте применяется подход, базирующийся на вариантах использования, то основная декомпозиция выполняется в терминах вариантов использования, а не в терминах модулей. В таблице перечислены 26 вариантов использования и их сложность.

Номер варианта использования	Описание	Сложность
1	**Экран навигации**	**Сложный**
2	**Обновление личной информации**	**Средний**
3	**Добавление адреса**	**Средний**
4	**Обновление адреса**	**Сложный**
5	**Удаление адреса**	**Сложный**
6	**Добавление телефонного номера**	**Средний**
7	**Обновление телефонного номера**	**Сложный**
8	**Удаление телефонного номера**	**Сложный**
9	**Добавление адреса электронной почты**	**Средний**
10	**Обновление адреса электронной почты**	**Средний**
11	**Удаление адреса электронной почты**	**Средний**
12	**Обновление информации о занятости для компании**	**Средний**
13	**Обновление финансовой информации для компании**	**Средний**
14	**Обновление информации о счете**	**Средний**
15	**Поддержка действий со счетом**	**Сложный**
16	**Поддержка комментариев к счету**	**Простой**
17	**Просмотр предыстории информации о компании**	**Сложный**
18	**Просмотр предыстории информации о счете**	**Сложный**
19	**Просмотр предыстории уровня опции и сервисных опций**	**Простой**
20	**Просмотр предыстории действий и комментариев**	**Простой**
21	**Просмотр предыстории ролей**	**Сложный**
22	**Просмотр информации счета**	**Простой**
23	**Просмотр информации о владельцах счета**	**Сложный**

24	**Просмотр необработанных ордеров по счету**	**Сложный**
25	**Закрытие/Возобнавление счета**	**Простой**
26	**Выполнение интеллектуального обновления для бизнес-партнеров**	**Сложный**

Таблица 12. Варианты использования в проекте

В таблице 13 показаны средняя трудоемкость создания для каждого типа варианта использования и общая трудоемкость создания.

Тип варианта использования	**Трудоемкость (на вариант использования, в человеко-днях)**	**Число элементов**	**Общая трудоемкость создания (в человеко-днях)**
Простые варианты использования	1	5	5
Средние варианты использования	5	9	45
Сложные варианты использования	8	12	96
Итого			146

Таблица 13. Трудоемкость создания проекта

Для оценивания распределения трудоемкости по стадиям проекта использовано нижеследующее распределение. В таблице 14 даны оценки для каждой стадии и общая оценка.

Действие	**Оценка**	
	В человеко-днях	**В % от общей трудоемкости**
Требования	**50**	**10**
Проектирование	**60**	**12**
Создание	**146**	**29**
Комплексное тестирование	**35**	**7**
Регрессионное тестирование	**10**	**2**
Приемочные испытания	**30**	**6**
Управление проектом	**75**	**15**
Управление конфигурацией	**16**	**3**
Обучение	**50**	**10**
Другое	**40**	**6**
Оценка трудоемкости	**501 человеко-день**	**100%**

Таблица 14. Оценка трудоемкости проекта

Если использовать методологию точек вариантов использования, то как было описано ранее, сначала по вариантам использования определяются UUCP (точки нескорректированных вариантов использования): каждому простому варианту использования назначается 5 точек, варианту использования средней сложности — 10 точек, и сложному варианту использования — 15 точек. Числа простых, средних и сложных вариантов использования равняется 5, 9 и 12 со соответственно, откуда получаем:

UUCP = 5 * 5 + 9 * 10 + 12 * 15 = 295

Для того чтобы принять в расчет разнообразные факторы, сначала назначим вес факторам, связанным со сложностью технологии, и получим фактор сложности технологии. Выберем следующие значения факторов (в порядке, заданном в таблице 2): 4, 3, 5, 3, 4, 5, 5, 0, 4, 1, 2, 0, 5, получив в результате TFactor, равный 40 (8 + 3 + 5 + 3 + 4 + 2,5 + 2,5 +0+4+1+2+ 0 + 5) и TCF, равный 1,0. Далее вычислим фактор окружения. Факторам, связанным с окружением, назначим следующий вес: 3, 1, 3, 4, 5, 5, 0 и 3; в результате получим EFactor, равный 22 (4,5 + 0,5 + 3 + 2 + 5 + 10 + 0 - 3) и EF, равный 0,74. Отсюда вычислим общее число точек вариантов использования:

UCP = 295 * 1,0 * 0,74 = 218,3

Используя для трудоемкости на одну UCP обычное значение в 20 человеко-часов, получим оценку трудоемкости:

218 * 20 = 4360 человеко-часов = 499 человеко-дней (по 8,75 часов в день), или 513 человеко-дней (по 8,5 часа в день). Эти оценки близки к полученным ранее оценкам, что повышает уверенность в результатах.

Для того чтобы в полной мере воспользоваться преимуществами подхода к выполнению проекта, ориентированного на процессы, важно накапливать и использовать имущество процесса.

Инструкции	**Контрольные перечни**	**Шаблоны/Формы**
Инструкции по оценке трудоемкости и графика работ	Контрольный перечень анализа требований	Документ спецификации требований
Процедура групповой экспертизы	Контрольные перечни плана тестирования элементов и тестирования систем	Документ плана тестирования элементов
Инструкции по адаптации процесса	Контрольный перечень управления конфигурацией	Документ плана приемочных испытаний
Инструкции по оценке и мониторингу ошибок	Контрольный перечень отчета о состоянии	План управления проектом
Инструкции по измерениям и анализу данных	Контрольный перечень экспертизы требований	План управления конфигурацией
Инструкции по управлению рисками	Контрольный перечень экспертизы требований	Отчет об анализе показателей
Инструкции по трассировке требований	Контрольный перечень экспертизы плана проекта	Отчет о состоянии этапа
Инструкции по предупреждению возникновения ошибок	Контрольный перечень экспертизы кода на C++	Отчет по анализу предупреждения ошибок

Таблица 15. Инструкции, контрольные перечни и шаблоны для управления проектом

Все инструкции, контрольные перечни и шаблоны должны быть доступны в оперативном режиме и регулярно обновляются. В таблице 12 показан пример подобных материалов, имеющих отношение к управлению проектом.

4.6. Выводы по четвертой главе.

1. Для хорошего тестирования и обучения программирующих коллективов в обучающую систему должен быть заложен набор знаний, описанных в данной главе.
2. Разработаны механизмы и описания трансформаций представлений во внутреннее представление системы.
3. Разработаны механизмы пополнения знаний и методы выбора проектных решений на основе прецендентов.
4. Разработана архитектура обучающей системы процесса разработки программного обеспечения.

Заключение

В диссертационной работе получены следующие основные результаты:

1. Проанализирован ряд систем и методик их проектирования. В результате анализа выявлены положительные и отрицательные аспекты традиционных методов проектирования систем.
2. Так как в Грузии, обучение и сертификация по CMM/CMMI пока носит эпизодический характер, проводить официальное обучение получают право только инструкторы, имеющие авторизацию SEI, а для сертификации необходимо обращаться в соответствующие организации. Поэтому очень актуальна проблема создания тестирующей (когнитивной) системы для оценки возможностей организации, а также обучения принципам организационной культуры.
3. Тестирующая (обучающая) система на базе модели CMM может быть использована по крайней мере следующими 4-мя категориями специалистов:

   - группа оценки организации (Assessments Team) – для выявления сильных и слабых сторон организации;

   - группа оценки контрагентов (Evaluation Team) – для выявления риска при выборе контрагентов и для отслеживания контрактов;

   - руководители и технический персонал – для понимания деятельности по планированию и претворению в жизнь программы совершенствования процесса в их организации;

   - группа совершенствования процесса – как руководство в своих действиях.
4. В работе предложено процессы обучения описывать лабиринтной формальной комбинаторной моделью, конкретизация которой дает конкретные задачи обучения под разными углами. Для внешнего представления знаний предложено использование объектно-ориентированного, предусмотренного для анализа и проектов, универсального языка моделирования UML. Для внутреннего представления знаний рассматриваются контекстно управляемые

семантические сети, которые могут быть описаны с использованием XML-языка. В конкретных задачах обучения для преобразования структуризованных знаний предложен метод модульных трансформаций представлений. Чем более структуризованно опишутся знания, тем больше шанс успешного завершения процессов обучения и использования знаний.

5. Описана информация необходимая для заполнения базы знаний и базы данных процессов. База данных процессов содержит данные по производительности завершенных проектов. Она включает данные по рискам, трудоемкости и ее распределению, ошибкам и их распределению, размеру ПО и другим характеристикам проектов.
6. Разработаны механизмы пополнения знаний и методы выбора проектных решений на основе прецендентов.
7. Менеджер проекта может лучше спланировать свой проект, если имеет доступ к опыту, накопленному в завершенных проектах. Инфрастуктура планирования проекта помогает эффективно собирать данные и полученные уроки и делать их доступными для менеджеров проектов.
8. Рассматривается процесс разработки программного обеспечения в динамике в когнитивной системе, позволяющее сосредотачиться на динамике программного обеспечения, а также развитии и ведении систем.
9. Разработана архитектура системы обучения методам коллективной разработки программного обеспечения.

Используемая литература

1. S.T. Redwine, N. Davis. Processes to Produce Secure Software, Nat'l Cybersecurity Partnership Task Force Report, 2004; www.cyberpartnership.org/init-soft.html.

2. G.E. McGraw. DIMACS Workshop on Software Security, IEEE Security & Privacy, vol. 1, no. 2, 2003.

3. N. Davis, J. Mullaney. The Team Software Process in Practice: A Summary of Recent Results, tech. report CMU/SEI-2003-TR-014, Software Eng. Inst., Carnegie Mellon Univ., September 2003.

4. A. Hall, R. Chapman. Correctness by Construction: Developing a Commercial Secure System. IEEE Software, vol. 19, no. 1, 2002.

5. R. Linger, S. Powell. Developing Secure Software with Cleanroom Software Engineering. Cybersecurity Summit Task Force Subgroup on Software Process, February 2004.

6. H. Mills, R. Linger. Cleanroom Software Engineering. Encyclopedia of Software Engineering, 2nd ed., J.Marciniak, ed., John Wiley & Sons, 2002.

7. S. Prowell et al. Cleanroom Software Engineering: Technology and Process, Addison Wesley, 1999.

8. J. Herbsleb et al. Benefits of CMM-Based Software Process Improvement: Initial Results, tech. report CMU/SEI-94-TR-013, Software Engineering Institute, Carnegie Mellon University, 1994.

9. D.R. Goldenson, D.L. Gibson. Demonstrating the Impact and Benefits of CMMI, special report CMU/SEI-2003-SR-009, Software Eng. Inst., Carnegie Mellon Univ., 2003.

10. J. Saltzer, M. Schroeder. The Protection of Information in Computer Systems. Proc. IEEE, vol. 63, no. 9, 1975.

11. P. Neumann. Principles Assuredly Trustworthy Composable Architectures: Emerging Draft of the Final Report, tech. report for SRI Project 11459, as part of DARPA's Composable High-Assurance Trustworthy Systems (CHATS) programs, to be published Sept. 2004.

12. J. Viega, G. McGraw. Building Secure Software: How to Avoid Security Problems the Right Way, Addison Wesley, 2001.

13. G. Hoglund and G. McGraw. Exploiting Software: How to Break Code, Addison-Wesley, 2004.

14. J. Barnes. High Integrity Software: The SPARK Approach to Safety and Security, Addison Wesley, 2003.

15. W.R. Bush, J.D. Pincus, D.J. Siela. A Static Analyzer for Finding Dynamic Programming Errors. Software Practice and Experience, vol. 30, June 2000.

16. S.J. Vaughn. Building Better Software with Better Tools. Computer, vol. 36, no. 9, September 2003.

17. Raymond J. Madachy. Software Process Dynamics. Published by John Wiley & Sons, Inc., Hoboken, New Jersey. vol. 601. 2007.

18. David M. Raffo, Wayne Wakeland. Moving Up the CMMI Capability and Maturity Levels Using Simulation. Technical report CMU/SEI-2008-TR-002, January 2008.

19. Панкаж Джалота. Управление программным проектом на практике. Издательство "Лори", 2005, 225 с.

20. Руководство к Своду знаний по управлению проектами (Руководство PMBOK®), третье издание, Project Management Institute, Inc., 388, Американский национальный стандарт ANSI/PMI 99-001-2004.

21. З.Б. Босикашвили, О.Б. Шония, Л.Г. Бежанишвили. Автоматизированное проектирование учебных курсов. Московский Государственный Университет им. М.В. Ломоносова. Труды V Московской международной конференции по исследованию операций. 2007.

22. ლ. ბეჟანიშვილი. შესაძლებლობათა სიმწიფის მოდელის შესახებ. საქართველოს ტექნიკური უნივერსიტეტი. შრომები. მართვის ავტომატიზებული სისტემები №1(2), ISSN 1512-3979, თბილისი, 2007 წ.

23. З.В. Босикашвили, О.Б. Шония, Л.Г. Бежанишвили, Д.Ш. Капанадзе. Автоматизированное проектирование электронных учебных курсов. Научно-технический журнал «Информационные технологии в проектировании и производстве», №3 2007, статья, ФГУП «ВИМИ» Моква.

24. Дж.Д. Мейер Джейсон Тейлор Алекс Макман Прашант Бансод Кевин Джонс. Коллективная разработка с использованием Visual Studio Team Foundation Server. 2007, Microsoft Corporation, 575 с.

25. Technical Report CMU/SEI-87-TR-23. Preliminary Version. September 1987. "A Method for Assesing the Software Engineering Capability of Contractor". W.S.Humphrey, W.L.Sweet. Software Engineering Institute. Carnegie Mellon University. Pitsburg, Pensilvania 15213

26. Rubin, Howard. "Software Process Maturity". American Programmer, January, 1991.

27. Humphrey, Watts. Managing The Software Process. Reading, MA: Addison-Wesley, 1989

28. Jones, T.C., Applied Software Measurement, McGraw-Hill, 1991.

29. Dreger, J.B., Function Point Analysis, Prentice-Hall, 1989.

30. Серия стандартов ISO 9000

31. ANSI/IEEE Std 983-1986, Software Quality Assurence Planning.

32. ANSI/IEEE Std 1028-1988, Standard for Software Reviews and Audits.

33. Cleal, David, Knowledge based systems: Implications for human-computer interfaces. Horwood. Chichester, 1988, 400p.

34. Krzysztof Czarnecki, Simon Helsen. Classification of Model Transformation Approaches. University of Waterloo, Canada, 2003.

35. Jernej Kovse, Theo Härder. Generic XMI-Based UML Model Transformations. ZDNet UK Whitepapers, 2002.

36. XSL Transformations (XSLT) v1.0. W3C Recommendation, Nov. 1999. http://www.w3.org/TR/xslt

37. D. Chamberlin. XQuery: An XML query language. IBM systems journal, no 4, 2002.

38. Гогичаишвили Г.Г., Бежанишвили Л.Г., Коркия И.Л. Формальные средства синтеза текста в экспертной системе “ЭСВТЭ”. Тезисы докладов VI Всесоюзной научно-технической школы “Интеллектуальные банки данных -90”, Бакуриани.

39. Бежанишвили Л.Г. Формальные средства синтеза текста в системах общения (ტექსტის სინთეზის ფორმალური საშუალებანი ურთიერთობის სისტემებში). Труды ГПИ #4(346), УДК 681.327, Тбилиси, 1989, статья.